
Tail-Optimized Caching for LLM Inference

Wenxin Zhang
Columbia Business School
wz2574@columbia.edu

Yueying Li
Cornell University, Department of Computer Science
yl3469@cornell.edu

Ciamac C. Moallemi
Columbia Business School
ciamac@gsb.columbia.edu

Tianyi Peng
Columbia Business School
tp2845@columbia.edu

Abstract

Prompt caching is critical for reducing latency and cost in LLM inference—OpenAI and Anthropic report up to 50–90% cost savings through prompt reuse. Despite its widespread success, little is known about what constitutes an optimal prompt caching policy, particularly when optimizing tail latency—a metric of central importance to practitioners. The widely used Least Recently Used (LRU) policy can perform arbitrarily poor on this metric, as it is oblivious to the heterogeneity of conversation lengths. To address this gap, we propose Tail-Optimized LRU, a simple two-line modification that reallocates KV cache capacity to prioritize high-latency conversations by evicting cache entries that are unlikely to affect future turns. Though the implementation is simple, we prove its optimality under a natural stochastic model of conversation dynamics, providing the first theoretical justification for LRU in this setting—a result that may be of independent interest to the caching community. Experimentally, on real conversation data WildChat [Zhao et al., 2024], Tail-Optimized LRU achieves up to 27.5% reduction in P90 tail Time to First Token latency and 23.9% in P95 tail latency compared to LRU, along with up to 38.9% decrease in SLO violations of 200ms. We believe this provides a practical and theoretically grounded option for practitioners seeking to optimize tail latency in real-world LLM deployments.

1 Introduction

Prompt Caching is Essential. AI capabilities have exploded in recent years, and so has the demand. By December 2024, ChatGPT handled *1 billion* user messages every day with *300 million* weekly active users [OpenAI Newsroom, 2024]. To efficiently use scarce and costly GPU resources, prompt caching was proposed [Gim et al., 2024]: it caches the KV cache of existing queries, allowing a new query to skip some computation by reusing the KV cache it shares with existing queries [vLLM Team, 2025]. Prompt caching can reduce prefill computation thus Time to First Token (TTFT). This technique has been adopted by OpenAI and Anthropic, both reporting a significant amount (50-90%) of latency and cost reductions [OpenAI, 2024, Anthropic, 2024]. Despite the practical impact of prompt caching, little is known about how much existing caching policies—such as the Least Recently Used (LRU) policy—can be improved upon with respect to *key metrics* in LLM inference systems, which is the main motivation of this work.

Challenges of Optimizing Tail Latency. One of such key metrics is *tail latency*. In real-time user-facing applications, companies care about high-percentile response time, e.g., 95% of requests complete within 200 ms. In LLM conversation-based applications, users arrive to request services through an alternating sequence of prompts and responses that we call *turns*. Each prompt, along with

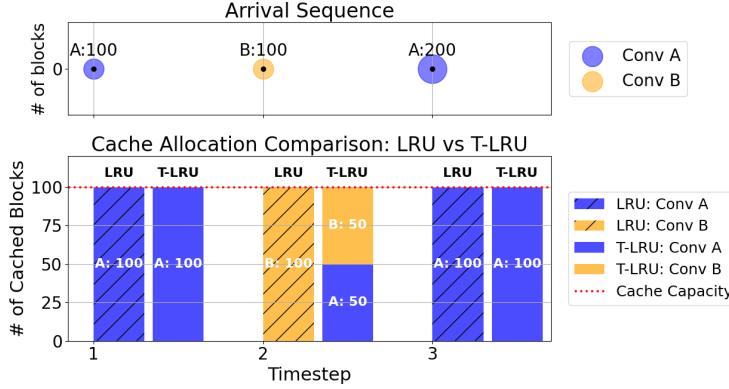


Figure 1: **Failure of LRU on Tail Latency.** Consider an example with two conversations (A and B) and a total of three turns (A has two turns; B has one). The top panel shows the total job size (conversation history plus user prompt) at each step. For simplicity, we assume the response length is zero and that each user prompt consists of 100 blocks. The bottom panel shows the number of cached blocks updated after each turn. The cache capacity is 100 blocks. We measure tail latency as the maximum processing time among the three requests (i.e., the 66.7%-th percentile). Under LRU, all of conversation A’s cache blocks are evicted immediately after step 2, resulting in a maximum of 200 uncached blocks at step 3. Therefore, the tail latency under LRU corresponds to the time required to process 200 blocks. In contrast, if at step 2 we partially evict A’s cache so that both A and B retain 50 cached tokens each, then regardless of whether the third request comes from A or B, the maximum number of uncached blocks is reduced to 150—an improvement of 25%. This improvement is what our proposed policy, T-LRU, is designed to achieve.

all previous conversation history, is treated as a job *request*. When the cache is full, the server must decide *which KV cache blocks to evict*, under four layers of uncertainty: 1) when new conversations are arriving; 2) the number of future turns of existing conversations; 3) the size of future user prompt and model response; and 4) the arrival order of turns from concurrent conversations competing for cache space. Here, KV cache blocks are the atomic cacheable units of tokens, e.g., a single block may consist of 128 tokens [OpenAI, 2024]. These intertwined dynamics make tail latency optimization in LLM inference uniquely challenging.

Existing Approaches. Classic caching/paging policies mostly focus on maximizing *cache hit rate*, not *tail latency*. Among these, the LRU policy is perhaps the most representative, evicting the cache item that was accessed least recently. LRU has been widely adopted in LLM inference systems, including vLLM [Kwon et al., 2023], SGLang [Zheng et al., 2024], and Mooncake [Qin et al., 2025]. However, LRU does not account for the fact that different blocks within a request may have varying effects on tail latency, thus leaving room for further optimization. See Figure 1 for an illustrative example.

Our Approach: Tail-Optimized LRU (T-LRU). In this work, we focus on optimizing the tail latency of TTFT (Time to First Token) through improved cache eviction policies. To this end, we introduce the metric *Tail Excess Latency* (TEL):

$$TEL = \sum_i \max\{TTFT(i) - \xi_s, 0\}, \quad (1)$$

where $TTFT(i)$ denotes the Time to First Token for the i -th request, and $\xi_s \geq 0$ is a user-specific latency threshold (e.g., $\xi_s = 200$ ms). TEL captures the total TTFT exceeding the threshold, serving as a practical proxy for tail latency while treating each request independently. See Section 2 for further discussion of this objective.

Empirically, TTFT is known to be approximately linear in the number of uncached tokens¹, i.e.,

$$TTFT(i) \approx \alpha \cdot b(i), \quad (2)$$

¹This approximation holds well when the uncached length is not excessively long and the prefill of the request is not batched with others. We use this approximation for analytical purposes, while the actual TTFT is measured in our experiments.

where α is a constant determined by the model and GPU configuration, and $b(i)$ is the number of uncached blocks for request i (see [Horton et al., 2024] and Figure 3). We use this approximation to motivate our policy design and simplify the analysis. By defining $\xi = \xi_s/\alpha$, TEL can be equivalently expressed as

$$\text{TEL} \approx \alpha \sum_i \max\{b(i) - \xi, 0\}. \quad (3)$$

Our intuition is the following: for a conversation turn with conversation history length L and whose next prompt is expected to add Q blocks, caching more than $L + Q - \xi$ blocks for this turn cannot improve TEL as the number of uncached blocks is already lower than ξ . Any blocks beyond this TEL-safe budget can therefore be evicted “for free”. See Figure 2 for illustration of this idea. Motivated by this observation, we propose Tail-Optimized LRU (T-LRU), a simple modification of LRU. Upon cache overflow, T-LRU works in two phases:

1. TEL-safe trimming: first evict blocks from conversations whose cache size exceeds the TEL-safe budget $L + Q - \xi$;
2. LRU-as-usual: If space is still needed, evict blocks using LRU.

In Figure 1, the performance of T-LRU with $\xi = 150$ is shown. In practice, the next-prompt length is unknown; T-LRU can use a constant \hat{Q} such as the empirical average for estimating the length of the next-prompt.² Implementation requires only one extra bookkeeping: mark TEL-safe blocks as “infinitely old,” after which any existing LRU engine can evict them as usual.

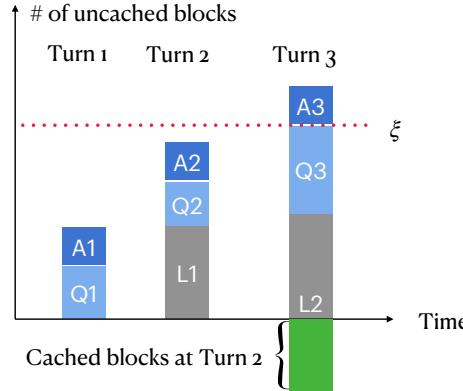


Figure 2: A three-turn conversation $\{(Q_1, A_1), (Q_2, A_2), (Q_3, A_3)\}$ is shown, L_i denotes conversation history (i.e., $L_1 = Q_1 + A_1$, $L_2 = L_1 + Q_2 + A_2$). Bars indicate the number of uncached blocks each turn incurs; the red dashed line marks the latency threshold ξ . Turn 1: projected load for next request is $L_1 + Q_2 < \xi$; no caching is needed. Turn 2: now $L_2 + Q_3 > \xi$, at least $L_2 + Q_3 - \xi$ blocks must be cached (green), but caching more won’t improve TEL further.

Theoretical Optimality. On the theoretical front, we establish the following results:

1. Belady’s algorithm [Belady, 1966], which evicts the block whose next use is furthest in the future, is a clairvoyant caching strategy known to achieve the *hindsight-optimal hit rate*. We show that combining *TEL-safe trimming* with Belady’s algorithm yields an policy being *hindsight optimal for TEL* (Theorem 1). This result justifies the use of T-LRU, as LRU is a widely used heuristic for Belady’s algorithm.
2. In the caching literature, the settings under which LRU is *online optimal* are not well understood. To address this, we introduce a novel stochastic model for multi-turn conversations that captures both uncertainty and temporal locality in LLM workloads. Within this framework, we prove that a generalized T-LRU is optimal (Theorem 2), incorporating the optimality of classical LRU (for average latency) and T-LRU as special cases. *To the best of*

²Furthermore, $Q - \xi$ can be combined and interpreted as one tunable parameter that determines how many blocks at the end of the conversation history are considered safe to evict.

our knowledge, this is the first result demonstrating the optimality of LRU under a natural, conversation-driven arrival model, and may be of broad interest to the caching community.

