Hybrid Verified Decoding: Learning to Allocate Verification in Speculative Decoding

Anonymous ACL submission

Abstract

Large Language Model (LLM) generation remains expensive because autoregressive decoding calls the model once for each new token. Speculative decoding reduces this cost by drafting multiple tokens and verifying them with the target model in one step, but its speedup depends on how many drafted tokens are accepted. Parameter-free draft sources can propose long continuations at low cost in structured and agentic workloads, yet a cache match that looks promising at one generation step may have low payoff at the next. We propose Hybrid Verified Decoding, which predicts the accepted length of a cache draft before verification and uses this payoff estimate to choose between cache verification and a model-based drafter. Across three LLMs and sixteen datasets, Hybrid Verified Decoding is especially effective on agentic workflows, where it outperforms EAGLE3 in every setting with a 2.73x average speedup. Our analysis shows how prompt structure creates cache opportunities, how high-payoff cache drafts concentrate in a small part of the draft space, and how payoff-guided selection reduces sequential decoding work, pointing to runtime draft selection as a promising direction for speculative decoding.

1 Introduction

Large language model (LLM) generation is limited by autoregressive decoding: each new token requires a forward pass over the current prefix and key-value (KV) cache. Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) reduces this sequential cost through a draft-then-verify interface: a lower-cost source proposes candidate tokens, and the target LLM itself verifies which prefix can be committed. Its speedup depends on how many proposed tokens are accepted at each verification step.

Recent speculative decoding methods instantiate the draft source in different ways. Model-based drafters, such as the EAGLE family (Li et al., 2024b,a, 2025), improve draft reliability through learned proposal models. Lookup, retrieval, and cache-based methods (Saxena, 2023; He et al., 2024; Fu et al., 2024; Oliaro et al., 2026) reuse continuations from the prompt, generation history, or external corpora at low proposal cost. Such reuse is common in structured and agentic workloads, where recurring context can support long cache drafts. The payoff of a cache draft, however, depends on the current generation step. A local match can produce a long draft, but a similar prefix may require different arguments, a different edit goal, or a different piece of evidence. When the target model accepts only a short prefix, the cache draft turns a cheap proposal into wasted verification.

We study this per-step decision in verified decoding and propose Hybrid Verified Decoding, which estimates the accepted length of the current cache draft before verification. If the predicted payoff is high, the cache draft is submitted for verification. Otherwise, the system uses a model-based drafter. The target model remains responsible for committed tokens, while the learned component allocates verification work across draft sources and generation steps. We train the payoff predictor from trace replay and evaluate the resulting decoder across three LLMs and sixteen datasets. Our analysis investigates the full path from input structure to cache opportunity, verification payoff, predictor selection, execution cost, and prompt-template sensitivity. Our contributions are:

- We formulate state-dependent draft selection in verified decoding as payoff-guided verification allocation, with accepted length as the payoff variable.
- We propose Hybrid Verified Decoding, which predicts the accepted length of the current cache draft and chooses between cache-based drafting and a model-based drafter before verification.

- We present a replay-based training procedure that derives payoff labels from generation traces, enabling lightweight supervision for the runtime predictor.
- We evaluate our method across three LLMs, sixteen datasets, and four baselines approaches spanning model-based, cache-based, rule-based hybrid, and greedy decoding.
- We provide a systematic analysis of cache opportunity, verification payoff, predictor behavior, execution cost, and prompt-template sensitivity, and implement a vLLM serving prototype for deployment-oriented validation.

2 Related Work

Parameter-Based Speculative Decoding. Parameter-based methods improve speculative decoding by adding learned proposal modules. The EAGLE family (Li et al., 2024b,a, 2025) uses model representations to construct stronger drafts. Medusa (Cai et al., 2024) and PARD (An et al., 2025) expand candidate generation through multiple heads or parallel draft modules. These methods invest additional parameters and training to improve draft reliability or candidate diversity. Hybrid Verified Decoding uses a model-based drafter as its fallback path and focuses on the verification decision made before that fallback is used.

Parameter-Free Speculative Decoding. Parameter-free methods propose draft tokens without training a separate proposal model. Prompt Lookup Decoding (Saxena, 2023), REST (He et al., 2024), Lookahead Decoding (Fu et al., 2024), and SuffixDecoding (Oliaro et al., 2026) reuse continuations from the prompt, generation history, or external datastores. This line of work lowers proposal cost and fits workloads with repeated context, structured formats, or recurring intermediate states. A cache or retrieval match still leaves an open runtime question: how many of the proposed tokens will the target model accept at the current generation step? Our method builds on parameter-free drafting by estimating this accepted length before verification and using it to decide whether the current cache draft should be verified.

Adaptive Speculation. Adaptive speculation methods adjust the behavior of speculative decoding at runtime. AdaEAGLE (Zhang et al., 2024) adapts draft structures, Nightjar (Li et al., 2026) and AdaSpec (Huang et al., 2026) adjust serving-time speculation, ECHO (Hu et al., 2026) and TALON (Liu et al., 2026a) control token trees or candidate budgets, and MARS (Song et al., 2026) modifies verification behavior. These methods show that speculation benefits from runtime control over draft length, tree shape, budget, or verification criteria. They leave an important gap for parameter-free drafts: after a cache match is found, the system still needs to decide whether the proposed continuation is worth verification. Hybrid Verified Decoding addresses this gap by predicting the accepted length of the proposed cache draft before verification and using that payoff estimate to choose between cache verification and a model-based drafter at each generation step.

3 Method

Speculative decoding accelerates generation when one verification step accepts multiple draft tokens. A model-based drafter improves draft reliability through training. In contrast, a parameter-free draft source reuses continuations from existing context at low proposal cost, matching the repeated structure often found in repetitive, structured, and agentic workloads. However, the quality of such drafts depends strongly on the current decoding state: an incorrect reuse may be rejected early by the target model, turning a low-cost proposal into additional verification overhead. We formulate this challenge as per-step payoff prediction: before running verification, we predict how many tokens in the current parameter-free draft are likely to be accepted by the target model, and use this prediction to decide whether to verify the draft or fall back to a model-based drafter. Figure 1 summarizes this workflow.

