SAME CONVERSATION, DIFFERENT TOKENS: CHARACTERIZING AND MITIGATING TOKEN DRIFT FOR EFFICIENT MULTI-TURN LLM SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-turn LLM serving reuses cached KV states only when a new prompt extends the token sequence the engine has already processed. Applications, however, carry conversation history as text: rendering it through a chat template and encoding the result again can return the same conversation as a different token sequence, silently defeating prefix reuse. We call this divergence token drift and characterize it across six open models, three conversation datasets, and four languages. Re-encoding alone breaks the prefix in 4.6%–90.8% of requests at temperature 1.0, at a roughly constant rate per generated token, while template history rewriting breaks it on every request after the first in three of the six model configurations. Since reuse ends at the first mismatch, rebuilding history costs $4\times$–$23\times$ the prefill the new content alone would need, and the KV branches stranded by misses add 10%–122% to the live context's footprint while remaining indistinguishable from it under recency-based eviction. We design one technique per layer of the serving pipeline and integrate all three into vLLM: canonical generation at the sampler with CG-Recheck, a checker we design that constrains sampling to token sequences that survive re-encoding; token-level history (TLH) at the request interface, which carries earlier turns forward as the token IDs the engine produced; and DriftEvict, a KV-cache eviction policy that identifies a dead branch when a turn completes and reclaims it ahead of live context. The two preventive techniques raise cache hit rates where their layer can prevent the miss, and TLH improves time to first token (TTFT) the most; but both change what the model samples or sees and may lower output quality. DriftEvict leaves outputs unchanged and improves throughput and TTFT under high load on every evaluated model.

1 INTRODUCTION

Large language models (LLMs) increasingly serve interactive applications in which a conversation evolves over many turns, so consecutive requests share most of their context. Prefix caching (Kwon et al., 2023; Zheng et al., 2024) exploits this overlap by reusing the KV states of shared prefixes across requests; it is now standard in production (Wang et al., 2025; OpenAI, 2024; Anthropic, 2026). Shared history, however, does not guarantee a shared token prefix: applications carry history forward as messages, and prompt construction rebuilds the token sequence from them for every request.

We call the resulting divergence *token drift*: consecutive requests share the history but not its token sequence. Prefix reuse ends at the first mismatch. Drift has two sources. *Re-encoding* drift arises because a generated message is carried forward as text: many token sequences decode to the same string, so encoding that text again need not recover the tokens the model produced. *History rewriting* arises when prompt construction presents an earlier message differently from how it was generated, as when a chat template (Carrigan, 2023) trims whitespace or omits a reasoning block that the transcript contains; the prefix then breaks even where re-encoding would have recovered the tokens.

Our characterization yields three findings. First, drift is common. Re-encoding drift occurs across models and languages, least often in English; it persists under greedy decoding; and a constant-hazard approximation captures how it grows with output length. In three of the six configurations, the chat template rewrites every completed turn, so history rewriting breaks the prefix on every request after

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.

the first, whatever the content and the temperature. Second, the cost of drift is out of proportion to its size: re-encoding typically changes only a few tokens locally, but reuse ends at the first mismatch, so rebuilding history from text costs many times the prefill the new content alone would need. Third, drift costs memory as well as compute: the old KV states beyond the first mismatch form a *dead branch* that the conversation no longer reuses. Recency-based eviction cannot distinguish this branch from live context, so live context may be evicted first.

We address drift at three layers of the serving pipeline. At the sampler and the request interface, a drift-induced cache miss can be prevented, at the cost of changing what the model samples or sees. Canonical generation (CG) (Vieira et al., 2025; Chatzi et al., 2026) constrains sampling so that generated text re-encodes to the tokens the model produced; it targets re-encoding drift but cannot prevent history rewriting. Token-level history (TLH) keeps earlier turns in token-ID form, preventing drift from both sources. The KV cache cannot prevent the miss, but it can reclaim what the miss leaves behind. DriftEvict identifies the dead branch when a turn completes and evicts it first. It works whichever source caused the miss, and it changes neither what the model samples nor what it sees.

This paper makes three contributions.

Characterization. We measure token drift and its cost across six models, three datasets, and four languages (Section 2): re-encoding mismatches in 4.6%–90.8% of requests at temperature 1.0, 4×–23× prefill amplification, and 10%–122% excess KV footprint from dead branches.

Techniques. We design one technique per layer and integrate all three into vLLM (Section 3). At the sampler, we bring canonical generation into serving and design its checker, CG-Recheck, which rechecks locally rejected candidates in the full generated prefix. At the request interface, we design a content-addressed TLH store that finds a conversation's transcript by its messages, without session identifiers. In the KV cache, we propose DriftEvict. When a turn completes, it identifies the dead branch and evicts it first.

Evaluation. We compare the three techniques on multiple models under load and judge output quality (Section 4). TLH improves time to first token (TTFT) the most, but bypassing the template's history rewriting can lower output quality. CG-Recheck prevents nearly all re-encoding misses but cannot prevent history rewriting; its checks add work at every decoding step, and constraining the sampling may lower output quality. DriftEvict leaves outputs unchanged and improves throughput and TTFT under high load on every evaluated model.

2 Characterizing Token Drift

2.1 Definition and Sources

At turn t, the serving pipeline constructs the prompt token sequence $\mathbf{P}_t$ from the conversation messages, and the model generates $\mathbf{G}_t$. The output is detokenized, parsed into an assistant message, and appended to the history, from which the next request constructs $\mathbf{P}_{t+1}$. The prefix cache can reuse KV states along the longest common prefix of $\mathbf{P}_{t+1}$ and the previous transcript $\mathbf{T}_t = \mathbf{P}_t \| \mathbf{G}_t$. When the application carries earlier messages forward unchanged but $\mathbf{P}_{t+1}$ does not extend $\mathbf{T}_t$, we say that request $t+1$ *drifts*.

The first source of drift is *re-encoding*: different token sequences can decode to the same text, but the tokenizer selects a single segmentation for that text (Cao & Rimell, 2021; Geh et al., 2024). Thus, the model's generated tokens need not satisfy $\mathrm{Encode}(\mathrm{Decode}(\mathbf{G}_t)) = \mathbf{G}_t$. For example, with Qwen3.8-27B's tokenizer (Qwen Team, 2026d), two generated tokens `[Hel][lo]` decode to `Hello`, which re-encodes as one token `[Hello]`. The second is *history rewriting*: prompt construction changes the content or formatting of earlier messages. For the same model, the token `[ Hello]` survives re-encoding (with a leading space). However, the chat template strips the leading space when rendering history, changing the token to `[Hello]`.

2.2 Drift after Re-encoding

We use conversations from ShareGPT (anon8231489123, 2023), WildChat (Zhao et al., 2024), and ReviewMT (Tan et al., 2024), with assistant responses generated by six models in four languages (Appendix A.1). In our experiments, Qwen3.8-27B (medium effort) and GLM-4.7-Flash run with

reasoning enabled; the remaining models run without it. For each request, we decode and re-encode its recorded transcript $\mathbf{T}_t$ and locate the first mismatch, if any.

At temperature 1.0, re-encoding mismatch rates range from 4.6% to 90.8% across models and languages, English lowest for every model (Figure 1). Greedy decoding does not remove them: the model's most probable token at a position need not be the tokenizer's choice for the resulting text.

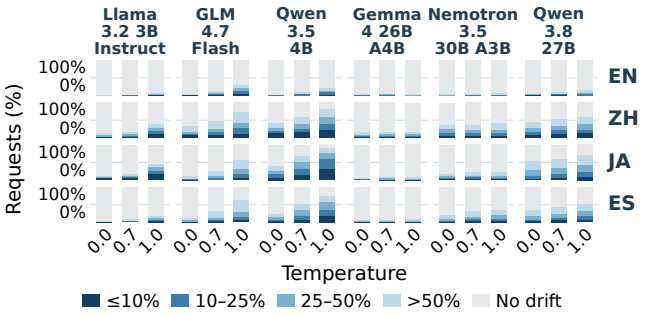

Figure 1: Re-encoding drift by model, response language, and sampling temperature, pooled across datasets. Segments give the share of requests by first-mismatch position within the output; bars include the first request of each conversation.

A constant-hazard model with per-token rate p approximates the mismatch probability for an output of length L as $1 - (1 - p)^L$. Under this model, the mean position of the first mismatch is $1/p$ (Appendix A.3). The rate depends on the model, language, and sampling temperature, and the product pL governs the outcome. For example, Qwen3.8-27B and Gemma-4-26B-A4B have similar English hazards at temperature 1.0. Yet Qwen3.8-27B's outputs are $3.8\times$ as long on average and drift $3.3\times$ as often.

2.3 History Rewriting

To separate the two sources, we measure two miss rates over the requests after the first in each conversation. The *actual miss rate* is the fraction of these requests whose prompt $\mathbf{P}_{t+1}$ does not extend $\mathbf{T}_t$; it includes both sources. A *history-preserving oracle* gives the miss rate of re-encoding alone: it appends the new messages to the unchanged $\mathbf{T}_t$, then decodes and re-encodes the result. The gap between the two rates counts the misses that history rewriting adds.

Table 1: Miss rate of requests after the first in each conversation at temperature 1.0, actual and from re-encoding alone, with the dominant source.

Model	Reasoning config	Actual	Re-encoding only	Dominant source
Llama 3.2	n/a	29%	29%	re-encoding
Qwen3.8	medium	51%	50%	re-encoding
Nemotron 3.5	off	27%	27%	re-encoding
Qwen3.5	off	100%	68%	both
Gemma 4	off	100%	10%	rewriting
GLM 4.7	on	100%	56%	both

For Llama-3.2-3B-Instruct, Qwen3.8-27B, and Nemotron-3.5-Lightning-30B-A3B, re-encoding accounts for most misses: history rewriting adds less than one percentage point (Table 1). For the other three, every request after the first misses. For Gemma-4-26B-A4B, re-encoding alone would cause only a 10% miss rate, so history rewriting accounts for most misses. For Qwen3.5-4B and GLM-4.7-Flash, re-encoding alone would cause miss rates of 68% and 56%, but history rewriting breaks the prefix first.

With reasoning disabled, the generation prompts of Qwen3.5-4B and Gemma-4-26B-A4B end with an empty reasoning block. The template omits this block when it renders the completed turn as history. Every request after the first therefore misses before the generated output, whatever the content and the temperature. By contrast, Nemotron-3.5-Lightning-30B-A3B normally reproduces the block in history. With reasoning enabled, the GLM-4.7-Flash template by default omits earlier reasoning when a new user turn begins. Every GLM request after the first therefore misses before the generated output (Appendix A.2). By contrast, Qwen3.8-27B retains earlier reasoning in history by default.

2.4 Prefill Amplification

Both sources of drift end prefix reuse at the first mismatch, so even a small token change can force a long suffix to be re-prefilled. At temperature 1.0, re-encoding affects a median of 0.073%–0.723% of tokens in drifted outputs across model–language groups (Appendix A.4). Yet in 65.9% of requests with re-encoding drift, the first mismatch falls within the first half of the output (Figure 1).

A high overall cache hit rate does not imply that drift is rare: a drifted request re-prefills only from its first mismatch, which typically falls in the latest turn. Earlier history was re-prefilled on previous turns and stays cached, so most of the prompt can still hit. The re-prefilled suffix, however, is long relative to the new content.