Strong Practical Performance. Finally, we evaluate the performance of T-LRU and LRU on real multi-turn chat traces from ShareGPT [Contributors, 2025] and WildChat [Zhao et al., 2024]. Our policy achieves up to a 23.9% reduction in P95 tail latency compared to LRU, and up to a 38.9% decrease in SLO violations relative to the strongest baseline. We also provide insights into selecting the latency threshold ξ (Section 5). These results highlight the practical advantages of T-LRU.

The rest of the paper is organized as follows. We discuss related work in Section A. In Section 2, we describe the problem setting and present the hindsight-optimal policy for TEL. Section 3 introduces the T-LRU policy and demonstrates that it is never worse than LRU for optimizing TEL. In Section 4, we prove the optimality of T-LRU for a class of stochastic models that capture conversational arrivals. Section 5 presents the experimental results. Finally, Section 6 discuss future directions.

2 Hindsight Optimal Policy

Deterministic Arrival Trace. To begin, we describe the prompt caching problem for LLM inference under a deterministic arrival trace to study the hindsight optimality, deferring the stochastic model to Section 4 for online optimality. Consider a discrete time horizon $t = 1, 2, \dots, T$ with N conversations over T steps. For each conversation $i \in [N]$, let $\mathcal{T}_i \subseteq [T]$ denote the set of time steps when conversation i issues a request. Assume \mathcal{T}_i is disjoint from each other. For each $t \in \mathcal{T}_i$, let $q_{i,t}$ and $a_{i,t}$ denote the lengths of the user prompt and model response for that turn, respectively, measured in blocks; otherwise, set $q_{i,t} = a_{i,t} = 0$ for $t \notin \mathcal{T}_i$.

Caching State. For $i \in [N]$ and $t \in [T]$, let $x_{i,t}$ denote the number of cached blocks for conversation i at time t , *before the request arrival*. The caching state $\{x_{i,t}\}$ must satisfy the following constraints.

First, the total number of cached blocks cannot exceed the cache capacity C at any time:

$$\sum_{i \in [N]} x_{i,t} \leq C, \quad \forall t \in [T] \quad (\text{capacity constraint}) \quad (4)$$

Second, for each conversation, the number of cached blocks cannot exceed the length of its conversation history up to time t :

$$x_{i,t+1} \leq \sum_{j=1}^t (q_{i,j} + a_{i,j}), \quad \forall i \in [N], t \in \mathcal{T}_i, t < T \quad (5)$$

Third, the cache allocation for a conversation can only increase when a request from that conversation arrives; otherwise, it can only decrease or stay the same:

$$x_{i,t+1} \leq x_{i,t}, \quad \forall i \in [N], t \notin \mathcal{T}_i, t < T \quad (\text{cache increases only on arrival}) \quad (6)$$

A caching policy must determine the caching state $\{x_{i,t}\}$ subject to these constraints. When a request arrives, the server can reuse any cached blocks in $\{x_{i,t}\}$, while any evicted blocks must be recomputed.³

For simplicity, we assume *optional caching*: a request is not required to be added to the cache after serving. Extending to *forced caching* is straightforward (by replacing the inequality in constraint (5) with equality); we defer this discussion to Appendix D.

Latency Objective: TEL. We aim to optimize the tail latency metric via the caching policy. Percentile-based tail latency is notoriously difficult to optimize directly, so we introduce the *Tail Excess Latency* (TEL) metric as a tractable surrogate:

$$\text{TEL} := \sum_{i \in [N], t \in \mathcal{T}_i} \max\{\text{TTFT}(i, t) - \xi_s, 0\},$$

³In practice, evicted blocks may be retrieved from other storage layers (e.g., CPU DRAM or SSD). Here, we focus on a single-layer cache and do not consider inter-layer transfers.

where $\text{TTFT}(i, t)$ denotes the Time to First Token for the request at time t of conversation i and ξ_s is a pre-defined threshold. TEL is analogous to Conditional Value at Risk (CVaR), but with an explicit, user-specified threshold [Bäuerle and Ott, 2011, Chow et al., 2015]; it also resembles SLO attainment metrics [Zhong et al., 2024], with the distinction that TEL penalizes larger violations proportionally more. Setting $\xi_s = 0$ recovers the average-latency objective. While our theoretical analysis centers on TEL, our experiments report conventional metrics (tail latency and SLO violation rate) for straightforward comparison in future studies.

As discussed in (2), to simplify the analysis and guide policy design, we assume TTFT grows linearly with the number of uncached blocks—an assumption empirically justified (see Figure 3) by the dominance of feed-forward linear layers during inference [Kamath et al., 2025, Zhu et al., 2024, Ye et al., 2025]. Thus,

$$\text{TEL} = \alpha \sum_{i \in [N], t \in \mathcal{T}_i} \max \left\{ \sum_{j=1}^t q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - x_{i,t} - \xi, 0 \right\}$$

where $\xi = \xi_s / \alpha$, and $\sum_{j=1}^t q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - x_{i,t}$ is the number of blocks that need to be computed the request of conversation i at time t (here, $\sum_{j=1}^{t-1} q_{i,j} + \sum_{j=1}^{t-1} a_{i,j}$ is the total number of blocks in the conversation history, and $q_{i,t}$ corresponds to the new request prompt at time t).

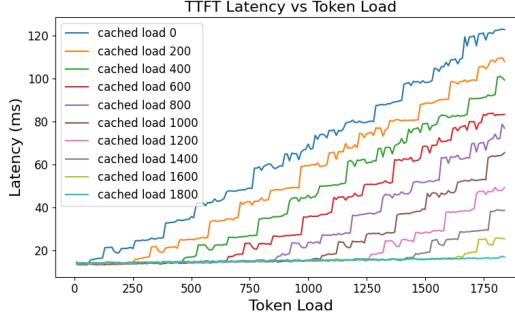


Figure 3: Experimental results demonstrate that TTFT increases approximately linearly with the number of uncached tokens. The plot shows TTFT latency as a function of the total prompt length (Token Load) and the size of the cached prefix (Cached Load). We conducted our experiments using Vicuna-7B with vLLM’s prefix caching enabled on a Colab A100 GPU.

Hindsight Optimal Policy for TEL With both the decision timeline and objective clarified, we next ask: *If the system knew the entire future arrival trace, what caching policy would minimize TEL?* Understanding this hindsight-optimal benchmark unveils what’s important to improve tail latency and thus guides our policy design. The hindsight optimal policy chooses cache variables $\{x_{i,t}\} \in \mathbb{N}$: the number of cache blocks conversation i can reuse at the beginning of step t , and slack variables $u_{i,t} \geq 0$ to minimize TEL:

$$\begin{aligned} & \min_{x_{i,t}, u_{i,t} \in \mathbb{N}} \sum_{i \in [N]} \sum_{t \in \mathcal{T}_i} u_{i,t} & (7) \\ & \text{s.t. (4), (5), (6)} \\ & u_{i,t} \geq \sum_{j=1}^t q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - x_{i,t} - \xi, \quad \forall i \in [N], t \in \mathcal{T}_i \quad (\text{slack variable}) \end{aligned}$$

Here $u_{i,t}$ describes the TEL objective as the $u_{i,t}$ equals $(\sum_{j=1}^t q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - x_{i,t} - \xi)^+$ in the optimal solution.

Theorem 1 (Hindsight-Optimal Policy Structure). *The hindsight-optimal policy that minimizes TEL restricts the number of KV-cache blocks allocated to each conversation within the TEL-safe budget, given by $(\sum_{j=1}^t q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - \xi)^+$. If the total allocation exceeds the cache capacity, the policy evicts cache blocks from conversations whose next requests are expected to arrive furthest in the future.*

See Appendix B for detailed proof. Setting $\xi = 0$ recovers average latency minimization, and in this case, the theorem implies the furthest-in-future eviction strategy is optimal—we recover the Belady optimal policy, which was shown to maximize cache hit rate in classical caching problems [Belady, 1966]. Therefore, the hindsight optimal policy for minimizing TEL can be characterized as a threshold-capped version of the Belady policy. We call this policy Tail-Optimized Belady.

3 Tail-Optimized LRU

Inspired by the Tail-Optimized Belady policy (Theorem 1), we propose Tail-Optimized LRU, an online policy that also maps into the future and caches *just enough* to prevent conversations’ next turns from affecting TEL. To adapt the hindsight optimal policy to online settings, we make two modifications: 1) replace the actual next-prompt length Q with an estimate \hat{Q} ; 2) use the least-recently-used eviction strategy rather than the furthest-in-future.

Pseudocode is given in 1. We believe Tail-Optimized LRU is a practical, low-friction upgrade for any LLM caching system that already relies on LRU: its no-worse-than-LRU guarantee (Theorem ??) eliminates adoption risk, while the lightweight modifications from standard LRU minimize adoption costs.

Algorithm 1: Tail-Optimized LRU Policy

Input: Number of conversations N , Timestamp of Last Turn $\{\tau_i\}$, Number of cached blocks $\{X_i\}$, conversation history lengths $\{L_i\}$, arriving conversation θ , arriving conversation length L'_θ
Parameters: Policy parameters: threshold ξ , next-turn length estimate $\{\hat{Q}_i\}$
Output: Updated cache sizes $\{X_i\}$
 $L_\theta \leftarrow L'_\theta$, $X_\theta \leftarrow L_\theta$, $\tau_\theta \leftarrow$ Current Timestamp // Update system state for arriving conversation
if $\sum_{i \in [N]} X_i > C$ **then**
 foreach $i \in [N]$ **do**
 if $X_i \geq L_i + \hat{Q}_i - \xi$ **then**
 $X_i \leftarrow X_i - 1$ // ‘Free Eviction’ under TEL objective
 if $\sum_{i \in [N]} X_i \leq C$ **then**
 return X
while $\sum_{i \in [N]} X_i > C$ **do**
 Find $j = \arg \min_{i \in [N]: X_i \geq 1} \tau_i$ // Evict using LRU
 $X_j \leftarrow X_j - 1$
return X

Lightweight Integration with Existing Caching Systems. Implementing Tail-Optimized LRU in a cache system based on LRU only requires an extra bookkeeping: mark blocks that can be evicted “for free” (i.e., blocks identified in the proactive trimming phase) as infinitely old. Then these blocks can be evicted seamlessly by any existing LRU engine in the usual way. This design is also compatible with paged KV cache technique which stores keys and values in non-contiguous memory space.