3.1 Verified Drafting as a Common Interface

Consider a decoding step t with current prefix $x_{1:t}$, consisting of the input prompt and the tokens generated so far. A draft source D proposes a length-k continuation conditioned on this prefix:

$$d_t = (\hat{x}_{t+1}, \ldots, \hat{x}_{t+k}) = D(x_{1:t}).$$

The target model then verifies d_t and accepts a prefix of the proposal. We denote the number of accepted draft tokens by

$$a_t(d_t) \in \{0, \ldots, k\}.$$

Target verification also returns the token block com-

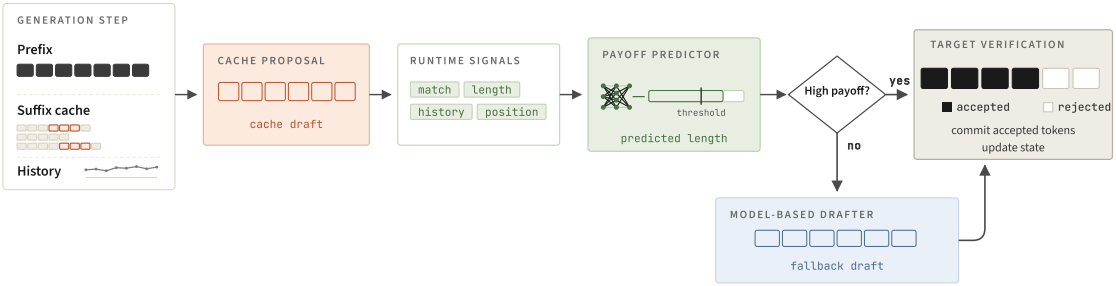


Figure 1: Overview of Hybrid Verified Decoding.

mitted after this step:

$$(a_t(d_t), z_t) = \mathrm{Verify}(x_{1:t}, d_t; M).$$

The first $a_t(d_t)$ tokens of z_t match the draft prefix $(\hat{x}_{t+1}, \ldots, \hat{x}_{t+a_t(d_t)})$. If verification rejects the draft before completion, z_t also contains the target-model token at the rejection point, so decoding continues under verification.

The accepted length $a_t(d_t)$ directly determines the benefit of a draft. A large accepted length lets one verification step commit multiple draft tokens. A small accepted length, especially when it is close to zero, leaves the system with the cost of proposing and verifying a draft but little speculative progress. Draft selection should therefore account for the expected accepted length under the current decoding state, since this quantity determines whether the cost of producing and verifying the draft can be amortized.

Our method leaves the target model and the verification rule unchanged. The only runtime choice is which candidate continuation is sent for verification. This makes draft selection a payoff estimation problem: *can we predict, before verification, whether the current draft will yield a sufficiently long accepted prefix?*

3.2 Cache-Based Drafting

Parameter-free drafting constructs proposals by matching recurring token patterns in the input and generation history, without training a separate drafter. We instantiate this source with a suffix cache, following SuffixDecoding (Oliaro et al., 2026). The cache indexes suffix matches over the input prompt and the tokens committed so far, and returns candidate continuations from matched spans. Given the current prefix $x_{1:t}$, the cache-based drafter $D_{\mathcal{C}}$ returns

$$d_t^c = D_{\mathcal{C}}(x_{1:t}; \mathcal{C}),$$

where $\mathcal{C}$ denotes the suffix cache and d_t^c is the cache draft.

This mechanism requires no additional model forward pass. Structured and agentic workloads often contain recurring patterns across prompts and generations, creating opportunities for long cache drafts at low proposal cost. The same property also creates runtime risk: a local suffix match does not guarantee that the following continuation remains valid in the current state. For example, repeated tool-call structure may still require different arguments. If such a continuation is sent for verification, the target model may accept only a short prefix. Cache-based drafting exposes a direct tradeoff: it can produce long drafts at low proposal cost, but verification is worthwhile only when the draft is likely to yield a sufficiently long accepted prefix.

3.3 Learning When to Verify a Cache Draft

The value of a cache draft depends on the prefix length accepted during verification, which is unknown prior to verification. We denote it by

$$y_t = a_t(d_t^c),$$

where d_t^c is the cache draft at decoding step t. A large y_t lets one verification step advance the generation by multiple tokens. A small y_t turns the cache proposal into verification overhead. Our goal is to estimate y_t from the current decoding state before verification.

For each cache draft, we compute a runtime feature vector

$$\phi_t = \Phi(x_{1:t}, d_t^c, \mathcal{C}, h_t),$$

where h_t summarizes recent decoding history. The features describe signals available before verification: the suffix match, the draft length, the current decoding position, recent cache-draft behavior, and lightweight token-structure indicators (see Appendix B for details). The payoff predictor maps

these features to an accepted-length estimate,

$$\hat{y}_t = g_\theta(\phi_t).$$

We instantiate g_θ as a lightweight multilayer perceptron over runtime features. A threshold τ turns the estimate into a verification decision: cache drafts with $\hat{y}_t \geq \tau$ are sent to verification, while lower-scoring states use a model-based drafter.