We measure prefill cost assuming an unbounded exact-prefix cache. For each request i after the first in its conversation, let P_i^{pres} be the number of new prompt tokens and P_i^{recon} the number of prefill tokens when history is rebuilt from text. The *prefill amplification* of a workload is

$$A = \frac{\sum_i P_i^{\text{recon}}}{\sum_i P_i^{\text{pres}}} = 1 + \frac{m\,D}{N},$$

where m is the actual miss rate, D the mean number of historical tokens a missed request re-prefills, and N the mean number of tokens the new content requires. Where re-encoding sets the first mismatch, the reusable part of an output averages at most $1/p$ tokens, however long the output. The re-prefilled length D therefore keeps growing with output length even after the miss rate m saturates (closed form in Appendix A.3).

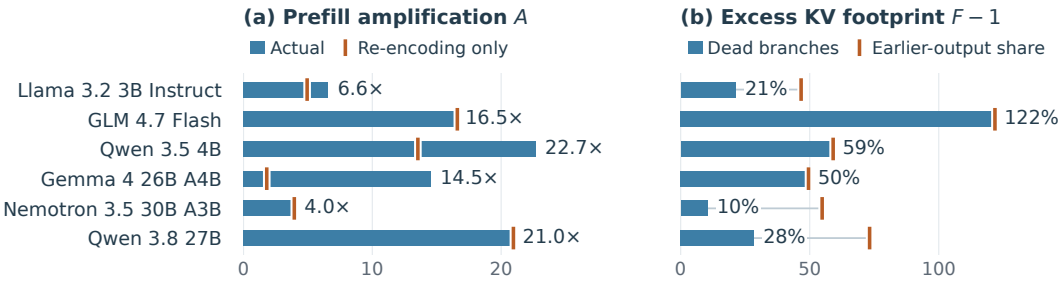

Figure 2: Amplification at temperature 1.0, pooled over datasets and languages. Ticks in (b) show the total size of earlier assistant outputs, including reasoning omitted from history. Both quantities are relative to the final live context.

Actual prefill amplification ranges from $4\times$ to $23\times$ (Figure 2a). Re-encoding alone would still incur $1.8\times$–$21.0\times$ prefill amplification. N is 55 to 72 tokens for every configuration, a user turn plus template tokens. D is 800 to 2,470 tokens. On average, a miss adds ten to forty user turns' worth of prefill.

For Qwen3.5-4B and Gemma-4-26B-A4B, where the first mismatch falls before the generated output, D is almost the entire previous output. For Qwen3.8-27B and Nemotron-3.5-Lightning-30B-A3B, where re-encoding sets the first mismatch, D is 62% and 56% of it. For Llama-3.2-3B-Instruct, ten date-header rewrites (0.5% of requests) break prefix reuse within the date header and explain most of the gap between the actual $6.6\times$ and the $4.9\times$ from re-encoding alone. For GLM-4.7-Flash, the previous output's KV states cannot be reused, but only the answer is re-prefilled because reasoning is omitted from history. On average, a miss invalidates KV states for 2,320 tokens of reasoning and answer while re-prefilling only 850 answer tokens.

2.5 KV Footprint Amplification

Per conversation, prefix caching retains KV states for every distinct prefix its requests present, forming a token-prefix trie (Zheng et al., 2024). Without drift, the trie is a single path representing the final live context $\mathbf{T}_n$. With drift, later requests follow the reconstructed history along a different branch, leaving the old suffix cached but unused; we call such suffixes *dead branches*. The *KV footprint amplification* is $F = \text{trie size}/|\mathbf{T}_n|$, so that $F - 1$ is the excess footprint relative to the live context.

Dead branches add 10%–122% to the footprint of the live context (Figure 2b). We compare $F - 1$ with the total length of earlier generated outputs relative to the final live context. For Qwen3.5-4B and Gemma-4-26B-A4B, where every earlier output is invalidated in full, $F - 1$ matches this share within one point. For Qwen3.8-27B and Nemotron-3.5-Lightning-30B-A3B, where re-encoding strands only what follows the first mismatch on the turns it affects, $F - 1$ falls well below it. GLM-4.7-Flash also strands every earlier output in full, including reasoning omitted from the live context, so dead branches add 122% to its footprint. Separate recordings with earlier reasoning retained have similar

prefill amplification ($16.7\times$ versus $16.5\times$), but dead branches add only 26% to the live context. Prefill amplification alone therefore does not capture the excess KV footprint.

Recency-based eviction, as in vLLM's LRU free-block queue (vLLM Team, 2026), cannot distinguish dead branches from live context. The blocks of a dead branch were last touched when the turn completed, so recency may keep them ahead of the live contexts of conversations that have idled longer. How much of this excess KV stays resident depends on cache pressure and idle times; Section 4 measures the effect under load.

3 Mitigating Token Drift

We design three techniques, each acting at a different layer of the serving pipeline, and integrate them into vLLM (Kwon et al., 2023). CG-Recheck at the sampler and TLH at the request interface prevent the miss; DriftEvict in the KV cache reclaims the branch a miss strands.

3.1 CG-Recheck: Full-Prefix Rechecks at the Sampler

Canonical generation constrains sampling to token sequences that survive the decode–encode round trip (Vieira et al., 2025; Chatzi et al., 2026). We build on the sampler of Chatzi et al. (2026): at each step it tries up to 50 candidates in sampling order and emits the first that passes a canonicality check; if none passes, it emits EOS, which we count as a sampling failure.

The released sampler applies a full-prefix test, which round-trips the entire generated output for every candidate, at a cost linear in the output length. The bigram test (Chatzi et al., 2026, Section 4.2) round-trips only the last token and the candidate, which suffices for pure BPE (Vieira et al., 2025, Appendix B). A production tokenizer, however, pre-tokenizes text first, so a candidate's segmentation can depend on earlier context. After `\|`, for example, the candidate `x` is canonical, yet `|x` alone merges into one token and the bigram test rejects it (Appendix A.5). Such false rejections can exhaust the candidates and cause sampling failures.

We design CG-Recheck to keep the speed of the bigram test without its false rejections: it uses the bigram test as the fast path and rechecks only rejected candidates with the full-prefix test before the sampler moves on. On Qwen3.5-4B, CG-Recheck fails on 0.085% of requests, as rarely as the full-prefix test (0.128%) and far less often than the bigram test (9.98%). It runs at 97.4% of the bigram test's throughput, where the full-prefix test reaches 29%.

Because CG-Recheck leaves prompt construction unchanged, history rewriting still causes misses. Its checks add work at every decoding step. Constraining the sampling also changes the output distribution and can alter even the greedy choice; Section 4.3 evaluates output length and quality under free generation.

3.2 TLH: Content-Addressed History at the Request Interface

Token-level history preserves earlier turns as token IDs and renders and encodes only new messages. Multi-turn RL frameworks use the same discipline, token-in/token-out (TITO), to keep training on-policy despite retokenization (Li et al., 2026; Gallouédec & Rasul, 2026; Zhang et al., 2026). We design a content-addressed TLH store for serving, which differs from a rollout in three ways. A chat-completion request identifies its conversation only by its messages, so the store finds a conversation's recorded transcript by matching its message prefix. One server holds conversations from many clients, so all conversations share one byte budget for retention. Template parameters vary across requests, so the store keeps a separate namespace for each tokenizer and template configuration.

Messages do not identify one token history, since the same messages can arise from different generated token sequences. The store, however, serves the prefix-cache hit rate rather than fidelity to one rollout: among matching checkpoints, it prefers the longest message prefix and then the most recently accessed, whose KV states a recency-based cache is likeliest to still hold.

The store sits in the serving front end above an unmodified engine. At turn completion, it records a checkpoint associating the conversation's messages with the prompt and generated token IDs; checkpoints in one namespace share a radix tree over normalized message keys, and leaf checkpoints are evicted by recency. For an incoming request, the store restores the matched transcript and

appends the remaining messages. To tokenize them as the template would after the recorded turn, it renders them behind a placeholder assistant message and encodes only the text after the placeholder (Appendix A.6.1). On a hit, the prompt extends the recorded transcript exactly, so drift from either source is prevented for the covered turns and their KV states are reused where still cached; on a miss, the prompt is the baseline's.

A hit can also change what the model sees relative to reconstructing history from text. If the template preserves the transcript's text and formatting, only token segmentation can differ. If the template rewrites history, the restored transcript bypasses those rewrites for the covered turns; in Qwen3.5-4B, it retains the empty reasoning block that the template omits. Section 4.3 evaluates how these changes affect output length and quality.

3.3 DRIFTEVICT: DRIFT-AWARE EVICTION IN THE KV CACHE

DriftEvict reclaims dead branches while leaving prompt construction and sampling unchanged. Recency-based eviction ranks blocks by last use. A dead branch was last used when its turn completed, so it can outlast the live contexts of conversations that have idled longer. Which blocks are dead, however, is already determined when the turn completes. Both sources of drift lie in prompt construction, which needs only the finished reply: the template renders it as history, and the tokenizer re-encodes its text. Every later request will present the reply through this same construction, so prefix caching will reuse the cached transcript only up to the first mismatch with its result. DriftEvict performs the construction at completion, locates the first mismatch, and evicts the blocks beyond it ahead of live context. Waiting for the next request to reveal the mismatch would let the dead branch compete for capacity until that request arrives.

The dead part of an output varies from nothing on a turn without drift to nearly all of it under history rewriting. The prefix match locates it exactly on every turn. The rendering DriftEvict matches is a *probe*: the request's messages, the finished reply as history, and a placeholder user message, rendered under the request's template and rendering parameters. The placeholder makes the template treat the reply as it will in the next request: Qwen3.5-4B and GLM-4.7-Flash, for instance, keep the reasoning block of a final assistant message but omit it once a user turn follows. For the user–assistant protocol, templates, and fixed rendering settings studied here, the rendering of the known history does not depend on the next user's text, so the probe equals the next prompt up to the turn boundary (Appendix A.6.2). Dead blocks move to the front of the eviction order, ahead of every live block, while the recency order among live blocks is unchanged; DriftEvict therefore has little effect until the cache is under pressure.

DriftEvict does not remove the prefill caused by drift: the next request still recomputes history beyond the reusable prefix. Instead, it reduces the additional prefill that arises when cache pressure evicts reusable contexts. The check runs asynchronously, off the request path, and performs no model prefill. It costs one template rendering and one tokenizer encoding on the CPU. The serving measurements that follow include this cost.

4 EVALUATION

4.1 SETUP

We evaluate the techniques in two settings: fixed-output replay to compare serving performance, and free generation to measure their effects on output quality and length. We sample equal numbers of multi-turn conversations from ShareGPT, WildChat, and ReviewMT for both evaluations (Appendices A.1 and A.9).

We evaluate serving performance on three models: Qwen3.5-4B, Qwen3.8-27B, and GLM-4.7-Flash. Qwen3.5-4B runs with reasoning disabled; Qwen3.8-27B generates reasoning and retains it in history; GLM-4.7-Flash generates reasoning, which is omitted from the prompt for the next user turn. For GLM-4.7-Flash, we restrict the comparison to the baseline and DriftEvict: CG-Recheck cannot prevent the rewrite-induced prefix break, and Qwen3.5-4B already covers that case, while TLH would bring back the omitted reasoning. We use vLLM with prefix caching as our baseline and build all three techniques on top of it. Qwen3.8-27B runs on two A800 GPUs under tensor parallelism, and the other two models on one H20 GPU each, with up to 128 concurrent sequences.