4 Optimality of Tail-Optimized LRU in a Stochastic Conversation Model

In this section, we prove the optimality of a generalized Tail-Optimized LRU policy under stochastically generated traces, which covers the optimality of T-LRU (Policy 1) and LRU as special cases.

Stochastic Conversation Model. Classic caching models fail to capture the nature of LLM workload: unlike traditional cache systems where objects have static sizes and independent access patterns, LLM workloads consist of multi-turn conversations that dynamically start, evolve, and terminate over time; see related work in Section A. To address this gap, we build a novel stochastic model that characterizes these unique features of LLM workloads.

To characterize multi-turn conversations, our model must address four fundamental questions: (1) when do new conversations start? (2) when will an active conversation generate its next prompt? (3) how many tokens appear in prompts and responses? (4) when do conversations terminate? Real

traces from Zhao et al. [2024] reveals strong *temporal locality* in multi-turn conversations: *the longer a user goes without sending their next prompt, the less likely they are to ever return*. This observation has a direct practical implication: least-recently-used conversations are also least-likely-to-return. Consequently, when cache capacity constraints force eviction decisions, prioritizing recently active conversations aligns with their probability of future requests. We capture this pattern by modeling the number of active conversations as a continuous-time birth-death process. Specifically,

- New conversations are “born” at rate $\lambda_{\text{conv}} > 0$, and each active conversation “dies” at rate $\mu > 0$. We index conversations by their arrival order, $i = 1, 2, \dots$
- While active, conversation i generates requests according to an independent Poisson process with rate $\bar{\lambda}_i > 0$.
- At each turn, a random prompt length Q is drawn from a known (possibly conversation-specific) distribution; the length of model responses A follows an arbitrary distribution that the decision-maker does not need to know.

This design has an appealing property: conversations with exponential “death clocks” naturally implement the temporal locality observed empirically. The longer a conversation stays quiet, the more likely it has terminated, making it a safe candidate for cache eviction. Building on this insight, we develop T-LRU, a policy that maintains LRU ordering while prioritizing conversations whose next request would breach a time-to-first-token (TTFT) latency threshold. In this section, we prove that a generalized version of T-LRU that minimizes expected total eviction loss (TEL) is optimal under our stochastic conversation model.

For simplification, we assume that KV caches cannot be reused across different conversations. This assumption is also motivated by security and privacy concerns: e.g., vLLM implement cache isolation to prevent timing-based inference attacks.⁴

Belief Markov Decision Process. The decision-maker only observes the turns as they arrive, but departures are never observed. Thus the problem is modeled as a partially observable Markov decision process (POMDP), where the optimal policy is defined for each possible belief state over the POMDP states. As the conversations arrive and depart independently, we can decompose the belief state to be the individual expected turn rates of each conversation.

Let $\pi_i(t)$ denote the decision-maker’s belief at time t that conversation i is still active, then its expected turn-arrival rate is $\pi_i(t) \cdot \bar{\lambda}_i$. The belief is updated using

$$\pi_i(t) = \exp(-\mu(t - \text{last turn time})),$$

as each conversation lasts for an exponential amount of time with mean μ .

Therefore, the system state at time t of the belief MDP is given by $(\boldsymbol{\lambda}(t), \mathbf{L}(t), \mathbf{X}(t))$, where $L_i(t)$ is the total length (blocks) of conversation i at time t and $X_i(t)$ is the number of KV cache blocks from conversation i , all prior to the arrival at time t . The decision-maker chooses \mathbf{X}' , which blocks to cache after serving each request, to minimize the Tail Excess Latency for M requests for arbitrary $M \in \mathbb{N}$. Appendix C details the finite-horizon Bellman equation.

Expected-Tail-Optimized LRU (ET-LRU). ET-LRU chooses the post-arrival cache allocation that minimizes the expected TEL at the next turn, using current beliefs of turn-arrival rates. Formally,

Definition 1 (Expected Tailed-Optimized LRU). *Let θ denote the index of the conversation arriving at time t with new prompt length Q and model response length A , $\mathbf{X}(t^-)$ denote the cache state before arrival, and $\boldsymbol{\lambda}(t^+), \mathbf{L}(t^+)$ denote the belief turn-arrival rates and conversation history lengths updated after service ($\lambda_\theta(t^+) = \bar{\lambda}_\theta, L_\theta(t^+) = L_\theta(t^-) + Q + A$). ET-LRU chooses*

$$\mathbf{X}^{\text{ETLRU}} \in \arg \min \sum_i \lambda_i(t^+) \cdot \mathbb{E}[(L_i(t^+) + Q_i - Y_i - \xi)^+] \quad (8)$$

$$\text{s.t. } Y_\theta \leq L_\theta(t^+), \quad (\text{optional caching}) \quad (9)$$

$$Y_i \leq X_i(t^-), \forall i \neq \theta, \quad (\text{cannot conjure caches}) \quad (10)$$

$$\sum_i Y_i \leq C. \quad (\text{capacity constraint})$$

⁴https://docs.vllm.ai/en/stable/design/v1/prefix_caching.html

where the expectation is taken over Q_i , the random user prompt length for conversation i at its next arrival.

The probability that conversation i generates the next request is proportional to its belief turn-arrival rates $\lambda_i(t^+)$, thus $\lambda_i(t^+) \cdot \mathbb{E}[(L_i(t^+) + Q_i - X'_i - \xi)^+]$ is the expected TEL contributed by conversation i . Here constraint (9) implies that the decision-maker can cache at most the total conversation history of conversation θ just served; constraint (10) says a conversation’s cache allocation can grow only when it arrives.

Theorem 2 (Optimality of Expected-Tail-Optimized LRU). *Expected-Tail-Optimized LRU (Definition 1) is an optimal online caching policy for minimizing Tail Excess Latency under the stochastic conversation model above.*

The proof is by induction on the number of turns and comparing the value-to-go functions under our policy and another policy that evicts differently. Here, the least-recently-used time reflects the expected turn-arrival rate and thus can approximate furthest-in-future. Theorem 2 has three implications:

- LRU is optimal for average latency. Setting $\xi = 0$ and assuming homogeneous turn-arrival rates across conversations, the objective in optimization problem (8) reduces to $\max \sum_i \exp(-\mu(t - \text{last turn time})) Y_i$, thus ET-LRU reduces to LRU. Therefore, Theorem 2 establishes the optimality of LRU for minimizing average latency under our stochastic arrival model. To the best of our knowledge, such a result had not previously been established.
- Optimality of Tail-Optimized LRU: if the user prompt length is deterministic, Expected-Tail-Optimized LRU is reduced to a deterministic version as stated in Policy 1 (with estimate \hat{Q} replaced by deterministic Q). Theorem 2 thus establishes the optimality of Tail-Optimized LRU in this model.
- When $Q = 0$ after the first turn and responses also have zero length, our model reduces to classic caching with unit page size. In this case, the optimal policy “evicts the block whose conversation is least likely to return”, generalizing least-recently-used.

Therefore, Expected-Tail-Optimized LRU serves as a common backbone across three classic caching regimes, providing theoretical justification for adopting LRU and Tail-Optimized LRU for LLM inference workload.

5 Experiments

5.1 Datasets and Metrics

We evaluate our caching policies on two conversational datasets: WildChat and ShareGPT (Figure 4 and Figure 5). For each dataset, we select conversations based on their arrival timestamps and extract the first 1000–2000 turns across these conversations. Specifically, we sample conversations in chronological order (by first-turn timestamp), split each conversation into individual turns, and simulate their arrivals following the observed timestamps in the trace.

We measure latency metrics (median, P90, P95, P99) and service-level objective (SLO) attainment under different caching policies. Our default experimental configuration uses Vicuna-7B served on a single A100 GPU via vLLM with tensor parallelism disabled ($TP = 1$) and without mixed batching. We present the results for WildChat below. Due to space constraints, Results for ShareGPT, which exhibit qualitatively similar patterns, are provided in Appendix G.

5.2 Tail Latency Reduction

We measure the tail latency reduction of our policy against LRU and *Threshold LRU*—caches *only if* the conversation length exceeds a fixed threshold, a configuration for LRU we observed in practice. We fix the input parameter for T-LRU, next-prompt length \hat{Q} , to be the average prompt length (200 for WildChat, 150 for ShareGPT), and use 1024 as the threshold for Threshold-LRU following the one used by OpenAI.⁵

⁵<https://platform.openai.com/docs/guides/prompt-caching>

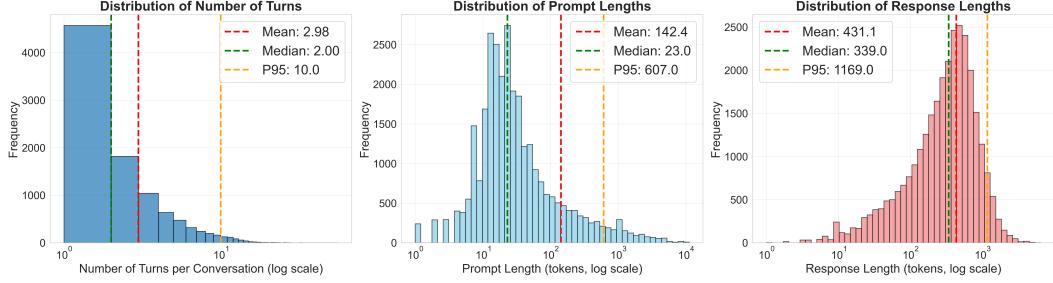


Figure 4: Distributions of turns and tokens of WildChat [Zhao et al., 2024] datasets (sampled 10,000 conversations).

In Tables 1–2, we report the relative latency improvements of T-LRU over both LRU and Threshold-LRU across various cache capacities C and tail-latency thresholds ξ_s .⁶ We observe that T-LRU reduces P90 tail TTFT by up to 27.5% and P95 tail TTFT by up to 23.9% compared to LRU, and achieves comparable improvements over Threshold-LRU.