We derive training labels by treating each token position in a target-model generation trace as a decoding state. For a position t, the visible prefix is $x_{1:t}$. The suffix cache built from this prefix returns a cache draft d_t^c. The following tokens $x_{t+1:}$ under greedy decoding provide the target continuation for that state. The accepted-length label is the longest prefix of the cache draft that agrees with this continuation:

$$y_t = \max\left\{\ell \in \{0, \ldots, |d_t^c|\} : d_{t,1:\ell}^c = x_{t+1:t+\ell}\right\}.$$

We train g_θ with accepted-length regression:

$$\min_\theta \sum_t \left(g_\theta(\phi_t) - y_t\right)^2.$$

3.4 Hybrid Verified Decoding

For each decoding step, Hybrid Verified Decoding obtains a cache draft d_t^c and computes its predicted accepted length $\hat{y}_t = g_\theta(\phi_t)$. The draft sent to verification is selected by

$$d_t = \begin{cases} d_t^c, & \text{if } \hat{y}_t \geq \tau, \\ D_m(x_{1:t}), & \text{otherwise,} \end{cases}$$

where D_m denotes the model-based drafter. Algorithm 1 summarizes the resulting decoding loop. The selected draft d_t is verified by the target model under the standard speculative decoding verification rule:

$$(a_t(d_t), z_t) = \text{Verify}(x_{1:t}, d_t; M),$$

where M is the target model. The committed block z_t is appended to the output, the suffix cache is updated with z_t, and decoding proceeds.

Algorithm 1 Hybrid Verified Decoding

Require: target model M, model-based drafter D_m, suffix cache $\mathcal{C}$, payoff predictor g_θ, threshold τ

Initialize the current prefix x with the input prompt
Initialize $\mathcal{C}$ from cache-warmup traces and the input prompt
Initialize the recent-history summary h
while generation is not finished **do**
  $d^c \leftarrow D_{\mathcal{C}}(x; \mathcal{C})$
  $\phi \leftarrow \Phi(x, d^c, \mathcal{C}, h)$, $\hat{y} \leftarrow g_\theta(\phi)$
  if $\hat{y} \geq \tau$ **then**
    $d \leftarrow d^c$
  else
    $d \leftarrow D_m(x)$
  end if
  $a, z \leftarrow \text{Verify}(x, d; M)$
  Append the committed block z to x
  Update $\mathcal{C}$ and h with z
end while

4 Experiments

4.1 Experiment Setup

We evaluate our method on three instruction-tuned target models: Qwen3-8B, Qwen3-4B (Yang et al., 2025), and Llama3.1-8B (Grattafiori et al., 2024). Each target model is paired with its corresponding EAGLE3 checkpoint as the model-based fallback drafter. All end-to-end timing runs are measured on H100 GPUs with deterministic decoding.

Our evaluation covers 16 datasets across six workload families: agentic and workflow editing (Delegate-52, InstructEdit/FineEdit), repository-level and standalone coding (RepoBench, SWE-bench OpenHands, Magicoder, MBPP), structured generation and tool use (BFCL, Spider), multi-hop QA (HotpotQA, MuSiQue, 2WikiMultiHopQA), long-context document tasks (InfiniteBench, CNN/DailyMail, GovReport), and open-ended instruction/chat (Alpaca, MT-Bench). These workloads differ in input length, structural constraints, context reuse, and output format.

All methods are evaluated inside the generation loop of the official EAGLE repository[1] and use the chat template associated with each target model. We report run-level throughput as generated tokens per second, and define speedup as the throughput ratio relative to the corresponding baseline.

4.2 Baselines

We compare our hybrid verified decoding method from Section 3.4 with four baselines: target-only decoding, model-based speculative decoding, cache-based drafting, and frequency-based hybrid decoding.

[1] https://github.com/SafeAILab/EAGLE

		Qwen3-8B				Qwen3-4B				Llama3.1-8B			
Group	Dataset	G	E3	SD	FH	G	E3	SD	FH	G	E3	SD	FH
Agentic	Delegate-52	3.25	1.44	2.14	1.06	3.40	1.90	1.64	1.18	3.46	3.38	0.81	1.24
	InstructEdit/FineEdit	3.57	1.51	2.30	1.32	4.21	2.67	1.72	1.22	7.56	5.49	1.15	1.20
Code	RepoBench	2.54	1.09	2.15	1.15	3.57	1.56	1.44	0.81	2.12	2.99	1.09	1.26
	SWE-bench OpenHands	3.70	1.09	3.21	1.28	2.50	1.08	1.52	0.88	3.31	2.11	2.60	1.26
	Magicoder	3.31	1.08	3.79	1.34	1.73	1.02	1.82	0.87	5.86	2.51	4.28	1.27
	MBPP	4.49	1.17	4.78	1.54	2.73	0.90	2.11	1.01	3.68	0.99	4.65	1.05
Structured	BFCL	2.31	0.98	3.13	1.27	2.19	0.98	2.47	1.13	1.02	0.90	1.32	1.01
	Spider	3.12	0.97	2.98	1.33	1.56	1.01	1.79	0.88	4.46	1.85	3.61	1.24
Multi-hop QA	HotpotQA	2.57	0.95	3.77	1.37	2.03	0.81	1.97	1.02	2.23	0.93	2.70	0.96
	MuSiQue	3.55	0.95	2.88	1.17	2.12	1.18	1.35	1.05	3.00	3.22	1.15	1.97
	2WikiMultiHopQA	2.73	0.96	3.28	1.40	2.01	1.19	1.65	0.98	2.52	1.04	2.49	0.87
Long-context	InfiniteBench	5.03	2.42	3.04	1.21	5.53	2.51	2.40	1.16	2.23	2.84	1.47	2.94
	CNN/DailyMail	2.33	0.94	3.85	1.34	1.79	0.99	2.75	1.02	2.43	0.94	3.83	1.04
	GovReport	2.03	0.95	3.36	1.18	1.52	1.01	2.08	1.02	0.95	1.31	0.98	1.15
Open-ended	Alpaca	2.73	0.97	3.84	1.40	2.06	0.85	2.40	0.83	3.14	0.96	5.68	1.05
	MT-Bench	3.34	0.98	4.68	1.41	1.59	0.99	2.06	0.87	3.24	1.18	4.76	1.10
Avg. across datasets		3.16	1.15	3.32	1.30	2.53	1.29	1.95	0.99	3.20	2.04	2.66	1.29

Table 1: Dataset-level end-to-end speedup of Hybrid Verified Decoding over four baselines across three target models. G, E3, SD, and FH denote Greedy Decoding, EAGLE3, SuffixDecoding, and Frequency-Based Hybrid Decoding. Cell shading follows speedup magnitude. The final row reports the unweighted dataset average.