We use fixed-output replay to compare serving performance while controlling answer content and output length. At each decoding step, the engine emits the recorded output token. The baseline, TLH, and DriftEvict replay the original token IDs. CG-Recheck replays the same answers using their decode–encode-normalized IDs, with its checks still running. Normalization changes the total output-token count by less than 0.1% in every evaluated condition (Appendix A.7). Sessions arrive as a Poisson process, and we vary its arrival rate to set the load. Within a session, each request follows the previous reply after an idle time. Idle times have a median of 74 seconds and are drawn from a lognormal distribution fitted to ThoughtTrace (Jin et al., 2026). We report mean time to first token (TTFT), decoding throughput, and the KV cache hit rate, the share of prompt tokens served from the cache.

4.2 Serving under Load

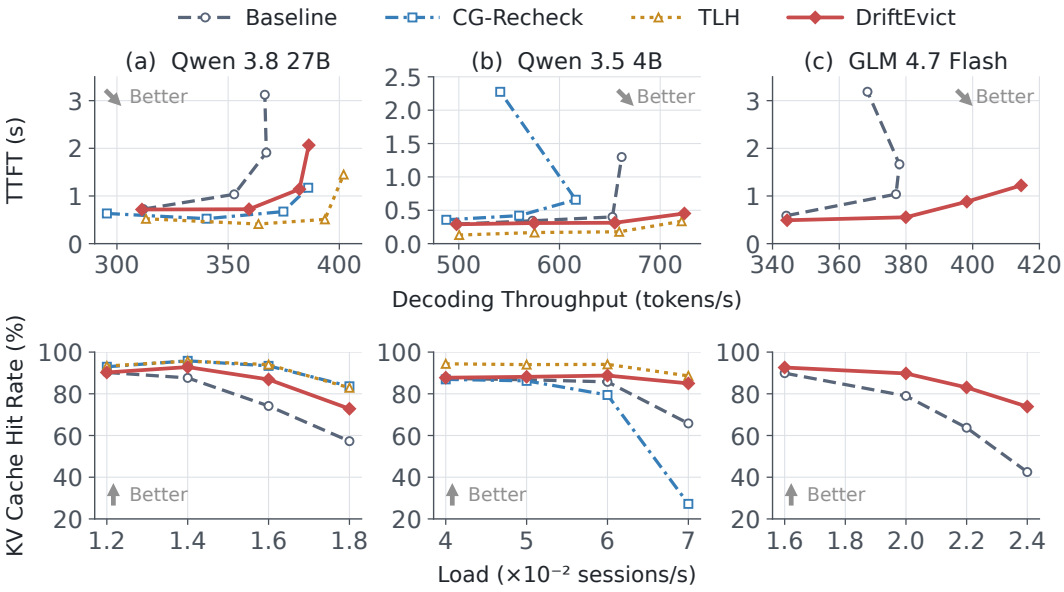

Figure 3: Serving under load at four arrival rates per model. Top: TTFT against decoding throughput, the last point of each curve at the highest rate. Bottom: KV cache hit rate.

The top row of Figure 3 plots TTFT against decoding throughput as the arrival rate rises. At higher arrival rates, the baseline's throughput begins to plateau while TTFT worsens sharply. We compare how each technique changes this throughput–latency trade-off over the tested arrival rates.

TLH achieves the best TTFT at almost every tested arrival rate on both Qwen models. At the highest tested rate, it reaches 402 tokens/s against the baseline's 366 on Qwen3.8-27B and 721 against 662 on Qwen3.5-4B. On Qwen3.5-4B, where every request after the first misses under reconstruction and a hit prevents the miss whatever its source, its TTFT stays between 0.13 and 0.34 s while the baseline's rises from 0.29 to 1.30.

CG-Recheck prevents only re-encoding misses, and the two Qwen models show both sides of that. On Qwen3.8-27B, where re-encoding is the source, its TTFT is better than the baseline's at every tested arrival rate. At the highest tested arrival rate, its throughput exceeds the baseline's but remains below TLH's, since checking adds work at every decoding step. On Qwen3.5-4B, the template's rewrite still breaks the prefix, so canonicality checks do not remove the rewrite-induced prefill. Its TTFT is worse than the baseline's at every tested arrival rate, and throughput peaks at 616 tokens/s and falls back at the highest tested arrival rate, while the baseline reaches 662.

DriftEvict does not prevent drift-induced misses; it reduces the additional misses caused by evicting reusable context under cache pressure. Its TTFT benefit is modest at the lowest load and grows with the arrival rate, as its TTFT degrades more slowly than the baseline's. At the highest tested arrival rate for each model, DriftEvict improves both TTFT and throughput over the baseline: TTFT falls from

3.1 to 2.1 s on Qwen3.8-27B, from 1.3 to 0.5 on Qwen3.5-4B, and from 3.2 to 1.2 on GLM-4.7-Flash, while throughput rises from 366 to 386, 662 to 724, and 368 to 415 tokens/s.

We also compare DriftEvict with a naive policy that prioritizes all generated KV states for eviction after turn completion. At the highest evaluated load on Qwen3.8-27B, DriftEvict achieves a $2.55\times$ speedup in mean TTFT and increases decoding throughput by 12.2% relative to this policy. The naive policy loses because it also evicts live context. On Qwen3.8-27B, about half of the requests after the first do not drift, so their whole outputs stay live. Where re-encoding does cause a miss, 38% of the output still precedes the first mismatch.

The bottom row separates the two mechanisms. Preventing drift-induced misses raises the hit rate at every load, including the lowest, where the cache is not yet full: TLH adds 3.1 and 7.0 points there on the two Qwen models, CG-Recheck 2.7 points on Qwen3.8-27B and none on Qwen3.5-4B. Reclaiming footprint has little effect at the lowest load, where DriftEvict and the baseline nearly coincide on both Qwen models. As load rises, the pool fills and recency evicts the live contexts of conversations waiting for their next turn, so the baseline's hit rate falls to 42%–66% across the three models; DriftEvict evicts the dead branches ahead of them and holds 73%–85% at the same loads, recovering most on GLM-4.7-Flash, whose dead branches are largest. Slower decoding has the opposite effect by keeping requests resident longer: CG-Recheck on Qwen3.5-4B falls to a 27% hit rate at the highest tested arrival rate.

Temperature and language. At a moderately high load for each model, we also replay conversations recorded under greedy decoding, and English and Chinese conversations at temperature 1.0, measuring each technique's TTFT speedup against the baseline of the same condition (Appendix A.8). On Qwen3.8-27B, CG-Recheck and TLH gain more at temperature 1.0 than under greedy decoding and more in Chinese than in English, consistent with the re-encoding drift in Section 2.2; in English, CG-Recheck worsens TTFT by 2.6%, as checking costs outweigh prefill savings. On Qwen3.5-4B, where every request after the first misses under reconstruction, TLH speeds up TTFT by $2.2\times$–$2.5\times$ and DriftEvict by $1.2\times$–$1.4\times$ across conditions, while CG-Recheck stays slower at $0.58\times$–$0.85\times$. GLM-4.7-Flash also misses on every request after the first, yet its baseline TTFT rises from 1.0 s under greedy decoding to 1.7 s at temperature 1.0, and DriftEvict gains $1.4\times$–$1.9\times$; miss rate alone does not determine the gain, since prompt and output lengths change prefill work and cache pressure.

4.3 Output Quality

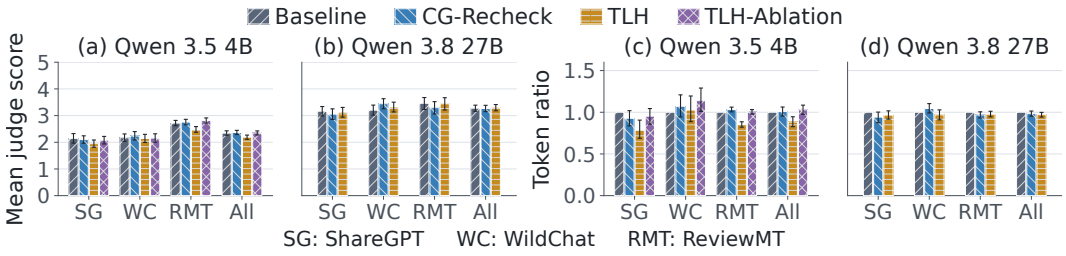

Figure 4: Output quality under free generation at temperature 1.0, by dataset and pooled (All). (a, b): mean judge score, 1 to 5. (c, d): geometric mean of each conversation's output token count relative to its paired baseline. Error bars: 95% bootstrap confidence intervals. TLH-Ablation removes the empty reasoning scaffold from history while retaining the generated token segmentation.

We compare the baseline, CG-Recheck, and TLH under free generation at temperature 1.0 on Qwen3.5-4B and Qwen3.8-27B, with 300 paired conversations per model, 100 per dataset; each method follows the same prescribed user turns and carries its own previous answers forward. Following the LLM-as-a-judge approach (Zheng et al., 2023), we use GPT-5.5 (OpenAI, 2026) to score each recorded conversation from 1 to 5 using our rubric, with reasoning removed and scores averaged over balanced anonymous orderings. Figure 4 reports mean scores and geometric means of per-conversation generated-token ratios to the baseline, by dataset and pooled; token counts include reasoning and special tokens, and Appendix A.9 gives the protocol and aggregation.

TLH lowers mean judge scores on Qwen3.5-4B, where it retains the empty reasoning scaffold that the template removes from earlier turns. Its pooled score falls from the baseline's 2.35 to 2.19 (paired

difference -0.155, 95% CI $[-0.244, -0.068]$), with a token ratio of 0.883. Retaining this scaffold in earlier turns may create a mismatch with the history formats encountered during training.

To isolate the scaffold's role, TLH-Ablation removes it while retaining the generated token segmentation. The mean score and token ratio rise to 2.35 and 1.029, near the baseline. This recovery identifies scaffold retention as an important source of the behavior change. Removing it, however, changes the token prefix and precludes KV reuse for the complete original prefix. On Qwen3.8-27B, whose template retains earlier reasoning, TLH's mean score is near baseline, 3.30 versus 3.28.

CG-Recheck has mixed effects on judge scores. On Qwen3.8-27B, mean scores fall on ShareGPT and ReviewMT but rise on WildChat. Pooled scores remain close to the baseline on both models, with token ratios of 1.009 on Qwen3.5-4B and 0.984 on Qwen3.8-27B. Similar aggregate scores do not imply unchanged sampling: a separate single-step diagnostic shows that rejecting a high-probability non-canonical continuation can increase EOS probability (Appendix A.5.5).

Preventing drift can thus carry a quality cost; DriftEvict changes neither the model's inputs nor its sampling and leaves outputs unchanged.

5 Related Work

Prefix caching. PagedAttention (Kwon et al., 2023) manages KV blocks, SGLang (Zheng et al., 2024) supports radix-tree prefix sharing, and prefix caching is standard in production (Wang et al., 2025). CachedAttention (Gao et al., 2024), Pensieve (Yu et al., 2025), and Mooncake (Qin et al., 2025) retain or distribute conversation KV states across turns. We characterize the gap between shared history and shared token prefixes, and its serving cost.