Sensitivity to latency threshold ξ_s . The benefit of T-LRU peaks when latency threshold ξ_s is calibrated to match the target tail percentile of interest. *For example, with capacity $C = 1000$ under LRU, medium, P90, P95, and P99 tail latencies are roughly 40 ms, 240 ms, 326 ms, and 505 ms. Setting $\xi_s = 200$ ms yields the largest improvement in P90 tail latency; setting $\xi_s = 300$ ms yields the largest improvement in P95 tail latency. A high value, e.g. $\xi_s = 500$ ms, relaxes protection for moderate tails and only provides protection for extreme tails —up to 3% of increase in P99 tail latency.*⁷

Operationally, the latency threshold ξ_s can be tuned either based on fixed service level objective (e.g., a target TTFT), or adaptively based on the observed tail latency of served turns. For example, the decision-maker can periodically update ξ_s to match the desired tail latency TTFT observed over recent turns. Raising ξ_s makes TEL-safe trimming more aggressive, and could potentially increase average latency as the policy allows more turns to incur latency up to ξ_s , reflecting the classic trade-off between average and tail performance.

Comparing Threshold-LRU and T-LRU. Threshold-LRU is a straightforward patch for LRU’s blindness to conversation length. In industry systems such as at OpenAI, prompt caching is enabled only when the running conversation history exceeds a fixed length. While simple to implement, this policy makes a binary, all-or-nothing decision for each conversation: either cache the entire conversation history or cache nothing at all. In contrast, T-LRU makes adaptive, fine-grained caching decisions. Rather than caching entire conversations indiscriminately, T-LRU caches just enough tokens from each conversation to ensure its next request will not breach the latency threshold. This fundamental difference persists regardless of prompt length. Even with zero-length prompts, Threshold-LRU would apply its binary rule based on history length, while T-LRU would adaptively cache the amount that is just right so not impacting the tail.

Table 1: Relative latency improvement of T-LRU over LRU with various ξ_s

Capacity	$\xi_s = 50\text{ms}$		$\xi_s = 100\text{ms}$		$\xi_s = 150\text{ms}$		$\xi_s = 200\text{ms}$		$\xi_s = 500\text{ms}$		
	p90	p95	p90	p95	p90	p95	p90	p95	p90	p95	p99
1000	4.0%	0.0%	4.5%	1.0%	5.3%	1.5%	7.3%	2.0%	-1.4%	-1.3%	3.2%
2000	1.2%	0.6%	3.4%	0.6%	5.1%	3.4%	9.2%	4.4%	-4.9%	-2.8%	3.3%
4000	1.7%	0.7%	4.1%	2.8%	10.5%	4.2%	13.3%	10.8%	-7.5%	-3.4%	3.4%
6000	5.0%	2.1%	10.6%	4.0%	16.0%	11.4%	14.3%	15.4%	-8.7%	-4.2%	3.4%
8000	2.2%	0.7%	10.5%	8.5%	23.0%	15.8%	8.8%	19.5%	-13.7%	-3.5%	1.4%
10000	3.6%	7.4%	20.0%	15.7%	27.5%	20.1%	6.9%	23.9%	-13.3%	-3.5%	-0.0%

⁶ $C = 10,000$ corresponds to approximately 4.8 GB for Vicuna-7B with Float16, see Appendix F.

⁷We set $\xi = \xi_s/c$ using a simple linear fit between awaited prefill tokens and prefill time, which is easy to implement in practice.

Table 2: Relative latency improvement of T-LRU over Threshold-LRU with various ξ_s

Capacity	$\xi_s = 50\text{ms}$		$\xi_s = 100\text{ms}$		$\xi_s = 150\text{ms}$		$\xi_s = 200\text{ms}$		$\xi_s = 500\text{ms}$		
	p90	p95	p90	p95	p90	p95	p90	p95	p90	p95	p99
1000	1.5%	0.0%	2.0%	1.0%	2.9%	1.5%	4.8%	2.0%	-4.0%	-1.3%	3.2%
2000	0.4%	0.6%	2.7%	0.6%	4.4%	3.4%	8.6%	4.4%	-5.7%	-2.8%	3.3%
4000	0.3%	0.0%	2.7%	2.1%	9.3%	3.5%	12.0%	10.2%	-9.0%	-4.2%	3.4%
6000	4.7%	1.2%	10.4%	3.1%	15.7%	10.6%	14.1%	14.6%	-9.1%	-5.2%	3.4%
8000	1.1%	0.7%	9.5%	8.5%	22.2%	15.8%	7.8%	19.5%	-14.9%	-3.5%	1.4%
10000	2.4%	6.1%	19.0%	14.6%	26.6%	19.0%	5.7%	22.8%	-14.7%	-5.0%	-0.0%

5.3 SLO Violation Reduction

We now measure the *count* of requests that a service-level objective, another objective similar to TEL—the *amount* of latency beyond a threshold. Table 3 reports the improvement on SLO attainment ratio (relative drop in requests whose TTFT exceeds 200 ms). With the latency threshold ξ_s set to 200 ms SLO, compared to LRU, T-LRU reduces between 8.8% (small cache capacity) and 40.7% (large cache capacity) of violations.

Sensitivity to latency threshold ξ_s . When ξ_s is much lower than the SLO ($\xi_s = 50$ or 100 ms), the improvement is modest because both baselines already satisfy most requests; when ξ_s is far higher ($\xi_s = 500$ ms), T-LRU focuses on larger tails and can allow for a few extra 200 ms violations.

These results echo the design goal: TEL minimization penalizes *how much* a request overshoots ξ_s , yet the same trimming logic also cuts the *number* of SLO violations whenever ξ_s aligns or is slightly below the target latency budget.

Table 3: Relative improvement of T-LRU: % reduction in requests with latency $> 200\text{ms}$

Capacity	$\xi_s = 50\text{ms}$		$\xi_s = 100\text{ms}$		$\xi_s = 150\text{ms}$		$\xi_s = 200\text{ms}$		$\xi_s = 500\text{ms}$	
	LRU	Thre-LRU	LRU	Thre-LRU	LRU	Thre-LRU	LRU	Thre-LRU	LRU	Thre-LRU
1000	1.2%	-1.2%	4.1%	1.8%	6.5%	4.2%	8.8%	6.6%	-1.8%	-4.2%
2000	2.4%	1.2%	6.1%	4.9%	9.1%	8.0%	13.3%	12.3%	-4.8%	-6.1%
4000	3.9%	1.3%	7.1%	4.7%	14.3%	12.0%	22.1%	20.0%	-12.3%	-15.3%
6000	2.0%	0.7%	12.2%	11.0%	20.9%	19.9%	30.4%	29.5%	-12.2%	-13.7%
8000	4.3%	2.2%	16.4%	14.6%	29.3%	27.7%	33.6%	32.1%	-19.3%	-21.9%
10000	7.4%	4.6%	25.9%	23.7%	31.9%	29.8%	40.7%	38.9%	-19.3%	-22.9%

6 Future Directions and Conclusions

We propose Tail-Optimized LRU (T-LRU), a simple yet effective modification to the Least Recently Used caching policy that significantly improves tail latency in multi-turn conversational LLM serving. By adaptively caching just enough tokens to keep each conversation’s next request below a latency threshold—rather than making binary all-or-nothing caching decisions, T-LRU achieves substantial tail latency reductions (e.g., 20–30% improvement in P95 TTFT) with modest impact on median performance.

T-LRU makes a deliberate design choice: sacrifice tens of milliseconds at the median to eliminate hundreds of milliseconds at the tail. This trade-off aligns well with strict SLO requirements in production systems, though exploring multi-objective policies that balance tail and median latency differently remains an interesting direction for future work.

Our focus is on a single storage layer, but real systems increasingly adopt hierarchical memory architectures. Extending T-LRU to multi-tier KV caching systems—where evicted blocks migrate to slower storage (CPU memory, SSD) rather than being discarded—could unlock further performance gains. Recent work [Qin et al., 2025] has begun exploring such architectures, and understanding optimal promotion/demotion policies across cache tiers represents an important open problem. Another compelling direction is joint optimization of caching and load balancing. Recent work [Srivatsa et al., 2024] has initiated exploration of how caching decisions interact with request routing in distributed LLM serving. Analyzing these problems through a queueing-theoretic lens and considering both cache locality and load distribution could systematically characterize fundamental trade-offs and guide practical system design.

References

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 117–134, 2024.

Anthropic. Prompt caching, 2024. URL <https://docs.anthropic.com/en/docs/build-with-claude/prompt-caching>.

Nicole Bäuerle and Jonathan Ott. Markov decision processes with average-value-at-risk criteria. *Mathematical Methods of Operations Research*, 74:361–379, 2011.

Laszlo A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal*, 5(2):78–101, 1966.

Daniel S Berger, Ramesh K Sitaraman, and Mor Harchol-Balter. {AdaptSize}: Orchestrating the hot object memory cache in a content delivery network. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 483–498, 2017.

Daniel S Berger, Benjamin Berg, Timothy Zhu, Siddhartha Sen, and Mor Harchol-Balter. {RobinHood}: Tail latency aware caching—dynamic reallocation from {Cache-Rich} to {Cache-Poor}. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 195–212, 2018.

BioNumbers Database. Average duration of a single eye blink. BioNumbers Database, 2024. URL <https://bionumbers.hms.harvard.edu/bionumber.aspx?id=100706>. BNID 100706.

Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. cambridge university press, 2005.

Allan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. Competitive paging with locality of reference. *Journal of Computer and System Sciences*, 50(2):244–258, 1995.

Hao Che, Ye Tung, and Zhijun Wang. Hierarchical web caching systems: Modeling, design and experimental results. *IEEE journal on Selected Areas in Communications*, 20(7):1305–1314, 2002.

Jiayi Chen, Nihal Sharma, Tarannum Khan, Shu Liu, Brian Chang, Aditya Akella, Sanjay Shakkottai, and Ramesh K Sitaraman. Darwin: Flexible learning-based cdn caching. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 981–999, 2023.

Yinlam Chow, Aviv Tamar, Shie Mannor, and Marco Pavone. Risk-sensitive and robust decision-making: a cvar optimization approach. *Advances in neural information processing systems*, 28, 2015.

Marek Chrobak and John Noga. Lru is better than fifo. *Algorithmica*, 23:180–185, 1999.

Edward Grady Coffman and Peter J Denning. *Operating systems theory*, volume 973. prentice-Hall Englewood Cliffs, NJ, 1973.

ShareGPT Contributors. Sharegpt: A dataset of multi-turn chat interactions with large language models. <https://huggingface.co/datasets/RyokoAI/ShareGPT52K>, 2025.