Greedy Decoding. Greedy decoding is the target-only reference. It uses neither an EAGLE3 drafter nor a suffix cache, and provides the wall-clock throughput without speculative decoding.

EAGLE3. EAGLE3 (Li et al., 2025) is a strong model-based speculative decoding baseline. It uses the EAGLE3 checkpoint corresponding to the target model to produce drafts, which are then verified by the same target model.

SuffixDecoding. SuffixDecoding (Oliaro et al., 2026) is the cache-based drafting baseline. It proposes drafts from a suffix cache and submits them for verification.

Frequency-Based Hybrid Decoding. Frequency-Based Hybrid Decoding is a rule-based hybrid baseline. It uses the same cache-first, EAGLE3-fallback structure as our method, with the verification decision based on the frequency-derived cache score from SuffixDecoding (Oliaro et al., 2026).

4.3 Main Results

Table 1 reports end-to-end speedup for Hybrid Verified Decoding across 16 datasets and three target models. Across the 48 model-dataset pairs, our method is faster than Greedy Decoding in 47 cases and faster than SuffixDecoding in 46 cases, with average speedups of 2.97x and 2.64x, respectively. These comparisons support the central design motivation: cache drafts can substantially reduce sequential decoding steps when accepted by the target model, while low-payoff cache drafts should be filtered before they introduce additional verification cost.

Agentic and workflow-style workloads make this trade-off especially concrete. On Delegate-52 and InstructEdit/FineEdit, Hybrid Verified Decoding outperforms both EAGLE3 and Frequency-Based Hybrid Decoding for all three target models. These workloads repeatedly expose tool-call structure, editing context, repository state, and intermediate outputs, creating opportunities for long cache drafts. At the same time, each step remains state-dependent: a similar tool call or editing context may require a different argument or a different target change. The learned payoff predictor addresses this state-dependent decision by estimating, before verification, whether the current cache draft is likely to yield a sufficiently long accepted prefix.

The model-level differences follow the cost trade-off identified in recent analyses of speculative decoding (Liu et al., 2026b; Yan et al., 2025): speedup depends jointly on accepted length, draft-

ing cost, verification cost, and system overhead, while acceptance behavior varies across requests, token positions, and datasets. On Qwen3-4B, Hybrid Verified Decoding still improves consistently over Greedy Decoding and SuffixDecoding, but its margin over Frequency-Based Hybrid Decoding is smaller than on Qwen3-8B and Llama3.1-8B. Smaller target models reduce the relative cost of verification, so the benefit of a learned switching policy becomes more visible when it clearly increases accepted length or avoids wasted cache verification. The results show that Hybrid Verified Decoding is strongest when cache opportunities appear frequently, yet draft correctness still depends on the current decoding state. Section 5 examines this pattern from input structure to cache opportunity, payoff prediction, execution cost, hardware behavior, serving integration, and prompt-template sensitivity.

5 Analysis

5.1 What Makes a Task Cache-Friendly?

The prompt-side view in Figure 2(a) shows that cache opportunity varies substantially across workload families. Agentic/workflow editing and code/repository workloads expose dense structural signals, including repeated entities, file paths, delimiters, and code/function patterns. Structured generation workloads show stronger schema-like and tool-oriented formats, while long-context workloads provide extended context and recurring evidence. Open-ended chat prompts are sparser along these structural dimensions. These patterns suggest that cache-based drafting benefits from inputs that repeatedly expose format, entities, contextual constraints, or workflow state, since these structures provide matchable cues for future continuations.

The runtime cache-draft view in Figure 2(b) shows a similar trend. Workloads with stronger prompt-side reuse structure tend to trigger cache drafts more often, with agentic/workflow editing and long-context tasks occupying the higher-selection region and open-ended chat tasks remaining near the lower-selection region. This correspondence indicates that cache opportunity comes from recurring structure, stable format, context recurrence, and workflow state. Prompt-side structure explains where cache drafts originate. The accepted length of a selected draft still depends on whether the continuation matches the current decoding state. We therefore next examine how cache opportunity turns into verification payoff.

5.2 When Do Cache Drafts Become Worth Verifying?

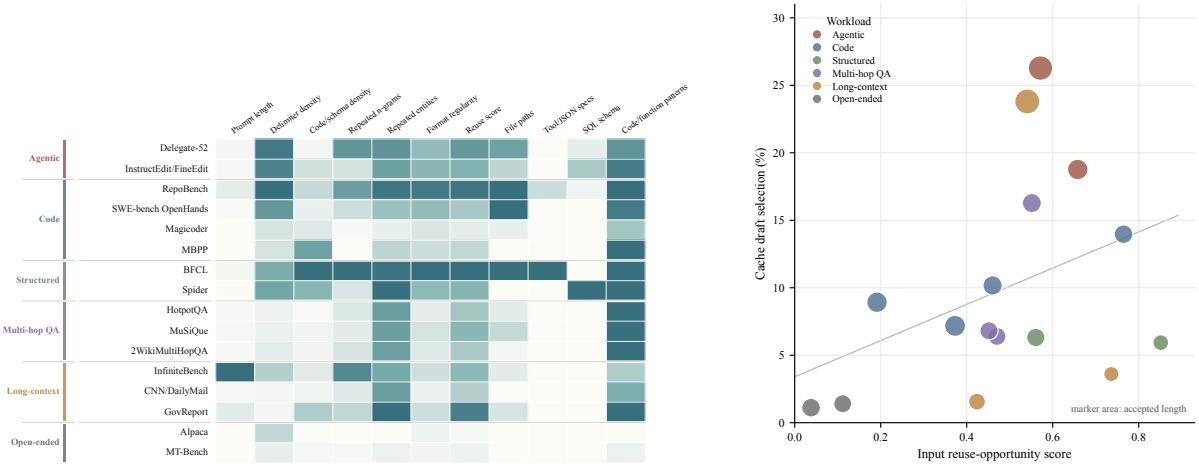
Figure 3 shows the verification payoff after a cache draft has been selected. The payoff distribution differs sharply across workload families. Agentic, code, and long-context workloads contain a larger share of high-payoff cache drafts, and their selected cache drafts reach longer mean accepted lengths. In these workloads, cache matches more often extend into a long accepted prefix. Structured and multi-hop QA workloads contain a larger fraction of low-payoff and marginal-payoff drafts. Open-ended workloads still contain some high-payoff cache drafts, but they produce far fewer selected cache drafts overall.