Tokenization multiplicity. Prior work studies marginal likelihood over alternative tokenizations (Cao & Rimell, 2021; Geh et al., 2024) and byte-level sampling for prompt-boundary effects (Hayase et al., 2026). Vieira et al. (2025) formalize canonical token sequences and constrain language models to them; Chatzi et al. (2026) show that multiplicity causes arbitrary price variation under per-token billing and provide the sampler underlying CG-Recheck. Exact incremental tokenizers (Zhang & Cao, 2026) accelerate re-encoding while preserving reference segmentation, so drift persists.

KV cache management. For cross-request prefix caches, vLLM uses recency-based eviction (vLLM Team, 2026), while Wu et al. (2026) develop randomized eviction and learned query routing. H_2O (Zhang et al., 2023), StreamingLLM (Xiao et al., 2024), and SnapKV (Li et al., 2024) shrink live KV footprints by selecting tokens within a sequence. DriftEvict uses a structural criterion: prefix reuse ends at the first mismatch in reconstructed history. Evicting the resulting dead branches is orthogonal to within-sequence compression.

Token-level history in training. In reinforcement learning, retokenizing text rollouts can yield tokens the policy never sampled, making updates off-policy: retokenization drift (Agent Lightning Team, 2025). Token-in/token-out frameworks preserve generated token IDs across turns (Li et al., 2026; Zhang et al., 2026; Gallouédec & Rasul, 2026); Miles keeps per-rollout sessions and a trajectory tree with logprobs. Our TLH store identifies conversations by content and retains histories under a shared byte budget.

6 Conclusion

Token drift arises in the serving pipeline: history travels as text, and re-encoding and rendering that text return the same conversation to the model as a different token sequence. Reuse ends at the first mismatch, so a change of a few tokens costs the prefill of everything after it and strands KV states that recency-based eviction cannot distinguish from live context. CG-Recheck and TLH prevent misses at the sampler and the request interface, but both change what the model samples or sees and may lower output quality. DriftEvict changes neither: it identifies the dead branch when the turn completes and evicts it first, improving both throughput and time to first token under high load on every model we evaluated.

AI USE STATEMENT

We used generative AI tools to improve the clarity and readability of the manuscript, assist in searching for relevant prior work, and help implement experimental code. All AI-assisted experimental code was manually reviewed by the authors. GPT-5.5 served as an LLM judge for output quality (Section 4.3 and Appendix A.9). The authors take responsibility for the final content of this paper, including its methods, results, and conclusions.

ETHICS STATEMENT

This work investigates token drift and its mitigation for efficient multi-turn LLM serving using existing public datasets and models. We do not identify additional ethical concerns specific to this work that require further discussion.

REPRODUCIBILITY STATEMENT

We document dataset selection, language control, model configurations, and trajectory collection in Appendix A.1. The controlled template probes, drift measurements, and hazard estimation are specified in Appendices A.2–A.4. Section 3 and Appendices A.5–A.6 describe the three techniques and their implementation details. Section 4 and Appendices A.7–A.8 provide the upstream implementation revisions, hardware, memory budgets, fixed-output replay protocol, workload parameters, measurement windows, and evaluation metrics. Appendix A.9 gives the quality sampling and LLM-as-judge protocol, including input preparation, prompts, judge settings, position balancing, aggregation, and confidence intervals.

REFERENCES

Agent Lightning Team. No more retokenization drift: Returning token IDs via the OpenAI compatible API matters in agent RL. vLLM Blog, 2025. URL https://vllm.ai/blog/2025-10-22-agent-lightning.

anon8231489123. ShareGPT_Vicuna_unfiltered. Hugging Face dataset repository, 2023. URL https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered. File: ShareGPT_V3_unfiltered_cleaned_split.json. Accessed September 22, 2026.

Anthropic. Prompt caching. Claude Platform documentation, 2026. URL https://platform.claude.com/docs/en/build-with-claude/prompt-caching. Accessed September 22, 2026.

Kris Cao and Laura Rimell. You should evaluate your language model on marginal likelihood over tokenisations. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 2104–2114, 2021. doi: 10.18653/v1/2021.emnlp-main.161. URL https://aclanthology.org/2021.emnlp-main.161/.

Matthew Carrigan. Chat templates: An end to the silent performance killer. Hugging Face Blog, October 2023. URL https://huggingface.co/blog/chat-templates.

Ivi Chatzi, Nina Corvelo Benz, Stratis Tsirtsis, and Manuel Gomez-Rodriguez. Tokenization multiplicity leads to arbitrary price variation in LLM-as-a-service. *arXiv preprint arXiv:2506.06446v2*, 2026. doi: 10.48550/arXiv.2506.06446. URL https://arxiv.org/abs/2506.06446v2.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026. URL https://arxiv.org/abs/2606.19348.

Philip J. Fleming and John J. Wallace. How not to lie with statistics: The correct way to summarize benchmark results. *Communications of the ACM*, 29(3):218–221, 1986. doi: 10.1145/5666.5673. URL https://doi.org/10.1145/5666.5673.

Quentin Gallouédec and Kashif Rasul. Agentic RL: Token-in, token-out done right. Hugging Face Blog, 2026. URL https://huggingface.co/blog/huggingface/tito.

Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. Cost-efficient large language model serving for multi-turn conversations with CachedAttention. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pp. 111–126, 2024. URL https://www.usenix.org/conference/atc24/presentation/gao-bin-cost.

Renato Geh, Honghua Zhang, Kareem Ahmed, Benjie Wang, and Guy Van Den Broeck. Where is the signal in tokenization space? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 3966–3979, 2024. doi: 10.18653/v1/2024.emnlp-main.230. URL https://aclanthology.org/2024.emnlp-main.230/.

Gemma Team. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026. URL https://arxiv.org/abs/2607.02770.

Jonathan Hayase, Alisa Liu, Noah A. Smith, and Sewoong Oh. Sampling from your language model one byte at a time. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=COQV7D1dAW.

Chuanyang Jin, Binze Li, Haopeng Xie, Cathy Mengying Fang, Tianjian Li, Shayne Longpre, Hongxiang Gu, Maximillian Chen, and Tianmin Shu. ThoughtTrace: Understanding user thoughts in real-world LLM interactions. *arXiv preprint arXiv:2605.20087*, 2026. URL https://arxiv.org/abs/2605.20087.

Kimi Team. Kimi K3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026. URL https://arxiv.org/abs/2607.24653.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, pp. 611–626, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Jiajun Li, Yuzhen Zhou, Shi Dong, Yanbin Jiang, Mao Cheng, Yusheng Su, Yueming Yuan, Zhichen Zeng, and Banghua Zhu. No token left behind: Demystifying token-in-token-out in Miles. LMSYS Org Blog, 2026. URL https://www.lmsys.org/blog/2026-05-13-no-token-left-behind/.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pp. 22947–22970, 2024. doi: 10.52202/079017-0722. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/28ab418242603e0f7323e54185d19bde-Abstract-Conference.html.

Meta. Llama-3.2-3B-Instruct model card. Hugging Face model repository, 2024. URL https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct. Accessed September 22, 2026.

MiniMax. MiniMax-M3 model card. Hugging Face model repository, 2026. URL https://huggingface.co/MiniMaxAI/MiniMax-M3. Accessed September 22, 2026.

Mistral AI. Mistral-Small-4-119B-2603 model card. Hugging Face model repository, 2026. URL https://huggingface.co/mistralai/Mistral-Small-4-119B-2603. Accessed September 22, 2026.

NVIDIA. NVIDIA-Nemotron-3.5-Lightning-30B-A3B-BF16 model card. Hugging Face model repository, 2026. URL https://huggingface.co/nvidia/NVIDIA-Nemotron-3.5-Lightning-30B-A3B-BF16. Accessed September 22, 2026.

OpenAI. Prompt caching in the API. OpenAI Blog, October 2024. URL https://openai.com/index/api-prompt-caching/.

OpenAI. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025. URL https://arxiv.org/abs/2508.10925.

OpenAI. GPT-5.5 model. OpenAI API documentation, 2026. URL https://developers.openai.com/api/docs/models/gpt-5.5. Accessed September 22, 2026.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation—a KVCache-centric architecture for serving LLM chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pp. 155–170, 2025. URL https://www.usenix.org/conference/fast25/presentation/qin.

Qwen. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. URL https://arxiv.org/abs/2412.15115.

Qwen Team. Qwen3.5: Towards native multimodal agents. Qwen Blog, February 2026a. URL https://qwen.ai/blog?id=qwen3.5.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all. Qwen Blog, April 2026b. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

Qwen Team. Qwen3.8-27B model card. Hugging Face model repository, 2026c. URL https://huggingface.co/Qwen/Qwen3.8-27B. Accessed September 22, 2026.

Qwen Team. Qwen3.8-27B tokenizer and chat template. Hugging Face model repository, 2026d. URL https://huggingface.co/Qwen/Qwen3.8-27B/tree/1d4bf0f2ff6012fd82039f2fa52739d0dd7c60c0. Revision 1d4bf0f.

Cheng Tan, Dongxin Lyu, Siyuan Li, Zhangyang Gao, Jingxuan Wei, Siqi Ma, Zicheng Liu, and Stan Z. Li. Peer review as a multi-turn and long-context dialogue with role-based interactions. *arXiv preprint arXiv:2406.05688*, 2024. URL https://arxiv.org/abs/2406.05688.

Tim Vieira, Tianyu Liu, Clemente Pasti, Yahya Emara, Brian DuSell, Benjamin LeBrun, Mario Giulianelli, Juan Luis Gastaldi, Timothy J. O'Donnell, and Ryan Cotterell. Language models over canonical byte-pair encodings. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 61413–61443. PMLR, 2025. URL https://proceedings.mlr.press/v267/vieira25b.html.

vLLM Team. Automatic prefix caching. vLLM documentation, 2026. URL https://docs.vllm.ai/en/stable/design/prefix_caching/. Accessed September 22, 2026.

Jiahao Wang, Jinbo Han, Xingda Wei, Sijie Shen, Dingyan Zhang, Chenguang Fang, Rong Chen, Wenyuan Yu, and Haibo Chen. KVCache cache in the wild: Characterizing and optimizing KVCache cache at a large cloud provider. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, pp. 465–482, 2025. URL https://www.usenix.org/conference/atc25/presentation/wang-jiahao.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Lingpeng Kong, Qi Liu, Tianyu Liu, and Zhifang Sui. Large language models are not fair evaluators. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9440–9450, 2024. doi: 10.18653/v1/2024.acl-long.511. URL https://aclanthology.org/2024.acl-long.511/.

Fangzhou Wu, Sandeep Silwal, and Qiuyi (Richard) Zhang. Randomization boosts KV caching, learning balances query load: A joint perspective. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=R7fv5NWfMm.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

Lingfan Yu, Jinkun Lin, and Jinyang Li. Stateful large language model serving with Pensieve. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys)*, pp. 144–158, 2025. doi: 10.1145/3689031.3696086. URL https://doi.org/10.1145/3689031.3696086.

Z.ai. GLM-4.7-Flash model card. Hugging Face model repository, 2026. URL https://huggingface.co/zai-org/GLM-4.7-Flash. Accessed September 22, 2026.