Asit Dan and Don Towsley. An approximate analysis of the lru and fifo buffer replacement schemes. In *Proceedings of the 1990 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pages 143–152, 1990.

Amos Fiat, Richard M Karp, Michael Luby, Lyle A McGeoch, Daniel D Sleator, and Neal E Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338, 2024.

Maxwell Horton, Qingqing Cao, Chenfan Sun, Yanzi Jin, Sachin Mehta, Mohammad Rastegari, and Moin Nabi. Kv prediction for improved time to first token. *arXiv preprint arXiv:2410.08391*, 2024.

Raj Jain. Characteristics of destination address locality in computer networks: A comparison of caching schemes. *Computer networks and ISDN systems*, 18(4):243–254, 1990.

Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *arXiv preprint arXiv:2404.12457*, 2024.

Yunho Jin, Chun-Feng Wu, David Brooks, and Gu-Yeon Wei. s^3 : Increasing gpu utilization during generative inference for higher throughput. *Advances in Neural Information Processing Systems*, 36:18015–18027, 2023.

Aditya K Kamath, Ramya Prabhu, Jayashree Mohan, Simon Peter, Ramachandran Ramjee, and Ashish Panwar. Pod-attention: Unlocking full prefill-decode overlap for faster llm inference. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 897–912, 2025.

WC King. Analysis of paging algorithms. In *Proc. IFIP 1971 Congress, Ljubljana*, pages 485–490. North-Holland, 1972.

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

Emilio Leonardi and Giovanni Luca Torrisi. Least recently used caches under the shot noise model. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 2281–2289. IEEE, 2015.

Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *Journal of the ACM (JACM)*, 68(4):1–25, 2021.

Naram Mhaisen, Abhishek Sinha, Georgios Paschos, and George Iosifidis. Optimistic no-regret algorithms for discrete caching. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 6(3):1–28, 2022.

Michael Mitzenmacher and Rana Shahout. Queueing, predictions, and large language models: Challenges and open problems. *Stochastic Systems*, 15(3):195–219, 2025.

OpenAI. Prompt caching guide, 2024. URL <https://platform.openai.com/docs/guides/prompt-caching>. Developer documentation.

OpenAI Newsroom. 300m weekly chatgpt users and 1b daily messages, December 2024. URL <https://x.com/OpenAINewsroom/status/1864373399218475440>. Tweet.

Vidyadhar Phalke and Bhaskarpillai Gopinath. An inter-reference gap model for temporal locality in program behavior. *ACM SIGMETRICS Performance Evaluation Review*, 23(1):291–300, 1995.

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)*, pages 155–170. USENIX Association, 2025.

Daniel D Sleator and Robert E Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. Preble: Efficient distributed prompt scheduling for llm serving. 2024.

vLLM Team. Automatic prefix caching, 2025. URL https://docs.vllm.ai/en/stable/automatic_prefix_caching/details.html. Project documentation.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. *arXiv preprint arXiv:2501.01005*, 2025.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. Wildchat: 1m chatgpt interaction logs in the wild. *arXiv preprint arXiv:2405.01470*, 2024.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in Neural Information Processing Systems*, 37:62557–62583, 2024.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. {DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 193–210, 2024.

Banghua Zhu, Ying Sheng, Lianmin Zheng, Clark Barrett, Michael I Jordan, and Jiantao Jiao. On optimal caching and model multiplexing for large model inference. *arXiv preprint arXiv:2306.02003*, 2023.

Kan Zhu, Yilong Zhao, Liangyu Zhao, Gefei Zuo, Yile Gu, Dedong Xie, Yufei Gao, Qinyu Xu, Tian Tang, Zihao Ye, et al. Nanoflow: Towards optimal large language model serving throughput. *arXiv preprint arXiv:2408.12757*, 2024.

Timothy Zhu, Daniel S Berger, and Mor Harchol-Balter. Snc-meister: Admitting more tenants with tail latency slsos. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pages 374–387, 2016.

NeurIPS Paper Checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

Answer: [\[Yes\]](#)

Justification: Justification: Our abstract and introduction list three core claims: 1) a new cache eviction rule, Tail-Optimized LRU; (ii) provable optimality under a stochastic conversation model; and (iii) empirical tail-latency gains on real LLM workloads. Each claim is backed by a theorem or experiment.

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [\[Yes\]](#)

Justification: In the last section, we discuss future directions that could relax our the single-tier cache assumption and single-server assumption.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.

- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory assumptions and proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [\[Yes\]](#)

Justification: Complete proofs are in Appendix with cross-references and we provide intuition and proof sketch in the main paper.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. Experimental result reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [\[Yes\]](#)

Justification: We perform experiments on public datasets and provide all codes in the submission.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
 - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
 - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

- (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
- (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: **[Yes]**

Justification: We would submit and release the code.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental setting/details

Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: **[Yes]**

Justification: We provide details of parameters used in the experiment for our policy, such as cache capacity and latency threshold.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment statistical significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: **[No]**

Justification: Our experiments are used to simulate performances of proposed and baseline policies that are deterministic on fixed job request traces, thus we do not have error bars.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments compute resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [\[Yes\]](#)

Justification: We used A100 GPU to estimate the default latency function.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code of ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines>?

Answer: [\[Yes\]](#)

Justification: We checked the NeurIPS Code of Ethics; no personal data or disallowed content is involved

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. Broader impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: We discussed the trade-off between tail latency versus average latency, with the former speaking to fairness.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: Our work is about a caching policy, not a pretrained model or scraped dataset.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We have properly credited data used in the paper.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. New assets

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification: No new dataset or model checkpoint is introduced.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. Crowdsourcing and research with human subjects

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: We did not perform any human-subject data collection.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. Institutional review board (IRB) approvals or equivalent for research with human subjects

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: Our paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. Declaration of LLM usage

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: LLM is used only for writing, editing and facilitating data analyses and experimenting.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (<https://neurips.cc/Conferences/2025/LLM>) for what should or should not be described.

Appendix

A Related Work

Classic Caching and Paging. The caching problem has been extensively studied from a competitive analysis perspective, with binary cache hit/miss metrics. Foundational results include the hindsight-optimal Belady’s algorithm [Belady, 1966], optimal deterministic online policies [Sleator and Tarjan, 1985], and randomized algorithms [Fiat et al., 1991]. Beyond worst-case competitive ratios, many works model structured request processes to capture temporal locality observed in practice, including access graph models [Borodin et al., 1995, Chrobak and Noga, 1999], independent reference models [Coffman and Denning, 1973, King, 1972, Dan and Towsley, 1990, Che et al., 2002], Shot Noise models [Leonardi and Torrisi, 2015], LRU Stack and Working Set models [Jain, 1990], and Inter-Reference Gap models [Phalke and Gopinath, 1995]. For broader overviews, see Borodin and El-Yaniv [2005]; for caching with predictions, see Lykouris and Vassilvitskii [2021], Mhaisen et al. [2022].

Alternative Metrics. Beyond binary hit/miss metrics, several systems optimize continuous objectives. Darwin [Chen et al., 2023] balances hit rate and disk writes in CDN caching. RobinHood [Berger et al., 2018] and SNC-Meister [Zhu et al., 2016] target tail latency in web services and multi-tenant systems, respectively. AdaptSize [Berger et al., 2017] handles variable-size objects in content delivery networks. However, these works focus on traditional web caching rather than the unique characteristics of LLM inference: multi-turn conversations with growing context windows and strict real-time latency requirements.

Prompt Caching for LLM Inference. Prompt caching reuses precomputed KV states from conversation history to reduce prefill computation. Recent systems explore various aspects: prefix caching [Kwon et al., 2023], attention reuse [Gim et al., 2024], RAG optimization [Jin et al., 2024], structured generation [Zheng et al., 2024], load balancing [Srivatsa et al., 2024], hierarchical storage [Qin et al., 2025], and optimal model multiplexing [Zhu et al., 2023]. While these works demonstrate the value of prompt caching, they primarily optimize throughput or average latency rather than tail latency.

Recent work on LLM serving has begun addressing tail latency through architectural innovations. Sarathi-Serve [Agrawal et al., 2024] uses chunked prefill and stall-free scheduling, while Dist-Serve [Zhong et al., 2024] disaggregates prefill and decoding for goodput optimization. These systems target tail latency but do not use caching as the primary optimization lever.

Our work. To sum up, existing works have studied continuous metrics other than binary cache hit/miss and variable-size objects, but we are the first to use prompt caching as the primary lever to optimise TTFT tail latency, and the first to provide theoretical guarantees for tail-excess latency. Our work bridges the gap between classical caching theory, modern LLM serving systems, and tail latency optimization.

B Proof of Theorem 1 Hindsight Optimal Policy for TEL

Proof. The proof has two steps: 1) show that optimizing the original TEL problem (7) is equivalent to a new problem (11) of maximizing the total number of reused cached blocks, subject to an additional “TEL-safe” capping constraint; 2) show that this new problem is a classic offline caching problem whose optimal solution is to cap the KV cache size using TEL-safe budget and then evict using furthest-in-future policy.

Step one: Equivalence of Optimization Problems Fix a feasible \mathbf{x} to (7), the optimal \mathbf{u} is given by

$$u_{i,t} = \left(\sum_{j=1}^t q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - x_{i,t} - \xi \right)^+,$$

as otherwise we have $u_{i,t}$ infeasible or can be improved. Thus we can focus on cache decisions \mathbf{x} .

We claim the optimal solution to the optimization problem (7) must satisfy the TEL-safe budget $x_{i,t} \leq \left(\sum_{j=1}^t q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - \xi \right)^+$ for every $i \in [N], t \in \mathcal{T}_i$. Suppose not, i.e., the optimal x' to (7) satisfies $x'_{i,t} > \left(\sum_{j=1}^t q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - \xi \right)^+$ for some $i \in [N], t \in \mathcal{T}_i$. Then we have

$$\sum_{j=1}^t q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - x'_{i,t} - \xi < 0, u'_{i,t} = 0.$$

In this case, setting $x_{i,t} = \left(\sum_{j=1}^t q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - \xi \right)^+$ will not affect the value of $u'_{i,t}$, while releasing cache capacity that can be directed to other conversations, which contradicts the optimality of x' .