This distribution separates cache opportunity from verification payoff. A cache match can provide a candidate continuation, but its value is determined by how many of its tokens the target model accepts in the current decoding state. High-payoff drafts amortize verification cost over multiple committed tokens. Low-payoff drafts add verification work while advancing generation by only a short prefix. Hybrid Verified Decoding therefore estimates the payoff of the current cache draft before verification and reserves cache verification for drafts with sufficient expected payoff. The next section analyzes whether the payoff predictor identifies these high-payoff drafts.

5.3 Can the Predictor Identify High-Payoff Drafts?

We enumerate the proposed cache drafts returned by the suffix cache on test prompts, and compute their realized accepted length y_t from the target-model trace. Drafts with $y_t \geq 6$ are treated as oracle high-payoff drafts. This oracle label is used for analysis and is not available during runtime decoding. The payoff predictor scores the same drafts with $\hat{y}_t$, and drafts with $\hat{y}_t \geq 6$ form the predictor-selected set. We then use oracle high-payoff drafts as the positive set and report the precision and recall of predictor selection. Appendix G defines the corresponding metrics.

Table 2 shows that cache-draft payoff is concentrated in a small portion of decoding states. Across the three target models, 4.8%, 6.6%, and 8.9% of proposed cache drafts meet the oracle high-payoff criterion. The predictor selects a similarly sized set, covering 3.8%, 5.4%, and 8.1% of proposed



(a) Prompt-side structure.

(b) Cache-draft selection.

Figure 2: Prompt-side cache opportunity and runtime cache-draft behavior. Panel (a) reports normalized input structure features by workload family. Darker cells indicate larger feature values after column-wise normalization. Panel (b) reports cache-draft selection rate against prompt-side reuse opportunity, with marker area scaled by mean accepted length.

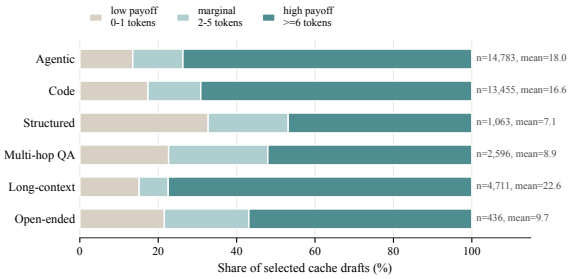


Figure 3: Verification payoff distribution of selected cache drafts by workload family. Bars group drafts by realized accepted prefix length, and annotations report selected draft count and mean accepted length.

Model	$P(y_t \geq 6)$	$P(\hat{y}_t \geq 6)$	Precision	Recall
Qwen3-8B	4.8	3.8	78.2	61.4
Qwen3-4B	6.6	5.4	82.9	67.8
Llama3.1-8B	8.9	8.1	87.6	79.8

Table 2: Payoff-prediction quality on proposed cache drafts from test prompts. All values are percentages. Oracle high-payoff drafts satisfy $y_t \geq 6$, and predictor-selected drafts satisfy $\hat{y}_t \geq 6$.

drafts. This selected set closely matches the oracle high-payoff set: precision reaches 78.2%, 82.9%, and 87.6%, while recall reaches 61.4%, 67.8%, and 79.8%.

The analysis reframes the cache decision as state selection. Cache drafts create many verification opportunities, but the payoff is concentrated in states where the continuation aligns with the current generation context. The payoff predictor identifies much of this high-payoff region before verification, allowing the hybrid decoder to focus cache verification on drafts that are more likely to amortize verification cost.

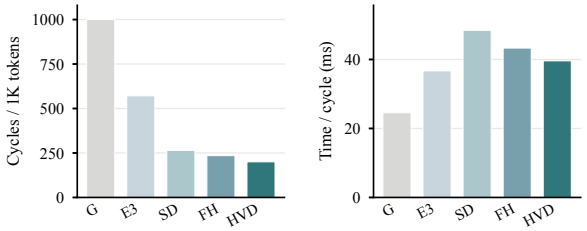


Figure 4: Execution breakdown on Llama3.1-8B. HVD denotes Hybrid Verified Decoding. Left: decoding cycles per 1K output tokens. Right: wall-clock time per decoding cycle. Lower values are better in both panels.

5.4 How Do Draft Decisions Translate into Speedup?

Execution Breakdown. Figure 4 breaks down the execution path on Llama3.1-8B. Hybrid Verified Decoding reduces the number of decoding cycles needed per 1K output tokens from 571 for EAGLE3, 264 for SuffixDecoding, and 235 for Frequency-Based Hybrid Decoding to 200. The learned rule converts cache-draft selection into fewer sequential verification steps.

The lower cycle count is accompanied by controlled per-cycle cost. Our method runs at 39.6 ms per cycle, lower than SuffixDecoding (48.4 ms) and Frequency-Based Hybrid Decoding (43.3 ms), and close to EAGLE3 (36.7 ms). The speedup

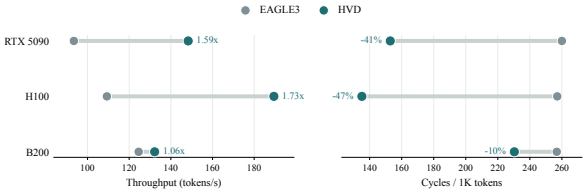


Figure 5: Cross-GPU case study on Qwen3-8B InfiniteBench. The experiment uses the same four prompts, prompt token limit, and decoding budget on RTX 5090, H100, and B200. Left: end-to-end throughput. Right: decoding cycles per 1K generated tokens.