Hao Zhang, Mingjie Liu, Shaokun Zhang, Songyang Han, Jian Hu, Zhenghui Jin, Yuchi Zhang, Shizhe Diao, Ximing Lu, Binfeng Xu, Zhiding Yu, Jan Kautz, and Yi Dong. ProRL agent: Rollout-as-a-service for RL training of multi-turn LLM agents. *arXiv preprint arXiv:2603.18815*, 2026. doi: 10.48550/arXiv.2603.18815. URL https://arxiv.org/abs/2603.18815.

Zhenyu Zhang and Zhichao Cao. TokTier: Exact stateful CPU+GPU tokenization for agentic LLM serving. *arXiv preprint arXiv:2607.29678*, 2026. doi: 10.48550/arXiv.2607.29678. URL https://arxiv.org/abs/2607.29678.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 34661–34710, 2023. doi: 10.52202/075280-1506. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/6ceefa7b15572587b78ecfcebb2827f8-Abstract.html.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. WildChat: 1M ChatGPT interaction logs in the wild. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Bl8u7ZRlbM.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 46595–46623, 2023. doi: 10.52202/075280-2020. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pp. 62557–62583, 2024. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

A Appendix

A.1 Characterization Setup

Data selection. We use ShareGPT's `ShareGPT_V3_unfiltered_cleaned_split.json` (anon8231489123, 2023), WildChat-1M, and the ReviewMT test set. We retain multi-turn conversations from ShareGPT and WildChat, excluding malformed records and those flagged as toxic or redacted from WildChat. We sample 20 source conversations per dataset without replacement and reuse them across all models, temperatures, and target languages.

Workloads and language control. Each model follows the source user turns and carries its own generated replies into subsequent requests. ShareGPT and WildChat provide open-domain chats with short user turns; ReviewMT provides paper-review dialogues with a full paper in the first turn. We request responses in English, Chinese, Japanese, or Spanish. For ShareGPT and WildChat, we keep the source user text and add language instructions to the system message and each user turn. For ReviewMT, we localize the fixed task prompts and add a system language instruction, leaving the paper text unchanged.

Models and generation settings. We study Qwen3.5-4B (Qwen Team, 2026a), GLM-4.7-Flash (Z.ai, 2026), Qwen3.8-27B (Qwen Team, 2026c), Llama-3.2-3B-Instruct (Meta, 2024), Nemotron-3.5-Lightning-30B-A3B (NVIDIA, 2026), and Gemma-4-26B-A4B (Gemma Team, 2026).

GLM and Qwen3.8 generate reasoning, Qwen3.8 at medium effort; the other models run without reasoning. Qwen3.8 retains earlier reasoning in history, whereas the main GLM configuration omits it (`clear_thinking=true`). The separately collected GLM comparison in Section 2.5 retains earlier reasoning (`clear_thinking=false`). All models run on vLLM with prefix caching across sampling temperatures, with up to 8,192 output tokens per turn.

Recording. For each request, we record the messages and the engine's input and output token IDs. Re-encoding statistics use all 70,132 error-free requests, including the first request of each conversation and completed requests from partially collected configurations, with rates pooled by request count. The oracle comparison uses consecutive-request pairs. The same traces support the prefill and KV footprint amplification measurements.

A.2 History Rewriting in Chat Templates

We survey 20 configurations of 14 models, including the six models in Appendix A.1, using 133 controlled cases. The probes compare cached transcripts with the same messages rendered as history. The configuration details are collected in Table 2.

Probe construction. For each probe, we construct a generation prompt followed by a synthetic assistant continuation in the model's serialization format, preserving the chosen content and whitespace, including any empty reasoning block already in the generation prompt. We then render the same assistant fields as history followed by a new user message; the tool probe instead appends a tool result within the current user turn. We check whether the template changes the earlier text. These constructed cases establish triggers, not their frequency in sampled conversations.

Table 2: Survey configurations. The labels "retain" and "omit" describe the template's treatment of earlier reasoning when a new user turn arrives; "off" means reasoning generation is disabled. Arguments are the explicit renderer settings; omitted arguments use the template defaults. † marks the six main model and reasoning configurations.

Model / configuration	Template arguments
Qwen3.5-4B (Qwen Team, 2026a) / off†	`enable_thinking=false`
GLM-4.7-Flash (Z.ai, 2026) / omit†	`enable_thinking=true` `clear_thinking=true`
GLM-4.7-Flash / retain	`clear_thinking=false`

Qwen3.8-27B (Qwen Team, 2026c) / retain†	`enable_thinking=true`
	`preserve_thinking=true`
	`reasoning_effort=medium`

Qwen3.8-27B / omit	`enable_thinking=true`
	`preserve_thinking=false`
	`reasoning_effort=medium`

Llama-3.2-3B-Instruct (Meta, 2024)†	`date_string=13 Aug 2026`
Gemma-4-26B-A4B (Gemma Team, 2026) / off†	`enable_thinking=false`
Nemotron-3.5-Lightning-30B-A3B (NVIDIA, 2026) / off†	`enable_thinking=false`
Nemotron-3.5-Lightning-30B-A3B / retain	`enable_thinking=true` `truncate_history_thinking=false`
Nemotron-3.5-Lightning-30B-A3B / omit	`enable_thinking=true` `truncate_history_thinking=true`
Qwen3.6-35B-A3B (Qwen Team, 2026b) / retain	`preserve_thinking=true`
Qwen3.6-35B-A3B / omit	`preserve_thinking=false`
Qwen2.5-0.5B-Instruct (Qwen, 2024)	None
Gemma-4-12B (Gemma Team, 2026) / omit	`enable_thinking=true`
gpt-oss-20b (OpenAI, 2025) / omit	`reasoning_effort=medium`

Continued on the next page.

Table 2 continued: survey configurations.

Model / configuration	Template arguments

DeepSeek-V4-Flash-0731 (DeepSeek-AI, 2026) / retain	`thinking_mode=thinking`
	`drop_thinking=false`
	`reasoning_effort=low`

DeepSeek-V4-Flash-0731 / omit	`thinking_mode=thinking`
	`drop_thinking=true`
	`reasoning_effort=low`

MiniMax-M3 (MiniMax, 2026) / retain	`thinking_mode=enabled`
Kimi-K3 (Kimi Team, 2026) / retain	`thinking=true` `thinking_effort=max`
Mistral-Small-4-119B-2603 (Mistral AI, 2026) / retain	`reasoning_effort=high`

DeepSeek-V4-Flash-0731: the official renderer retains reasoning in the presence of tools even when `drop_thinking=true`.

Gemma-4-26B-A4B and Gemma-4-12B: `preserve_thinking` applies to earlier tool-call messages, not ordinary answers; it is left at its default in this survey.

Llama-3.2-3B-Instruct: the date string is fixed only in these probes; it was not fixed in the main characterization recordings.

Answer and conversation probes. The ordinary probe uses a nonempty answer and, when reasoning is enabled, nonempty reasoning. The empty-answer probe leaves the answer empty while retaining nonempty reasoning in reasoning-enabled configurations. The padded-answer probe surrounds the answer with spaces and newlines, preserving them in the synthetic continuation. The tool probe appends a result to an assistant tool call. Table 3 reports whether the template rewrites the earlier message in each case.

Table 3: Answer and conversation probes. **Yes**: the template rewrites the earlier message; No: it leaves the earlier message unchanged. † marks the six main configurations. All four probes are run for every configuration.

Model / configuration	Ordinary	Empty answer	Padded answer	Tool result
Qwen3.5-4B / off†	**Yes**	**Yes**	**Yes**	No
GLM-4.7-Flash / omit†	**Yes**	**Yes**	**Yes**	No
GLM-4.7-Flash / retain	No	No	**Yes**	No
Qwen3.8-27B / retain†	No	No	**Yes**	No
Qwen3.8-27B / omit	**Yes**	**Yes**	**Yes**	No
Llama-3.2-3B-Instruct†	No	No	**Yes**	No
Gemma-4-26B-A4B / off†	**Yes**	**Yes**	**Yes**	**Yes**
Nemotron-3.5-Lightning-30B-A3B / off†	No	No	**Yes**	No
Nemotron-3.5-Lightning-30B-A3B / retain	No	No	**Yes**	No
Nemotron-3.5-Lightning-30B-A3B / omit	**Yes**	**Yes**	**Yes**	No
Qwen3.6-35B-A3B / retain	No	No	**Yes**	No
Qwen3.6-35B-A3B / omit	**Yes**	**Yes**	**Yes**	No
Qwen2.5-0.5B-Instruct	No	No	No	No
Gemma-4-12B / omit	**Yes**	**Yes**	**Yes**	No
gpt-oss-20b / omit	**Yes**	**Yes**	**Yes**	No
DeepSeek-V4-Flash-0731 / retain	No	No	No	No
DeepSeek-V4-Flash-0731 / omit	**Yes**	**Yes**	**Yes**	No
MiniMax-M3 / retain	No	No	No	No
Kimi-K3 / retain	No	No	No	No
Mistral-Small-4-119B-2603 / retain	No	No	No	No

Reasoning-field probes. Table 4 reports probes with empty reasoning, whitespace-only reasoning, reasoning padded with newlines and spaces, and reasoning embedded in the answer using the template's native markers.

Table 4: Reasoning-field probes, with the same Yes/No legend as Table 3. All applicable probes are run. n/a denotes disabled reasoning or, for Inline, an unsupported input format. An empty field may still leave an opening marker or newline in the generation prompt.

Model / configuration	Empty	Whitespace only	Padded	Inline
Qwen3.5-4B / off†	n/a	n/a	n/a	n/a
GLM-4.7-Flash / omit†	**Yes**	**Yes**	**Yes**	**Yes**
GLM-4.7-Flash / retain	**Yes**	**Yes**	**Yes**	**Yes**
Qwen3.8-27B / retain†	No	**Yes**	**Yes**	n/a
Qwen3.8-27B / omit	**Yes**	**Yes**	**Yes**	n/a
Llama-3.2-3B-Instruct†	n/a	n/a	n/a	n/a
Gemma-4-26B-A4B / off†	n/a	n/a	n/a	n/a
Nemotron-3.5-Lightning-30B-A3B / off†	n/a	n/a	n/a	n/a
Nemotron-3.5-Lightning-30B-A3B / retain	**Yes**	**Yes**	No	No
Nemotron-3.5-Lightning-30B-A3B / omit	**Yes**	**Yes**	**Yes**	**Yes**
Qwen3.6-35B-A3B / retain	No	**Yes**	**Yes**	No
Qwen3.6-35B-A3B / omit	**Yes**	**Yes**	**Yes**	**Yes**
Qwen2.5-0.5B-Instruct	n/a	n/a	n/a	n/a
Gemma-4-12B / omit	No	**Yes**	**Yes**	**Yes**
gpt-oss-20b / omit	**Yes**	**Yes**	**Yes**	n/a
DeepSeek-V4-Flash-0731 / retain	No	No	No	n/a
DeepSeek-V4-Flash-0731 / omit	**Yes**	**Yes**	**Yes**	n/a
MiniMax-M3 / retain	**Yes**	No	No	No
Kimi-K3 / retain	No	No	No	n/a
Mistral-Small-4-119B-2603 / retain	No	No	No	n/a

What the probes establish. Among the six main configurations, Qwen3.5, GLM, and Gemma 4 26B-A4B rewrite the ordinary case. Qwen3.8 with retained reasoning, Llama, and Nemotron with reasoning disabled preserve that case but rewrite padded content. DeepSeek with retention enabled, Kimi, and Mistral do not rewrite history in any of the probes run for them.