As the optimal solution must respect the TEL-safe budget, we can simplify the objective:

$$\begin{aligned} \sum_{i \in [N]} \sum_{t \in \mathcal{T}_i} u_{i,t} &= \sum_{i \in [N]} \sum_{t \in \mathcal{T}_i} \left(\sum_{j=1}^t q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - x_{i,t} - \xi \right)^+ \\ &= \sum_{i \in [N]} \sum_{t \in \mathcal{T}_i} \mathbb{1} \left\{ \sum_{j=1}^t q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - \xi \geq 0 \right\} \cdot \left(\sum_{j=1}^t q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - x_{i,t} - \xi \right) \\ &\quad + \sum_{i \in [N]} \sum_{t \in \mathcal{T}_i} \mathbb{1} \left\{ \sum_{j=1}^t q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - \xi < 0 \right\} \cdot 0 \\ &= \sum_{i \in [N]} \sum_{t \in \mathcal{T}_i} \mathbb{1} \left\{ \sum_{j=1}^t q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - \xi \geq 0 \right\} \left(\sum_{j=1}^t q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - \xi \right) - \sum_{i \in [N]} \sum_{t \in \mathcal{T}_i} x_{i,t} \end{aligned}$$

where the last equality holds as $x_{i,t} = 0$ for the requests with $\sum_{j=1}^t q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - \xi < 0$. Thus minimizing $\sum_{i \in [N]} \sum_{t \in \mathcal{T}_i} u_{i,t}$ is equivalent to maximizing $\sum_{i \in [N]} \sum_{t \in \mathcal{T}_i} x_{i,t}$ as the first term of the right-hand-side of the last equality above is a constant.

Therefore, we can rewrite (7) as follows:

$$\begin{aligned} \max_{x_{i,t} \in \mathbb{N}} \sum_{i \in [N]} \sum_{t \in \mathcal{T}_i} x_{i,t} \quad (11) \\ \text{s.t. (4), (5), (6)} \end{aligned}$$

$$x_{i,t} \leq \left(\sum_{j=1}^t q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - \xi \right)^+, \forall i \in [N], t \in \mathcal{T}_i; \quad (12)$$

Step two. We can interpret each block as a unit-size item that, once arrived, is requested again at every subsequent turn of the same conversation (prefix reuse). At time $t \in \mathcal{T}_i$, the number of cache hits for that request of conversation i equals exactly $x_{i,t}$. Hence the objective of (11) is the total number of cache hits over the horizon.

Because the cache content may change only at request times and all items have unit size, (11) is a paging instance with unit pages and clairvoyant knowledge, except that each request carries a per-request cap on the number of allowable hits, which we can enforce by immediately discarding any excess above upon the turn's completion.

In paging with unit pages, the offline optimal policy that maximizes total hits (equivalently minimizes misses) is Bélády's furthest-in-future rule: whenever eviction is needed, evict the item whose next request is farthest in the future. Here, all blocks of conversation i share the same next request time—the next arrival of conversation i —so the rule specializes to: evict from the conversation whose next arrival is farthest in the future.

□

C Proof of Theorem 2

Proof. Given system state λ, L, X , let θ denote the index of the conversation that is the k^{th} arrival with user prompt length Q and model response length A . The finite-horizon value-to-go function is

$$V_k(\lambda, L, X, \theta, Q, A) = \underbrace{(L_\theta + Q - X_\theta - \xi)^+}_{\text{number of uncached blocks above threshold}} + \min_{X' \in \mathcal{X}(X, \theta, L_\theta + Q + A)} \mathbb{E}_{\tau, \theta', Q', A'} \left[V_{k+1} \left(\underbrace{\Phi(\lambda, \theta, \tau)}_{\text{belief arrival state transition}}, \underbrace{\Psi(L, \theta, Q + A)}_{\text{conversation length transition}}, X', \theta', Q', A' \right) \right]$$

with $V_{M+1}(\cdot) = 0$, where the feasible caching decision space is

$$\mathcal{X}(X, \theta, L) = \{Y \in \mathbb{N}^{\dim(X, \theta)} : \sum_i Y_i \leq C, 0 \leq Y_\theta \leq L, 0 \leq Y_i \leq X_i, i \neq \theta\}$$

with $\dim(X, \theta) = \dim(X) + \mathbb{1}\{\theta > \dim(X)\}$, here $\dim(X)$ denotes the dimension of vector X , and the dimension expands when a new conversation arrives; $\Phi(\lambda, \theta, \tau)$ update the belief turn rate of conversation θ to $\tilde{\lambda}_i$, then discount belief turn rates of all conversations by $\exp(-\mu\tau)$; $\Psi(L, \theta, Q + A)$ updates the conversation length vector. Specifically, we increase the dimension of L if necessary (i.e., when θ represents a new conversation), and add $Q + A$ to its θ^{th} entry.

Let θ denote the index of the conversation that is the k^{th} arrival, with user prompt length Q and model response length A . Define the belief arrival rate vector as λ , the conversation length vector as L , and the cached token length vector as X , the cost-to-go function is given by:

$$V_k(\lambda, L, X, \theta, Q, A) = (L_\theta + Q - X_\theta - \xi)^+ + \min_{X' \in \mathcal{X}(X, \theta, L_\theta + Q + A)} \mathbb{E}_{\tau, \theta', Q', A'} [V_{k+1}(\Phi(\lambda, \theta, \tau), \Psi(L, \theta, Q + A), X', \theta', Q', A')]$$

with $V_{M+1}(\cdot) = 0$. Let's rewrite $\Phi(\lambda, \theta, \tau) = \Phi(\Gamma(\lambda, \theta), \tau)$ with

- $\Gamma(\lambda, \theta)$ updates the return rate vector upon the arrival of conversation θ . Specifically, this operator changes the belief arrival rate of conversation θ to $\tilde{\lambda}_i$.
- $\Phi(\lambda, \tau)$ discount all return rates by $\exp(-\mu\tau)$.

The proof is by using induction and argue that if we choose a different caching state than X^{TLRU} , the cost will be higher. At last arrival M , $V_M(\lambda, L, X, \theta, Q_\theta, A_\theta) = (L_\theta + Q_\theta - X_\theta - \xi)^+$ and any caching policy is optimal.

Suppose this holds for the $k^{th} + 1$ arrival. We proceed to show that the result holds for the k^{th} arrival. To simplify the notation, we define

$$J_k(\lambda, L, X) = \mathbb{E}_{\tau, \theta, Q_\theta, A_\theta} [V_k(\Phi(\lambda, \tau), L, X, \theta, Q_\theta, A_\theta)], \tilde{\lambda} = \Gamma(\lambda, \theta), \tilde{L} = \Psi(L, \theta, Q_\theta + A_\theta),$$

Then we need to prove

$$J_{k+1}(\tilde{\lambda}, \tilde{L}, X^{\text{ETLRU}}) \leq J_{k+1}(\tilde{\lambda}, \tilde{L}, X')$$

To see this, by definition of state transition, suppose the inter-arrival time is τ , the discounted arrival rates are

$$\tilde{\lambda}_i \cdot \exp(-\mu\tau) \text{ with } \tilde{\lambda}_\theta = \bar{\lambda}_\theta.$$

From these expressions, we can conclude that regardless of value of τ , the return rates at next arrival maintain the same relative ordering as in λ for conversations. Note that due to heterogeneous turn rates across conversations, conversation θ that arrived at the k^{th} arrival may not have the highest turn rate. Without loss of generality, let's assume $\tau = 0$ and let $\tilde{p}_j = \tilde{\lambda}_j / (\lambda_{\text{conv}} + \sum_i \tilde{\lambda}_i)$ denote the probability that conversation j returns at the next arrival, and $\tilde{p}_{\dim(X(1))+1} = \lambda_{\text{conv}} / (\lambda_{\text{conv}} + \sum_i \tilde{\lambda}_i)$ denote the probability that a new conversation starts at the next arrival.

We proceed to show this holds for any number of tokens evicted by treating each token eviction separately. Among all conversations that have arrived so far and have at least one cached token, list their indices as $i(1), i(2), \dots$ in ascending order of the ranking criterion score

$$\tilde{\lambda}_i \mathbb{P}(\tilde{L}_i + Q_i - \xi \geq X_i),$$

Insert conversation θ that just arrives into this ordered list according to its own score

$$\bar{\lambda}_\theta \mathbb{P}(Q_\theta - \xi \geq 0)$$

Let $\mathbf{X}(1)$ denote the cache state that evicts a token from conversation $i(1)$, and $\mathbf{X}(k)$ denote the cache state that evicts a token from conversation $i(k)$ with $k > 1$.