Dataset	vs. EAGLE3	vs. SuffixDecoding
MT-Bench	2.31x	1.21x
CNN/DailyMail	3.56x	1.16x

Table 3: vLLM serving case study on Qwen3-8B.

comes from reducing sequential decoding work while keeping verification-step cost controlled.

This breakdown connects draft selection to system-level speed. Hybrid Verified Decoding reduces the number of verification cycles while keeping the cost of each cycle close to the model-based drafter and below the cache-only and rule-based paths. The throughput gain comes from an execution shift: verification work is spent on decoding states that commit more tokens per cycle.

Cross-GPU Behavior. We use a small cross-GPU case study to compare EAGLE3 and Hybrid Verified Decoding under the same setting: Qwen3-8B on InfiniteBench with the same four prompts and decoding budget. Figure 5 reports both end-to-end throughput and cycles per 1K generated tokens. Relative to EAGLE3, Hybrid Verified Decoding reaches 1.59x and 1.73x speedup on RTX 5090 and H100, where cycles per 1K tokens drop from 260.0 to 152.9 and from 257.1 to 135.2, respectively. On B200, the speedup is 1.06x, with cycles per 1K tokens decreasing from 256.8 to 230.3. These results suggest that Hybrid Verified Decoding is particularly effective on consumer and standard server GPUs, where reducing verification cycles can translate into substantial throughput gains.

vLLM Prototype. We further validate Hybrid Verified Decoding in the vLLM (Kwon et al., 2023) serving stack. The serving setting tests whether payoff-guided switching remains effective under serving-time scheduling and execution. As shown in Table 3, Hybrid Verified Decoding is 2.31x and 3.56x faster than EAGLE3, and 1.21x and 1.16x faster than SuffixDecoding, on MT-Bench and CNN/DailyMail respectively. These results show that payoff-guided switching extends beyond the offline generation loop and delivers clear throughput gains in a mainstream serving system. We provide implementation details for the vLLM prototype in Appendix E.

Prompt construction	Method	Mean accepted tokens ↑	Cycles / 1K tokens ↓	Throughput ↑
Template omitted	EAGLE3	1.20	455	55.6
Official template	EAGLE3	4.21	192	134.4
Template omitted	Hybrid Verified Decoding	7.32	120	194.2

Table 4: Prompt-template diagnostic on Llama3.1-8B MT-Bench. Template omitted uses the same request content without the Llama chat template. Official template applies the Llama chat template before tokenization. Throughput is tokens per second.

5.5 How Does Prompt Construction Shape Draft Payoff?

We run a controlled prompt-template diagnostic on Llama3.1-8B MT-Bench, changing only the prompt construction while keeping the rest of the setup fixed. Table 4 isolates prompt-template sensitivity. When the Llama chat template is omitted, EAGLE3 reaches only 1.20 mean accepted tokens and requires 455 cycles per 1K tokens. Applying the official template raises mean accepted tokens to 4.21 and reduces cycles to 192, showing how strongly a learned model-based drafter can depend on prompt construction. Under the same template-omitted setting, Hybrid Verified Decoding reaches 7.32 accepted tokens and 120 cycles per 1K tokens. This result highlights the value of runtime draft selection: when the model-based drafter loses payoff under a prompt-construction mismatch, cache drafts can still provide high-payoff verification paths.

6 Conclusion

In this paper, we introduce Hybrid Verified Decoding, a payoff-guided method for state-dependent draft selection in speculative decoding. It predicts the accepted length of a cache draft before verification and uses this estimate to choose between cache verification and a model-based drafter. Across target models and task families, our experiments show improved end-to-end throughput and concentrated cache verification on high-payoff decoding states. These results point to a broader direction for speculative decoding, where runtime systems decide which drafts merit verification while continuing to build stronger drafters.

Limitations

Our experiments cover two model families, Qwen and Llama, and instantiate hybrid selection as a choice between suffix-cache drafting and EAGLE3 model-based drafting. We have not systematically evaluated additional model families or broader draft-source combinations, such as multi-head drafters.

References

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

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

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

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. Break the sequential dependency of LLM inference using lookahead decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 14060–14079. PMLR.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason Lee, and Di He. 2024. REST: Retrieval-based speculative decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1582–1595, Mexico City, Mexico. Association for Computational Linguistics.

Xinyi Hu, Yuhao Shen, Baolin Zhang, Hengxin Zhang, Jun Dai, Shuang Ge, Lei Chen, Yue Li, and Mingcheng Wan. 2026. Echo: Elastic speculative decoding with sparse gating for high-concurrency scenarios. *Preprint*, arXiv:2604.09603.

Kaiyu Huang, Hao Wu, Zhubo Shi, Han Zou, Minchen Yu, and Qingjiang Shi. 2026. Adaspec: Adaptive speculative decoding for fast, slo-aware large language model serving. In *Proceedings of the 2025 ACM Symposium on Cloud Computing*, SoCC '25, page 361–374, New York, NY, USA. Association for Computing Machinery.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR.

Rui Li, Zhaoning Zhang, Libo Zhang, Huaimin Wang, Xiang Fu, and Zhiquan Lai. 2026. Nightjar: Dynamic adaptive speculative decoding for large language models serving. *Preprint*, arXiv:2512.22420.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024a. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7421–7432, Miami, Florida, USA. Association for Computational Linguistics.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024b. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Forty-first International Conference on Machine Learning*.

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

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

Xiaoxuan Liu, Jiaxiang Yu, Jongseok Park, Ion Stoica, and Alvin Cheung. 2026b. Speculative decoding: Performance or illusion? *Preprint*, arXiv:2601.11580.