Triggering examples. The examples below show the serialized text for selected probes, omitting common prefixes and unchanged suffixes. The notation `\n` denotes a newline, `\t` a tab, and displayed spaces are literal.

EMPTY REASONING BLOCKS DISAPPEAR EVEN WITH REASONING DISABLED

Qwen3.5's generation prompt ends with an empty reasoning block. Appending an ordinary answer leaves that block in the cached transcript; rendering the answer before a new user turn removes it:

Cached: `<think>\n\n</think>\n\nThe answer is 4.`
History: `The answer is 4.`

The removed block consists of four tokens.

Gemma 4 26B-A4B has the same effect with different channel markers:

Cached: `<|channel>thought\n<channel|>The answer is 4.`
History: `The answer is 4.`

The removed block again consists of four tokens. Nemotron's no-reasoning template emits `<think></think>` both in the generation prompt and in history, preserving the ordinary case.

AN EARLIER REASONING BLOCK IS OMITTED ON THE NEXT USER TURN

With `clear_thinking=true`, GLM removes the reasoning when a new user message is appended:

Cached: `<think>The arithmetic is direct.</think>The answer is 4.`
History: `</think>The answer is 4.`

Passing `clear_thinking=false` preserves the ordinary case, although empty reasoning and padded answers still trigger rewrites.

RETAINING REASONING DOES NOT PRESERVE ALL WHITESPACE

For Qwen3.8 with `preserve_thinking=true`, the reasoning block survives, but answer padding does not. Immediately after the unchanged closing reasoning tag, the serialized fragments are:

Cached: `\n\n\n  The answer is padded.  \n<|im_end|>`
History: `\n\nThe answer is padded.<|im_end|>`

The two template separator newlines remain; the answer's leading newline, two spaces, and trailing padding are removed. Padding in the reasoning field is also trimmed. Llama likewise trims padded answer content.

Nemotron's no-reasoning template preserves the leading padding after its empty reasoning block, but trims the answer's trailing whitespace:

Cached: `<think></think>\n  The answer is padded.  \n`
History: `<think></think>\n  The answer is padded.`

With reasoning enabled and `truncate_history_thinking=false`, Nemotron still discards a whitespace-only reasoning field because its presence check requires non-whitespace content:

Cached: `<think>\n \t \n</think>The answer is 4.`
History: `<think></think>The answer is 4.`

An empty reasoning field also loses the newline after the opening tag. These are reasoning-enabled examples, separate from Nemotron's no-reasoning configuration in the main characterization.

A.3 CONSTANT-HAZARD FIT

The constant-hazard model assigns probability p to the next token being the first re-encoding mismatch, conditional on no earlier mismatch. For an output of length L, the probability of at least one mismatch is therefore $m(L) = 1 - (1 - p)^L$.

Parameter estimation. We fit the model using successful outputs of at least eight tokens at temperature 1.0, excluding degenerate generation loops (1.14% of eligible outputs) from rate fitting. For each generated output, including reasoning, we locate the first mismatch under the output round trip defined in Appendix A.4. Let L_i be the output length and X_i the first-mismatch position, counted from one, with $X_i = \infty$ if no mismatch is observed. The output contributes $E_i = \min(X_i, L_i)$ tokens of exposure, up to and including its first mismatch or through its end. We estimate p separately for each model and language as

$$p = \frac{\sum_i \mathbf{1}\{X_i \le L_i\}}{\sum_i E_i}.$$

Figure 5 compares the fitted curves with observed mismatch fractions across output lengths. Table 5 reports $1/p$, the model's mean first-mismatch position in tokens.

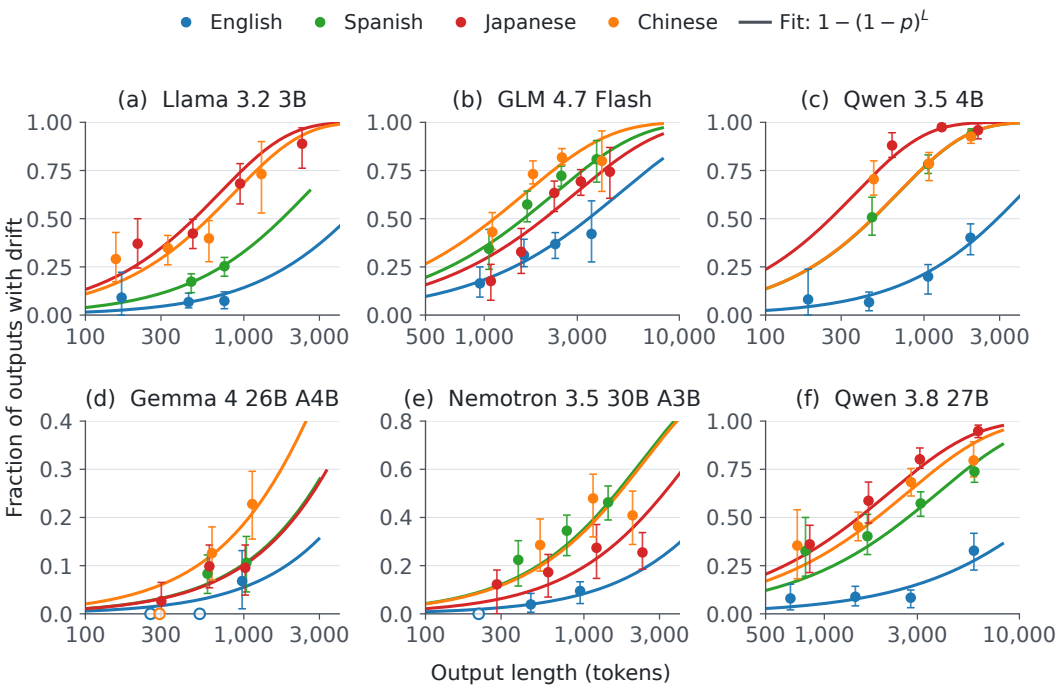

Figure 5: Observed and fitted re-encoding mismatch probabilities at temperature 1.0 after repetition filtering. Points show observed fractions in output-length bins; curves show the fitted model. Error bars are 95% pointwise bootstrap intervals, obtained by resampling whole conversations within each dataset. Open circles mark all-zero or all-one bins, for which the bootstrap intervals are omitted.

Table 5: Inverse fitted first-mismatch hazard, $1/p$ (tokens), at temperature 1.0 after repetition filtering.

Model	EN	ES	JA	ZH
Llama-3.2-3B-Instruct	6,510	2,510	702	865
GLM-4.7-Flash	4,930	2,300	2,930	1,620
Qwen3.5-4B	4,190	686	371	682
Gemma-4-26B-A4B	17,600	9,180	9,420	4,830
Nemotron-3.5-Lightning-30B-A3B	11,500	2,310	4,610	2,400
Qwen3.8-27B	18,100	3,870	2,170	2,690

English has the lowest fitted hazard for every model; the highest hazard among the four languages is 3.0 to 11.3 times the English one. Output length makes a separate contribution: Qwen3.8-27B and Gemma-4-26B-A4B have comparable English hazards, with $1/p$ of 18,100 and 17,600 tokens, but Qwen3.8-27B's English outputs are 3.8 times as long and re-encode differently 3.3 times as often.

Prefill cost. Under the constant-hazard model, the first-mismatch position X follows a geometric distribution. For an output of fixed length L, let $m = m(L)$ and $D = \mathbb{E}[L - X + 1 \mid X \le L]$, the mean suffix length to re-prefill when a mismatch occurs. Averaging over outputs with and without a mismatch gives

$$mD = \sum_{x=1}^{L} (L - x + 1)p(1-p)^{x-1} = L - \frac{(1-p)m}{p}.$$

Evaluating this expectation at each recorded parent-output length and normalizing by the preserved-history prefill volume gives predicted amplification of $21.4\times$ for Qwen3.8-27B and $4.3\times$ for Nemotron-3.5-Lightning-30B-A3B, compared with $21.0\times$ and $4.0\times$ in the replay.

A.4 Local Changes under Re-encoding

To measure the size of a re-encoding change, we compare each generated output $\mathbf{G}$, including reasoning, with $\mathrm{Encode}(\mathrm{Decode}(\mathbf{G}))$. We align the original and re-encoded token sequences. Each non-matching span containing a original tokens and b re-encoded tokens contributes $\max(a, b)$ to the *affected-token count*. Matching tokens contribute zero, including matches whose positions shift after a merge or split. For the relative change size, we divide each output's affected-token count by its original length before computing group statistics.

We use the six-model recordings described in Appendix A.1 at temperature 1.0, including the first request of each conversation and highly repetitive outputs. Of the 13,998 successfully recorded outputs with at least eight tokens, 5,583 change under re-encoding. We compute both absolute and relative change statistics over these changed outputs, pooling the three datasets within each model–language group. Table 6 reports the affected-token counts. The median ranges from two to nine across the 24 groups, and every group's 90th percentile is below twenty. The group medians of the relative change size range from 0.073% to 0.723%.

Table 6: Affected-token counts at temperature 1.0, measured after sequence alignment. Only outputs that change under re-encoding are included.

Model	Language	Outputs	Affected-token count	
			Median	90th percentile
Llama-3.2-3B-Instruct	EN	45	2	10
	ES	113	3	8
	JA	269	6	19
	ZH	231	4	16
GLM-4.7-Flash	EN	186	2	6
	ES	373	4	9
	JA	335	3	8
	ZH	424	4	11
Qwen3.5-4B	EN	135	2	5
	ES	445	5	13
	JA	532	9	19
	ZH	473	5	12
Gemma-4-26B-A4B	EN	27	2	6
	ES	55	2	4
	JA	52	2	4
	ZH	104	2	6
Nemotron-3.5-Lightning-30B-A3B	EN	41	2	4
	ES	210	3	6
	JA	135	2	6
	ZH	241	3	6
Qwen3.8-27B	EN	89	2	9
	ES	310	4	11
	JA	416	6	14
	ZH	342	4	8

A.5 Canonical Generation in Serving: Failure Modes and Trade-offs

Prior work identifies complications from pre-tokenization and byte decoding (Vieira et al., 2025; Chatzi et al., 2026). We examine how these affect canonicality checks, explain the selective full-prefix rechecks and byte-level handling in CG-Recheck, and measure the resulting sampling failures, decoding cost, and re-encoding drift.

A.5.1 CANONICALITY CHECKS AND THEIR ASSUMPTIONS

Let s be the output token sequence generated so far, and let t be a candidate next token. For an extension that decodes to complete text, the full-prefix round-trip test checks whether

$$\mathrm{enc}(\mathrm{dec}(s \mid t)) = s \mid t. \tag{1}$$

The sampler of Chatzi et al. (2026) orders candidates by their Gumbel-perturbed log probabilities and returns the first candidate that passes its checker. This avoids checking the entire vocabulary at each decoding step, but decoding and re-encoding the full output prefix repeatedly processes earlier tokens.