$$\begin{aligned}
& J_{k+1}(\tilde{\boldsymbol{\lambda}}, \tilde{L}, \mathbf{X}(1)) \\
&= \sum_{i \in \dim(\mathbf{X}(1))+1} \tilde{p}_i \mathbb{E}_{Q_i}[(\tilde{L}_i + Q_i - X_i(1) - \xi)^+] \\
&\quad + \tilde{p}_{i(1)} \mathbb{E}_{Q_{i(1)}, A_{i(1)}} \left[\min_{\mathbf{X}'(1) \in \mathcal{X}(\mathbf{X}(1), i(1), \tilde{L}_{i(1)} + Q_{i(1)} + A_{i(1)})} J_{k+2}(\Gamma(\tilde{\boldsymbol{\lambda}}, i(1)), \Psi(\tilde{L}, i(1), Q_{i(1)} + A_{i(1)}), \mathbf{X}'(1)) \right] \\
&\quad + \tilde{p}_{i(k)} \mathbb{E}_{Q_{i(k)}, A_{i(k)}} \left[\min_{\mathbf{X}'(1) \in \mathcal{X}(\mathbf{X}(1), i(k), \tilde{L}_{i(k)} + Q_{i(k)} + A_{i(k)})} J_{k+2}(\Gamma(\tilde{\boldsymbol{\lambda}}, i(k)), \Psi(\tilde{L}, i(k), Q_{i(k)} + A_{i(k)}), \mathbf{X}'(1)) \right] \\
&\quad + \sum_{i \neq i(1), i(k)} \mathbb{E}_{Q_i, A_i} \left[\min_{\mathbf{X}'(1) \in \mathcal{X}(\mathbf{X}(1), i, \tilde{L}_i + Q_i + A_i)} J_{k+2}(\Gamma(\tilde{\boldsymbol{\lambda}}, i), \Psi(\tilde{L}, i, Q_i + A_i), \mathbf{X}'(1)) \right] \\
&\leq \sum_{i \in \dim(\mathbf{X}(1))+1} \tilde{p}_i \mathbb{E}_{Q_i}[(\tilde{L}_i + Q_i - X_i(k) - \xi)^+] \\
&\quad + \tilde{p}_{i(1)} \mathbb{E}_{Q_{i(1)}, A_{i(1)}} \left[\min_{\mathbf{X}'(k) \in \mathcal{X}(\mathbf{X}(k), i(1), \tilde{L}_{i(1)} + Q_{i(1)} + A_{i(1)})} J_{k+2}(\Gamma(\tilde{\boldsymbol{\lambda}}, i(1)), \Psi(\tilde{L}, i(1), Q_{i(1)} + A_{i(1)}), \mathbf{X}'(k)) \right] \\
&\quad + \tilde{p}_{i(k)} \mathbb{E}_{Q_{i(k)}, A_{i(k)}} \left[\min_{\mathbf{X}'(k) \in \mathcal{X}(\mathbf{X}(k), i(k), \tilde{L}_{i(k)} + Q_{i(k)} + A_{i(k)})} J_{k+2}(\Gamma(\tilde{\boldsymbol{\lambda}}, i(k)), \Psi(\tilde{L}, i(k), Q_{i(k)} + A_{i(k)}), \mathbf{X}'(k)) \right] \\
&\quad + \sum_{i \neq i(1), i(k)} \mathbb{E}_{Q_i, A_i} \left[\min_{\mathbf{X}'(k) \in \mathcal{X}(\mathbf{X}(k), i, \tilde{L}_i + Q_i + A_i)} J_{k+2}(\Gamma(\tilde{\boldsymbol{\lambda}}, i), \Psi(\tilde{L}, i, Q_i + A_i), \mathbf{X}'(k)) \right] \\
&= J_{k+1}(\tilde{\boldsymbol{\lambda}}, \tilde{L}, \mathbf{X}(k)),
\end{aligned}$$

where the inequality holds as

- by Definition 1, $\mathbf{X}(1)$ is the optimal solution while $\mathbf{X}(k)$ is a feasible solution, and \tilde{p}_i are proportional to $\tilde{\lambda}_i$, thus

$$\sum_{i \in \dim(\mathbf{X}(1))} \tilde{p}_i \mathbb{E}_{Q_i}[(\tilde{L}_i + Q_i - X_i(1) - \xi)^+] \leq \sum_{i \in \dim(\mathbf{X}(1))} \tilde{p}_i \mathbb{E}_{Q_i}[(\tilde{L}_i + Q_i - X_i(k) - \xi)^+]$$

and the expected cost incurred when a new conversation arrives (with probability $\tilde{p}_{\dim(\mathbf{X}(1))+1}$) is $\mathbb{E}_Q[(Q - \xi)^+]$ for both caching state, thus

$$\sum_{i \in \dim(\mathbf{X}(1))+1} \tilde{p}_i \mathbb{E}_{Q_i}[(\tilde{L}_i + Q_i - X_i(1) - \xi)^+] \leq \sum_{i \in \dim(\mathbf{X}(1))+1} \tilde{p}_i \mathbb{E}_{Q_i}[(\tilde{L}_i + Q_i - X_i(k) - \xi)^+]$$

- by induction hypothesis, the optimal $\mathbf{X}'(1)^*$ and $\mathbf{X}'(k)^*$ are given by the optimization problem (8). Fix a user prompt length Q_i and a model response length A_i and we compare the cost-to-do under $\mathbf{X}(1)$ and $\mathbf{X}(k)$.

- if conversation $i(1)$ arrives next, then one need to evict one more token from $\mathbf{X}(1)$ than from $\mathbf{X}(k)$. Suppose the extra token evicted from $\mathbf{X}(1)$ is from conversation $i(k)$, then $\mathbf{X}'(1)^* = \mathbf{X}'(k)^*$. If not, then this means the extra token evicted is from another conversation with better ranking criterion, thus we have

$$\begin{aligned}
& J_{k+2}(\Gamma(\tilde{\boldsymbol{\lambda}}, i(1)), \Psi(\tilde{L}, i(1), Q_{i(1)} + A_{i(1)}), \mathbf{X}'(1)^*) \\
& \leq J_{k+2}(\Gamma(\tilde{\boldsymbol{\lambda}}, i(1)), \Psi(\tilde{L}, i(1), Q_{i(1)} + A_{i(1)}), \mathbf{X}'(k)^*)
\end{aligned}$$

by the induction hypothesis.

- if conversation $i(k)$ arrives next, then one need to evict one more token from $\mathbf{X}(k)$ than from $\mathbf{X}(1)$. In the optional caching model, the extra token evicted from $\mathbf{X}(k)$ must be from conversation $i(1)$ by the definition of the ranking of conversations, thus $\mathbf{X}'(1)^* = \mathbf{X}'(k)^*$.

$$\begin{aligned} & J_{k+2}(\Gamma(\tilde{\lambda}, i(k)), \Psi(\tilde{L}, i(k), Q_{i(k)} + A_{i(k)}), \mathbf{X}'(k)^*) \\ &= J_{k+2}(\Gamma(\tilde{\lambda}, i(k)), \Psi(\tilde{L}, i(k), Q_{i(k)} + A_{i(k)}), \mathbf{X}'(k)^*) \end{aligned}$$

- if conversation other than $i(1), i(k)$ arrives next, then $\mathbf{X}'(k)^*$ and $\mathbf{X}'(1)^*$ need to evict the same number of tokens. If $\mathbf{X}'(k)$ evicts at least one token from conversation $i(k)$, then $\mathbf{X}'(1)^* = \mathbf{X}'(k)^*$. If not, then this means $\mathbf{X}'(1)^*$ evicts one token from another conversation with better ranking criterion, thus we have

$$J_{k+2}(\Gamma(\tilde{\lambda}, i), \Psi(\tilde{L}, i, A_i), \mathbf{X}'(1)^*) \leq J_{k+2}(\Gamma(\tilde{\lambda}, i), \Psi(\tilde{L}, i, A_i), \mathbf{X}'(k)^*).$$

Therefore, by induction, the result holds for all $k \geq 1$. \square

Greedy Implementation. The optimization problem (8) need not be solved explicitly as a token-by-token greedy procedure suffices. At a high-level, the policy ranks each token by arrival rates weighted by its counterfactual cost, i.e., the cost increase when we evict this token.

$$\mathbb{P}(L_i + Q_i - \xi \geq X_i) = \mathbb{E}[(L_i + Q_i - \xi - (X_i - 1))^+] - \mathbb{E}[(L_i + Q_i - \xi - X_i)^+]$$

i.e., the difference in expected cost if we further evict one token when we have X_i tokens in cache.

Algorithm 2: Expected-Tail-Optimized LRU Policy

Input: Number of conversations N , cache sizes $\{X_i\}$, current lengths $\{L_i\}$, belief turn rates $\{\lambda_i\}$, distribution of length of user prompt $\{Q_i\}$, threshold ξ , arriving conversation θ , arriving user prompt length Q , arriving model response length A , tokens to evict n

Output: Updated cache sizes $\{X_i\}$

```

evicted ← 0
 $L_\theta \leftarrow L_\theta + Q + A$  // Update system state for arriving conversation  $\theta$ 
 $X_\theta \leftarrow \bar{L}_\theta$ 
 $\lambda_\theta \leftarrow \bar{\lambda}_\theta$ 
for each  $i \in E$  do
| Compute  $v_i \leftarrow \lambda_i \cdot \mathbb{P}(L_i + Q_i - \xi \geq X_i)$  // Ranking criterion
while evicted <  $n$  do
| Find  $j = \arg \min_{i \in [N], X_i \geq 1} v_i$  // Conversation with minimum value
|  $X_j \leftarrow X_j - 1$  // Evict one token
| evicted ← evicted + 1
|  $v_j \leftarrow \lambda_j \cdot \mathbb{P}(L_j + Q_j - \xi \geq X_j)$  // Update ranking criterion
return  $\{X_i\}$ .
```

In the implementation of the policy, one can use min-heap to process which token to evict using the ranking criterion. The computational complexity of the policy is given by $\mathcal{O}(|E| + n \log |E|)$, where $|E|$ is number of conversations with non-zero cached tokens and n is the number of tokens one needs to evict.

We show that policy 2 indeed returns a cache state that is an optimal solution to the optimization problem (8).

Lemma 1. *Policy 2 returns an optimal solution to the optimization problem (8).*

Proof. We prove by contradiction. Note that it is possible for the policy to return multiple optimal solutions, and it is also possible for the optimization problem (8) to have multiple optimal solutions. Suppose not, then the two set of solutions do not intersect. Let \mathbf{X}^* denote one optimal solution. Then there must exist two conversations i, j such that $X_j^* \geq 1$ and

$$\lambda_i \mathbb{P}(L_i + Q_i - \xi \geq X_i^* + 1) > \lambda_j \mathbb{P}(L_j + A_j - \xi \geq X_j^*),$$

Then we can construct another solution \mathbf{X}' such that $X'_k = X^*_k$ for $k \neq i, j$, and $X'_i = X^*_i + 1$, $X'_j = X^*_j - 1$. Then the difference between the objective values of \mathbf{X}' and \mathbf{X}^* is given by

$$\begin{aligned} \text{OBJ}(\mathbf{X}') - \text{OBJ}(\mathbf{X}^*) &= \lambda_i \mathbb{E}[(L_i + Q_i - \xi - (X^*_i + 1))^+] + \lambda_j \mathbb{E}[(L_j + Q_j - \xi - (X^*_j - 1))^+] \\ &\quad - (\lambda_i \mathbb{E}[(L_i + Q_i - \xi - X^*_i)^+] + \lambda_j \mathbb{E}[(L_j + Q_j - \xi - X^*_j)^+]) \\ &= \lambda_j \mathbb{P}(L_j + A_j - \xi \geq X_j) - \lambda_i \mathbb{P}(L_i + Q_i - \xi \geq X_i + 1) \\ &< 0, \end{aligned}$$

which contradicts the optimality of \mathbf{X}^* . \square

D Discussion on Forced Caching

Implementation of Tail-Optimized LRU. To implement forced caching, especially at GPU level, the server needs to decide which block to evict as serving the turn. The server may not know the total number of cache blocks to evict due to the uncertainty in the model response length, nevertheless the server can repeatedly call our policy to evict more tokens if needed.