Gabriele Oliaro, Zhihao Jia, Daniel F Campos, and Aurick Qiao. 2026. Suffixdecoding: Extreme speculative decoding for emerging AI applications. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Apoorv Saxena. 2023. Prompt lookup decoding.

Jingwei Song, Xinyu Wang, Hanbin Wang, Xiaoxuan Lei, Bill Shi, Shixin Han, Eric Yang, Xiao-Wen Chang, and Lynn Ai. 2026. Mars: Unleashing the power of speculative decoding via margin-aware verification. *Preprint*, arXiv:2601.15498.

Minghao Yan, Saurabh Agarwal, and Shivaram Venkataraman. 2025. Decoding speculative decoding. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6460–6473, Albuquerque, New Mexico. Association for Computational Linguistics.

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, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Situo Zhang, Hankun Wang, Da Ma, Zichen Zhu, Lu Chen, Kunyao Lan, and Kai Yu. 2024. Adaeagle: Optimizing speculative decoding via explicit modeling of adaptive draft structures. *Preprint*, arXiv:2412.18910.

A Experimental Details

All end-to-end evaluations use the same generation loop, prompt tokenization, target model, decoding budget, and hardware class for a given model-dataset setting.

Models. Table 5 lists the Hugging Face[2] checkpoints used for target models and EAGLE3 drafters.

Model	Target model checkpoint	EAGLE3 checkpoint
Qwen3-8B	Qwen/Qwen3-8B	Tengyunw/qwen3_8b_eagle3
Qwen3-4B	Qwen/Qwen3-4B	AngelSlim/Qwen3-4B_eagle3
Llama3.1-8B	meta-llama/Llama-3.1-8B-Instruct	yuhuili/EAGLE3-LLaMA3.1-Instruct-8B

Table 5: Target model and EAGLE3 checkpoints.

Generation Configuration. Table 6 lists the shared generation configuration.

Setting	Value
Batch size	1
Maximum new tokens	8192
Maximum sequence length	16384
Temperature	0.0
Data type	float16
Split seed	42

Table 6: Shared generation configuration.

[2] https://huggingface.co/

Drafting Configuration. Table 7 lists the EAGLE3 and suffix-cache configuration.

Component	Setting	Value
EAGLE3	Total draft tokens	32
EAGLE3	Tree depth	8
EAGLE3	Top-k	4
Suffix cache	Maximum tree depth	64
Suffix cache	Maximum draft tokens	64
Suffix cache	Maximum speculative factor	1.0
Suffix cache	Maximum speculative offset	0.0
Suffix cache	Minimum token probability	0.1

Table 7: Drafting configuration.

Hybrid Configuration. Frequency-Based Hybrid Decoding uses the default suffix-cache score of the SuffixDecoding baseline described in Section 4.2. Hybrid Verified Decoding uses a lightweight payoff predictor over runtime features available before verification. The cache draft is selected when the predicted accepted length is at least 6. Predictor features, architecture, and training details are given in Appendix B.

Evaluation Metrics. Main end-to-end results are measured on H100 GPUs. Including trace preparation, payoff-predictor training, and the reported timing, diagnostic, and hardware runs, the experiments used approximately 50 GPU-hours. For each run, we record generated token count N, wall-clock decoding time T, and decoding-cycle count C. Throughput is computed as

$$\text{Throughput} = \frac{N}{T}.$$

For a method m and baseline b evaluated on the same model, dataset, and prompt set, speedup is computed as

$$\text{Speedup}(m, b) = \frac{\text{Throughput}_m}{\text{Throughput}_b}.$$

For execution analysis, we also report decoding cycles per 1K generated tokens,

$$\text{CyclesPer1K} = 1000 \cdot \frac{C}{N},$$

and average time per decoding cycle,

$$\text{TimePerCycle} = 1000 \cdot \frac{T}{C},$$

where TimePerCycle is measured in milliseconds. The aggregate metrics reported in the paper are computed from the recorded run outputs.

Workload family	Dataset	Source
Agentic / workflow editing	Delegate-52	`microsoft/delegate52`
	InstructEdit/FineEdit	`YimingZeng/FineEdit_bench`
Code / repository	RepoBench	`tianyang/repobench_python_v1.1`
	SWE-bench OpenHands	`SWE-bench/SWE-bench_Verified`
	Magicoder	`ise-uiuc/Magicoder-Evol-Instruct-110K`
	MBPP	`google-research-datasets/mbpp`
Structured generation	BFCL	`llamastack/bfcl_v3`
	Spider	`NBAmine/xlangai-spider-with-context`
Multi-hop QA	HotpotQA	`hotpotqa/hotpot_qa`
	MuSiQue	`dgslibisey/MuSiQue`
	2WikiMultiHopQA	`Alab-NII/2wikimultihop`
Long-context	InfiniteBench	`xinrongzhang2022/InfiniteBench`
	CNN/DailyMail	`abisee/cnn_dailymail`
	GovReport	`ccdv/govreport-summarization`
Open-ended instruction / chat	Alpaca	`tatsu-lab/alpaca`
	MT-Bench	Official EAGLE GitHub repository (MT-Bench prompts)

Table 8: Dataset sources used to construct the evaluation prompt package.

B Payoff Label Construction and Predictor Training

Payoff Label Construction. We construct training examples by replaying cache lookup on recorded target-model generations. For each trace, we treat each token position t as a decoding state with prefix $x_{1:t}$. The suffix cache is queried with this prefix and returns a cache draft d_t^c. The label is the number of draft tokens that match the following tokens in the same trace:

$$y_t = \max\left\{\ell \in \{0, \ldots, |d_t^c|\} : d_{t,1:\ell}^c = x_{t+1:t+\ell}\right\}.$$

The recorded continuation therefore provides the accepted-length label for the cache draft, without additional target-model inference for each replayed draft.