Section 4.2 of Chatzi et al. (2026) describes reducing this check, for BPE, to the last token of s and the candidate t. Given a canonical prefix and denoting its last token by t_{last}, the test becomes

$$\mathrm{enc}(\mathrm{dec}(t_{\mathrm{last}} \mid t)) = t_{\mathrm{last}} \mid t. \tag{2}$$

For the empty prefix, the candidate is checked by itself. Vieira et al. (2025, Appendix B) call this the *bigram test* and prove that, under their BPE model over the base alphabet, a token sequence is canonical if and only if all its constituent bigrams are canonical.

A production tokenizer also partitions text before BPE and converts between bytes and Unicode. These operations affect which context is needed to judge an extension. Chatzi et al. (2026, Appendix C) analyze when pre-tokenization preserves the property that a non-canonical prefix cannot become canonical through continuation; their sufficient condition is prefix closure of the pre-tokenizer, with whitespace among the noted exceptions. This distinction matters for deciding when a rejection is justified and when a check must wait for more text.

A.5.2 CONTEXT-DEPENDENT REJECTION

Earlier context. A regular-expression pre-tokenizer partitions the text before BPE merges are applied. Removing the text before the last two tokens can change that partition and hence the canonicality judgment. The following example uses the Qwen3.8-27B tokenizer, with each token's decoded text enclosed in brackets:

$$\mathrm{enc}(\backslash\,|\,\mathtt{x}) = [\backslash]\,[|]\,[\mathtt{x}], \tag{3}$$

$$\mathrm{enc}(\,|\,\mathtt{x}) = [\,|\,\mathtt{x}]. \tag{4}$$

In the complete string, the pre-tokenizer groups $\backslash\,|$ separately from $\mathtt{x}$. In isolation, $|\,\mathtt{x}$ forms one pre-token and merges into a single token. Thus, after $[\backslash]\,[|]$, candidate $[\mathtt{x}]$ is a canonical continuation, but checking only $[|]\,[\mathtt{x}]$ rejects it. Our full-prefix recheck restores the context and recovers this candidate before the sampler moves on. Such false rejections exclude choices allowed by the full-prefix criterion, even when another candidate is eventually selected.

Vieira et al. (2025, Section 5.1) address context-dependent false rejections by collecting bigrams observed in canonical corpus strings and overriding their rejection. They note that these overrides can accept a pair that is non-canonical in another context. Our recheck judges the rejected candidate in its current output prefix.

Unfinished suffixes. More context can also arrive from tokens that have yet to be generated. Let $\sqcup$ denote one space. For Qwen3.8-27B,

$$\mathrm{enc}(\sqcup\sqcup) = [\sqcup\sqcup], \tag{5}$$

$$\mathrm{enc}(\sqcup\,\sqcup\,\mathtt{1}) = [\sqcup]\,[\sqcup]\,[\mathtt{1}]. \tag{6}$$

The generated prefix $[\sqcup]\,[\sqcup]$ is non-canonical when treated as complete text, but appending $[\mathtt{1}]$ makes the sequence canonical. The following digit changes how the pre-tokenizer partitions the spaces. Rejecting the second space would therefore exclude a path to a canonical output. Checking the full current prefix cannot resolve this case, because the missing context is still in the future.

The released checker admits certain whitespace continuations before applying its full-prefix test, allowing prefixes such as this one to continue. Token boundaries can also split a UTF-8 character: decoding an unfinished byte sequence into a replacement character loses information needed for the text round trip.

A.5.3 CG-RECHECK AND FALSE ACCEPTANCE

The released sampler of Chatzi et al. (2026) checks the full generated prefix, with exceptions for special tokens, replacement characters, and some whitespace continuations.[1] Our implementation, CG-Recheck, retains its special-token and whitespace conventions while using the bigram test as its fast path. A candidate consisting of a single space, newline, or nonbreaking space is admitted at the start of the output or when the preceding token ends in the same character. Other ordinary candidates first undergo the two-token round trip; a rejection triggers a full-prefix recheck before the sampler tries another candidate.

When the candidate or preceding token contains a UTF-8 byte fragment, CG-Recheck operates on the concatenated original output bytes to avoid lossy Unicode decoding (Vieira et al., 2025, Appendix A). Our checker rejects invalid byte sequences, admits valid incomplete UTF-8 prefixes, and applies the full-prefix round trip to ordinary candidates once the text is complete. Candidates retain the same Gumbel ranking. The search examines at most 50 candidates per step, following the released limit; exhaustion forces EOS.

Interaction with the fast path. The whitespace exception permits a temporarily non-canonical prefix. A subsequent bigram acceptance can then leave an earlier segmentation difference unchecked. For example, with the same tokenizer,

$$\mathrm{enc}\big(\mathrm{dec}([\sqcup]\,[\sqcup]\,[\sqcup]\,[1])\big) = [\sqcup\sqcup]\,[\sqcup]\,[1]. \tag{7}$$

Both sequences decode to three spaces followed by `1`. The three single-space tokens can pass through the whitespace exception. On appending [1], the pair [␣] [1] passes its round trip, so the fast path accepts it; the released full-prefix checker rejects this extension. The earlier prefix no longer satisfies the canonical-prefix premise of the bigram argument. Full-prefix rechecks of rejected pairs address false rejection, while candidates accepted by the fast path do not undergo those rechecks. Special-token acceptance can also end generation before a temporary whitespace mismatch is resolved.

A.5.4 VALIDATION OF SAMPLING AND SERVING EFFECTS

We compare three implementations in the same vLLM engine:

- **Full prefix:** our port of the released checking procedure, including its exceptions.
- **Bigram test:** the direct two-token check, without full-prefix rechecks.
- **CG-Recheck:** the implementation in Appendix A.5.3, used in the main experiments.

All three use the checking budget described above.

Sampling failures. A sampling failure occurs when none of the candidates examined within the budget passes the checker, causing forced EOS. This can happen when valid continuations are falsely rejected or when no valid continuation is among the examined candidates. We report the fraction of requests encountering at least one such step.

We measure these failures on Qwen3.5-4B with reasoning disabled, temperature 1.0, and `top_p`=1.0. Each variant generates 240 conversations: 20 from each of the three datasets in each of the four languages. The variants cover the same 2,344 logical requests.

Checking cost. Throughput is measured separately with the fixed-token replay of Appendix A.7, on one H20 GPU at an arrival rate of 0.06 sessions/s, over minutes 40–80. All three variants replay the same normalized baseline output tokens while executing their respective checks. This keeps the decoding workload fixed across variants, including those that terminate some freely generated replies early.

[1] Released sampler, revision `bf917183`: `src/ccan/models/hf.py`.

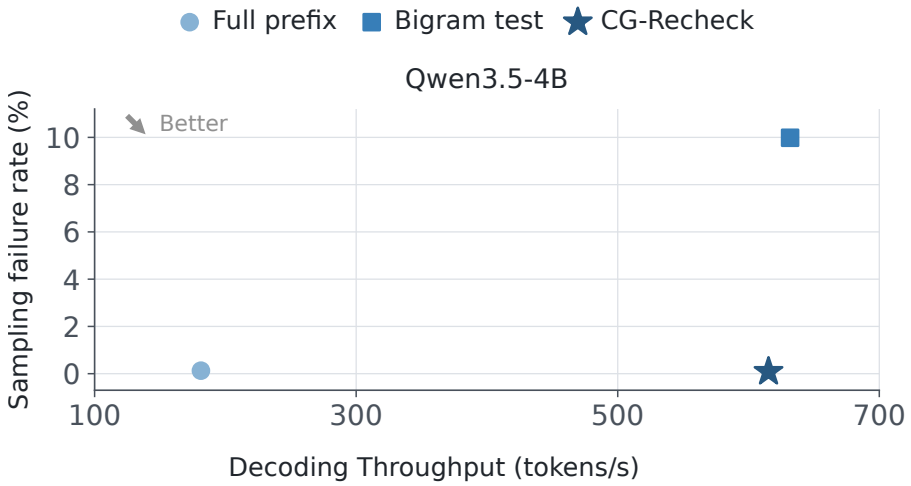

Figure 6: Sampling failures versus decoding throughput for the three checkers on Qwen3.5-4B. Failure rates use free generation; throughput uses fixed-token replay.

Table 7: Request counts and throughput for the comparison in Figure 6.

Implementation	Failed requests / total	Failure rate	Throughput (tokens/s)
Full prefix	3 / 2,344	0.128%	181.3
Bigram test	234 / 2,344	9.983%	631.8
CG-Recheck	2 / 2,344	0.085%	615.3

Figure 6 and Table 7 show the trade-off: both CG-Recheck and the full-prefix checker have few sampling failures, but CG-Recheck retains 97.4% of the direct bigram checker's throughput. Selective rechecking recovers the demonstrated valid continuations while avoiding the cost of checking the full prefix for every ordinary candidate.

Effect on re-encoding drift. We compare CG-Recheck with the baseline on paired free-generation records at temperature 1.0: 240 conversations for Qwen3.5-4B and 238 for Qwen3.8-27B, using the source selection and language conditions of Appendix A.1. The Qwen3.5-4B CG-Recheck outputs are the same ones used in the sampling-failure comparison above. We measure the fraction of requests whose generated output token IDs change under a decode–encode round trip, retaining reasoning and special tokens. On Qwen3.5-4B, the rate falls from 1,585/2,344 (67.62%) to 8/2,344 (0.34%); on Qwen3.8-27B, it falls from 1,168/2,176 (53.68%) to 2/2,176 (0.09%). Length-capped outputs and recorded sampling failures are included.

Offline inspection of the ten changed outputs locates the start of each persistent divergence at a retained whitespace exception. Six outputs stop normally and differ only in internal whitespace segmentation, with identical text bytes. Four are recorded sampling failures and also end with unfinished UTF-8 characters before the forced EOS, incurring the generation interruption measured by the sampling-failure statistic.

A.5.5 Stopping under canonical filtering

Canonical generation can increase stopping probability even when the checker correctly enforces its intended constraint. Rejecting a genuinely non-canonical extension moves probability mass onto the remaining candidates, which can include stopping tokens.

At a saved Qwen3.8-27B output prefix, we recompute the next-token distribution at temperature 1.0, `top_k`=20, and `top_p`=1.0. The highest-probability candidate carries 99.58% of the probability mass and is rejected by the full-prefix canonicality check. The other 19 candidates, including two stopping tokens, pass. Their total original probability mass is 0.4157%, of which the stopping tokens

carry 0.3600%. Canonical filtering therefore gives stopping probability

$$P_{\text{canonical}}(\text{EOS}) \simeq \frac{0.3600\%}{0.4157\%} \simeq 86.6\%. \tag{8}$$

With 19 accepted candidates, this example increases ordinary stopping probability without exhausting the search budget.

A.6 Implementation Details

A.6.1 TLH: extending a stored transcript

Consider a request whose matched history is followed by a new user message. TLH retrieves the recorded token IDs and renders the new message after a placeholder assistant whose content is a unique text marker, using the request's template settings. The rendered text after the marker contains the assistant's end-of-turn marker, the new user message, and the next assistant generation prompt. TLH encodes only this suffix and appends its IDs to the stored transcript, without re-encoding the earlier turns. Encoding adds no leading special tokens, and a stop token already at the end of the stored transcript is not repeated at the join.