Predictor Features. The predictor uses only signals available before verification. These features describe the cache match that produced the draft, the current decoding state, recent cache-draft behavior, and simple token patterns in the proposed draft. Table 9 groups these features into cache-match, decode-state, recent-history, and draft-structure signals. The feature set is designed to capture whether the current draft is likely to remain valid in the current decoding state, without using the target model's verification result for that draft.

Feature group	Description
Cache match	Suffix score, matched length, draft length
Decode state	Prompt length, generated length, relative decode position
Recent history	Recent cache-draft selection rate and accepted lengths over recent decoding steps
Draft structure	Indicators for whitespace, punctuation, line breaks, brackets, and delimiter-like tokens

Table 9: Payoff predictor feature groups.

Predictor Training. The predictor maps the pre-verification feature vector ϕ_t to a predicted accepted length:

$$\hat{y}_t = g_\theta(\phi_t).$$

We train it with squared-error regression:

$$\min_\theta \sum_t \left(g_\theta(\phi_t) - y_t\right)^2.$$

The predictor is a small multilayer perceptron. Table 10 lists the training configuration.

Runtime Use. At each decoding step, the same pre-verification features are computed for the current cache draft. Hybrid Verified Decoding selects the cache draft when the predicted accepted length is at least 6. Otherwise, it uses the model-based drafter. The target model and verification rule are unchanged.

Setting	Value
Hidden layers	2
Hidden size	256
Activation	ReLU
Learning rate	0.001
Batch size	65536
Epochs	20
Seed	42

Table 10: Payoff predictor training configuration.

C Additional End-to-End Results

Tables 11–13 report aggregate measurements over the sixteen datasets.

Method	Throughput tokens/s	Cycles /1K	Accepted tokens	Cache %	EAGLE3 %
Greedy	39.9	998.7	–	–	–
EAGLE3	109.5	240.2	3.16	0.0	100.0
Suffix	42.1	397.8	1.51	97.6	0.0
Frequency Hybrid	105.0	188.4	4.31	8.0	92.0
Hybrid Verified	133.0	189.7	4.27	9.7	90.3

Table 11: Aggregate results for Qwen3-8B. Columns report throughput, cycles per 1K output tokens, mean accepted length, and draft-source fractions. Dashes mark fields outside the greedy path.

Method	Throughput tokens/s	Cycles /1K	Accepted tokens	Cache %	EAGLE3 %
Greedy	38.3	1000.0	–	–	–
EAGLE3	76.7	309.7	2.23	0.0	100.0
Suffix	54.6	346.1	1.89	97.6	0.0
Frequency Hybrid	104.5	223.9	3.47	8.2	91.8
Hybrid Verified	101.1	228.6	3.37	8.8	91.2

Table 12: Aggregate results for Qwen3-4B, using the column definitions in Table 11.

Method	Throughput tokens/s	Cycles /1K	Accepted tokens	Cache %	EAGLE3 %
Greedy	40.7	1000.0	–	–	–
EAGLE3	47.7	571.0	0.75	0.0	100.0
Suffix	78.2	264.0	2.79	93.9	0.0
Frequency Hybrid	98.3	234.9	3.26	9.4	90.6
Hybrid Verified	126.2	200.1	4.00	18.6	81.4

Table 13: Aggregate results for Llama3.1-8B, using the column definitions in Table 11.

D Dataset and Prompt Details

We evaluate on sixteen datasets grouped into six workload families. For every dataset, we fix 20 prompts for reported timing and analysis, giving 320 measured prompts per target model. Separate cache-warmup prompts from the same dataset are used to initialize the suffix cache and construct replay-based payoff labels. The test prompts are never used for cache warmup or predictor training.

For Hugging Face datasets, Table 8 reports the dataset identifier. For other sources, it reports the project or benchmark source used to construct the prompt package.

E vLLM Serving Prototype

We implement Hybrid Verified Decoding in the vLLM speculative decoding path. For each request, the serving path runs the selected draft source, verifies the proposed tokens with the target model, and updates the decoding state used by the payoff predictor. The predictor scores cache drafts before verification and selects cache verification when the predicted accepted length reaches the threshold. Otherwise, the request uses the model-based drafter.

The serving case study uses Qwen3-8B with its EAGLE3 checkpoint and the same payoff predictor used in the offline experiments. We run batch size 1 serving on H100 and evaluate MT-Bench and CNN/DailyMail under deterministic decoding. Table 14 summarizes the serving configuration.

Setting	Value
Serving engine	vLLM
Target model	Qwen3-8B
Model-based drafter	EAGLE3
Cache drafter	Suffix cache
Predictor decision	Predicted accepted length ≥ 6
Batch size	1
Hardware	H100
Datasets	MT-Bench, CNN/DailyMail

Table 14: Configuration for the vLLM serving case study.

F Artifact Licenses and Intended Use

We use publicly available model checkpoints, datasets, benchmarks, and code artifacts for research evaluation of decoding efficiency under their respective licenses and access terms. The Qwen checkpoints are released under Apache-2.0, and Llama3.1-8B-Instruct uses the Llama 3.1 Community License. The EAGLE repository and vLLM are released under Apache-2.0. Dataset licenses and source terms vary across benchmarks, including Apache-2.0, MIT, CC-BY, CC-BY-SA, CC-BY-NC, and CDLA-Permissive terms. Our code will be released under the MIT License.

G Analysis Metric Definitions

In Section 5.3 and Table 2, a proposed cache draft is an oracle high-payoff draft when its replayed accepted length is at least 6. A predictor-selected draft has predicted accepted length at least 6. Precision is the fraction of predictor-selected drafts that are oracle high-payoff drafts, and recall is the fraction of oracle high-payoff drafts selected by the predictor. Cache selection rate is the fraction of decoding cycles in which Hybrid Verified Decoding submits a cache draft for verification. Mean accepted length is averaged over verified draft cycles for the corresponding method or draft source.

H AI Assistant Use

AI assistants were used to support coding and text editing. Their use followed the ACL policy on publication ethics. The authors reviewed the generated suggestions and are responsible for the submitted work.