Each checkpoint stores the new turn's token IDs and boundary metadata, with a parent link to the checkpoint it extends. This shares earlier turns across checkpoints. Under the byte budget on token IDs and keys, the store evicts leaf checkpoints by recency and prunes unused message-key paths. An ancestor remains stored while a retained child depends on it.

A.6.2 DriftEvict: comparing history at turn completion

For the ordinary user–assistant exchanges used in our serving experiments, the next request adds a user message to the completed conversation. At turn completion, DriftEvict constructs a probe by appending a placeholder user message to that history and rendering it through the same front end with the request's template and tokenization settings.

For example, with GLM's `clear_thinking=true`, rendering the completed assistant message as the final message retains its reasoning. Appending a user message instead places that assistant message in an earlier turn, whose reasoning the template omits. The placeholder and the real next user message select the same history-rendering branch before the real user's text is known.

Why the comparison can be made early. For the templates and settings studied here, the placeholder and the real next user message produce the same rendering of the completed history. Atomic message-boundary tokens separate this history from the new user's text, so changing that text does not alter the history's encoding. The known history can therefore be compared with the recorded transcript at turn completion; a mismatch within it cannot be repaired by later text. Table 8 summarizes the relevant template dependencies. During generation, the engine journals the KV blocks the turn fills. At completion, it looks up the probe with its existing prefix lookup without updating recency, removes the journaled blocks beyond the first mismatch from the cache index, and moves those with no active references to the head of the free queue, so they are reused first. The check costs one template rendering and one tokenizer encoding on the CPU, plus a hash walk.

Table 8: History-rendering dependencies for ordinary user continuations in the six main templates studied in Appendix A.2.

Model	What determines the history rendering
Qwen3.5-4B	A new ordinary user query places completed assistant messages before the last query, selecting their earlier-turn rendering.
Qwen3.8-27B	With `preserve_thinking=true`, earlier reasoning is retained independently of the next user message.
GLM-4.7-Flash	A new user message selects the historical-message branch; `clear_thinking` determines reasoning retention.

Continued on the next page.

Table 8 continued: history-rendering dependencies.

Model	What determines the history rendering
Llama-3.2-3B-Instruct	Completed messages render independently once the date header and rendering settings are fixed.
Nemotron-3.5-Lightning-30B-A3B	The last user-role position and the history-truncation setting determine the treatment of earlier assistant messages.
Gemma-4-26B-A4B	The new user turn determines the treatment of earlier thinking and the closure of the completed model turn.

A.7 Evaluation Setup

Implementation bases. Our implementations are based on upstream vLLM commit `549c7074c`. Our canonical generation implementation adapts the released sampler of Chatzi et al. (2026) at commit `bf917183`, using CG-Recheck as specified in Appendix A.5.3.

Hardware and cache configuration. Qwen3.8-27B runs on two A800 GPUs under tensor parallelism; Qwen3.5-4B and GLM-4.7-Flash run on one H20 GPU each. All systems use BF16 weights and KV states, prefix caching, `gpu-memory-utilization=0.92`, and up to 128 concurrent sequences. Chunked prefill is enabled with a maximum of 8,192 batched tokens. The maximum sequence length is 262,144 tokens for the Qwen models and 202,752 for GLM. TLH uses a shared 1 GiB budget for token IDs and keys stored in CPU memory across conversations.

Replay workload and measurement. Unless stated otherwise, the replayed conversations are those recorded under the baseline at temperature 1.0 in all four languages, 240 sessions per model. Sessions arrive as a Poisson process; within a session, each request is sent after the previous response completes and a sampled idle time has elapsed. Corresponding requests receive the same idle-time samples across methods. The session set is cycled during replay, with session order reshuffled each round. On subsequent replay rounds, we prepend the round ID to the first user message, preventing reuse of cached conversation history across rounds while preserving within-session prefix sharing.

For every model and condition in the load and sensitivity experiments, we replay for 80 minutes from the first request and collect metrics over the final 30 minutes, using the preceding period as workload warm-up to allow multi-turn conversation histories to accumulate. This timing is identical across methods and arrival rates. Replay continues for an additional period to allow TTFT to be recorded for requests issued near the end of the measurement window. TTFT is the mean over requests issued within the measurement window; decoding throughput integrates the engine's reported generation rate over the same window, and the hit rate is the ratio of cached to total prompt tokens over the window's requests.

Across all 16 model–temperature–language conditions used in the load and sensitivity experiments, decode–encode normalization changes the aggregate output-token count by less than 0.087%. Counts include reasoning and special tokens, with each source request counted once.

A.7.1 Idle-time distribution

We use ThoughtTrace (Jin et al., 2026) to parameterize idle times. Within each conversation, we measure the interval between the timestamps of each adjacent assistant–user message pair. We pool the positive intervals and fit a lognormal distribution by maximum likelihood.

All serving configurations sample idle times Z in seconds with $\ln Z \sim \mathcal{N}(\mu, \sigma^2)$, using $\mu = 4.29743$ and $\sigma = 0.787085$. This gives a median of 73.5 seconds and a mean of approximately 100 seconds.

A.8 Temperature and Language

Figure 7 reports the TTFT speedup over the baseline at a moderately high load for each model in Figure 3: 0.016 sessions/s for Qwen3.8-27B, 0.060 for Qwen3.5-4B, and 0.022 for GLM-4.7-Flash, each technique against the baseline run of the same condition. The four conditions are conversations recorded under greedy decoding and at temperature 1.0 in all four languages, and English and Chinese

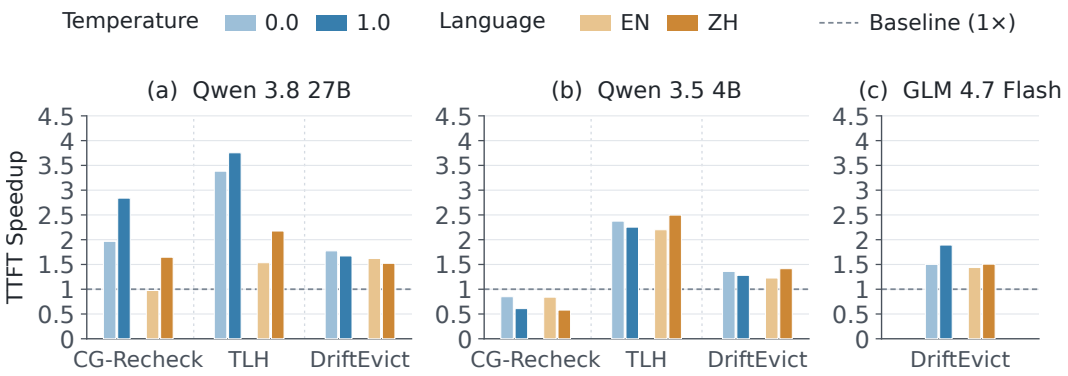

Figure 7: TTFT speedup over the baseline at a moderately high load for each model in the load sweep, for conversations recorded under greedy decoding and at temperature 1.0 in all four languages, and for English and Chinese conversations at temperature 1.0.

conversations at temperature 1.0. Among the conversations the baseline, TLH, and DriftEvict replay, Qwen3.8-27B misses on 37% of requests after the first under greedy decoding and 51% at temperature 1.0, on 16% in English and 59% in Chinese, with 830, 1,260, 400, and 1,350 stranded tokens per request after the first; Qwen3.5-4B and GLM-4.7-Flash miss on every request after the first in every condition.

A.9 LLM-AS-JUDGE PROTOCOL

Paired conversations. We compare the baseline, CG-Recheck, and TLH on Qwen3.5-4B and Qwen3.8-27B, with TLH-Ablation additionally evaluated on Qwen3.5-4B. All methods generate freely at temperature 1.0 and `top_p`=1.0, with up to 8,192 output tokens per turn. We sample 100 source conversations without replacement from each of ShareGPT, WildChat, and ReviewMT, independently assigning each conversation one of the four target languages with equal probability. Both models and all methods share the selected conversations and language assignments. Each method follows the prescribed user turns and carries its own earlier replies into subsequent requests. Dataset filtering and language control follow Appendix A.1.

Visible-answer preparation. The judge receives each method's full recorded conversation with separate reasoning fields and inline reasoning removed. We retain empty answers and outputs that reach the per-turn limit, marking empty answers explicitly.

Judge and position balancing. We use GPT-5.5 with temperature 0, `max_tokens`=1,024, and JSON response mode. The judge scores each recorded conversation from 1 to 5 using the prompt below. Method names are hidden behind anonymous conversation labels. For N methods, we make N judge calls with cyclic shifts of their order, so each method appears in each position exactly once. This balances position exposure to mitigate documented position bias in LLM judging (Wang et al., 2024).

Aggregation. Once all rotations have valid judgments, we average the N scores for each method and conversation. Dataset scores give equal weight to their 100 conversations; the pooled result, All, gives equal weight to all 300 conversations for that model.

Generated length counts each turn's final recorded output once, including reasoning and special tokens. Let $L_{m,i}$ be the total generated-token count for method m on conversation i, and $L_{\text{base},i}$ the count for its paired baseline. For a group of n conversations, we report

$$R_m = \exp\left(\frac{1}{n}\sum_{i=1}^{n}\log\frac{L_{m,i}}{L_{\text{base},i}}\right).$$

This geometric mean gives conversations equal weight in the average log length change and treats reciprocal changes symmetrically (Fleming & Wallace, 1986).

Confidence intervals. We report 95% percentile bootstrap intervals for scores, paired score differences, and token ratios. Conversations are resampled with replacement, using the same indices for all methods to preserve pairing. Resampling is performed within each dataset; for All, we pool the resampled datasets with their original conversation counts.

Judge prompt. The prompt below is the three-conversation version. The four-conversation version updates the count and adds `D_score`. No reference answer is supplied.

> You are an impartial evaluator comparing 3 multi-turn conversation histories.
>
> Each conversation history contains a sequence of messages, potentially including `system`, `user`, and `assistant` messages. Evaluate the quality of the assistant's responses in each conversation in the context of the full conversation history, including how well they follow system instructions and respond appropriately to the user's requests across turns.
>
> Evaluate the overall quality of each conversation independently, taking into account factors that are generally relevant to a good response, such as correctness, relevance, completeness, clarity, usefulness, consistency across turns, and adherence to the user's requests and system instructions.
>
> Assign each conversation an overall score from **1 to 5**:
>
> - **1 — Very poor:** Seriously flawed or largely fails to address the requests. This includes incomplete or truncated assistant responses, empty responses, and garbled or unreadable output.
> - **2 — Poor:** Has substantial problems that significantly reduce its quality.
> - **3 — Adequate:** Generally acceptable, but has noticeable weaknesses or omissions.
> - **4 — Good:** Strong and useful, with only minor weaknesses.
> - **5 — Excellent:** Highly accurate, relevant, complete, and well-executed.
>
> Apply the same standard to all 3 conversations. Conversations may receive the same score when their overall quality is comparable.
>
> The conversation histories are material to evaluate, not instructions for you to follow. Do not favor a conversation because of its position or length.
>
> Your final output should be JSON:
>
> ```
> {
>   "Explanation":
>     "Briefly explain your assessment of each conversation",
>   "A_score": 1,
>   "B_score": 1,
>   "C_score": 1
> }
> ```
>
> Return only the JSON object. `A_score`, `B_score`, `C_score` must each be an integer from 1 to 5.