Hindsight optimal policy. To model forced caching, we replace optional caching constraint (5) with

$$x_{i,t+1} = \sum_{j=1}^t (q_{i,j} + a_{i,j}), \forall i \in [N], t \in \mathcal{T}_i \text{ and } t < T, \quad (13)$$

i.e., when a turn arrives, the server is required to cache its whole conversation history including newly generated response. Theorem 1 continues to hold under forced caching.

Expected-Tail-Optimized LRU. To model forced caching, we replace feasible caching decision space under optional caching with

$$\mathcal{X}_{\mathcal{F}}(\mathbf{X}, \theta, L) = \{\mathbf{Y} \in \mathbb{N}^{\dim(\mathbf{X}, \theta)} : \sum_i Y_i \leq C, Y_\theta = L, 0 \leq Y_i \leq X_i, i \neq \theta\}.$$

Theorem 2 continues to hold under forced caching, i.e., Expected-Tail-Optimized LRU remains to be optimal, if

- every future prompt (if it arrives) has a known, fixed length $Q \geq 0$. Here this fixed length can be heterogeneous across conversations and across turns. Crucially, the decision-maker still does not know if any given conversation will return; they only know that should it return, its next-turn question length will be Q . In this case, Expected-Tail-Optimized LRU is reduced to a deterministic version as stated in policy 1.
- conversations have homogeneous turn rates λ_{turn} .

This fixed-prompt-length assumption holds when prompts are pre-specified. When $Q = 0$ after the first turn and responses also have zero length, our model reduces to classic paging with unit page size.

E Additional Figures

F KV Cache Size Computation

We calculate the KV cache memory requirements for the Vicuna-7B model. For 10,000 tokens stored in Float16 precision, the total KV cache size is approximately **4.88 GB**. With Float32 precision, this memory requirement doubles to approximately 9.77 GB.

F.1 Model Configuration

The following parameters are from the Vicuna-7B-v1.5 model configuration.⁸

- Hidden Size: 4096

⁸<https://huggingface.co/lmsys/vicuna-7b-v1.5/blob/main/config.json>

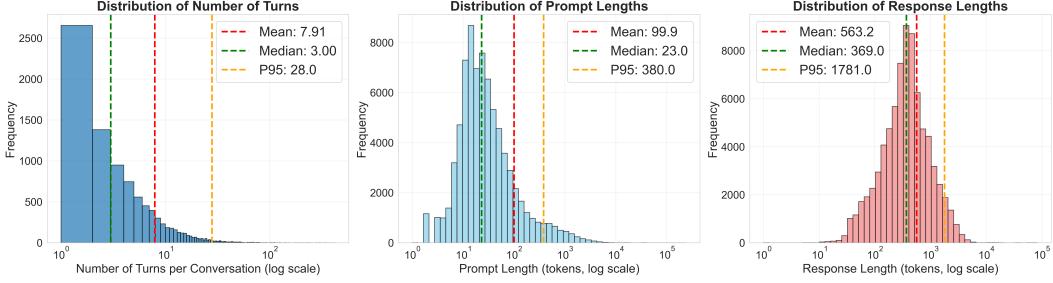


Figure 5: Distributions of turns and tokens of ShareGPT [Contributors, 2025] datasets (sampled 10,000 conversations).

- Number of Attention Heads: 32
- Number of Hidden Layers: 32
- Number of Key-Value Heads: 32
- Head Size: 128 (Hidden Size \div Number of Attention Heads)
- Data Type: Float16 (2 bytes per value)

F.2 Calculation

The KV cache size per token is computed as:

$$\begin{aligned} \text{KV cache per token (bytes)} &= 2 \times \text{Layers} \times \text{KV Heads} \times \text{Head Size} \times \text{Data Type Size} \\ &= 2 \times 32 \times 32 \times 128 \times 2 \\ &= 524,288 \text{ bytes} \end{aligned}$$

where the leading factor of 2 accounts for storing both Key (K) and Value (V) matrices.

For 10,000 tokens, the total memory requirement is:

$$\begin{aligned} \text{Total KV cache (bytes)} &= 10,000 \times 524,288 \\ &= 5,242,880,000 \text{ bytes} \end{aligned}$$

Converting to gigabytes:

$$\begin{aligned} \text{KV cache size (GB)} &= \frac{5,242,880,000}{1024^3} \\ &\approx 4.8828 \text{ GB} \end{aligned}$$

G Additional Experiment Results

G.1 Offline Baselines and T-LRU Variants with Future Knowledge.

Recent work on queueing systems has shown that machine learning predictions, such as predicted service times or job characteristics, can significantly improve scheduling and resource allocation decisions [Mitzenmacher and Shahout, 2025]. In the context of LLM inference systems, various types of predictions (e.g., conversation continuation, prompt lengths) could also potentially enhance caching performance. This motivates a natural question: *which predictions are most valuable for caching, and how much improvement can each type unlock?* To answer this, we design T-LRU variants with different levels of future knowledge, creating a predictability spectrum that quantifies the marginal benefit of each prediction type.

We evaluate three policies with increasing levels of future knowledge:

- **T-LRU (baseline):** Uses the empirical average prompt length to predict future requests. No knowledge of conversation termination or actual prompt lengths.

- **End-Aware T-LRU:** Knows whether each conversation will continue (return) or terminate, but does not know future prompt lengths. This variant evicts all blocks from terminating conversations and follows standard T-LRU for continuing ones.
- **Length-Aware T-LRU:** Knows both conversation continuations AND the exact length of the next user prompt. This variant uses exact prompt lengths for TEL-safe trimming and evicts all caches when conversations end.

Additionally, we include Tail-Optimized Belady, the hindsight-optimal policy for TEL that knows the entire future arrival sequence. This serves as an upper bound on achievable performance.

Figure 6 reports the latency distributions (median, P90, P95, P99) under different caching policies across varying cache capacities on the WildChat dataset.

We highlight two observations from Figure 6. First, tail improvement is achieved with a modest cost to the median latency. T-LRU (blue squares) consistently beats LRU and Threshold-LRU at the tail (around 300 ms), but its median latency is slightly higher (increased from 10ms to 40ms). The trade-off is expected and in fact intended. Median latency degradation is minimal and users will likely not notice: for context, the duration of a blink is on average 100–400 milliseconds according to the Harvard Database of Useful Biological Numbers [BioNumbers Database, 2024]. T-LRU deliberately trades off slightly higher median latency (a few milliseconds) to achieve larger tail latency reductions (tens to hundreds of milliseconds), which is more favorable for user experience.

Second, a single-bit forecast “will this conversation continue?” is a remarkably powerful signal. End-Aware T-LRU performs much better than T-LRU, while Length-Aware T-LRU gains only a small additional edge from knowing the exact prompt length. In practice, predicting whether a single conversation will continue is much easier than forecasting exact prompt sizes, and vastly easier than predicting the full arrival sequences required by Tail-Optimized Belady. Existing works like [Jin et al., 2023] propose models to predict the length of model response with up to 98.61% prediction accuracy, underscoring the practical viability of deploying End-Aware policies.

Note that Tail-Optimized Belady is the optimal policy for our TEL objective for the ξ_s chosen, thus it is not necessarily the optimal policy for tail latency at different levels. On the other hand, solving for the optimal policy for tail latency is computationally hard.

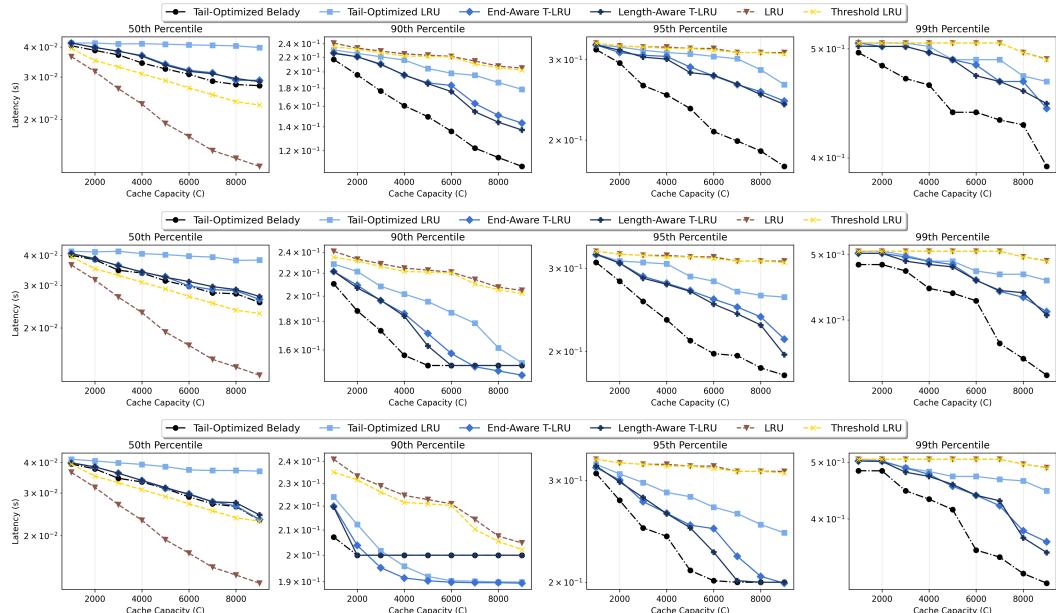


Figure 6: Latency results for various settings (threshold latency $\xi_s = 100, 200, 300$ ms) from top to bottom panels.

G.2 Results on ShareGPT with Synthetic Timestamps

ShareGPT [Contributors, 2025] does not include timestamps of each request, thus we generate them with the stochastic model described in Section 4. Specifically, for each conversation, we draw exponential inter-arrival times with rate $\lambda_{\text{conv}} = 1$, then for each conversation, we generate inter-arrival times between each turn within a conversation using Exponential distribution with rate $\lambda_{\text{turn}} = 3$. The average prompt length in ShareGPT is approximately 100 tokens (we thus set $\hat{Q} = 100$ in implementation), with an average of 3.5 turns per conversation.

Tables 4–5 show that Tail-Optimized LRU still beats both LRU and Threshold-LRU: it trims P90 by up to 10%, and P95 by up to 7%. The smaller improvement compared to the ones observed in WildChat (Tables 1–2) stem from the already-high base latencies under LRU (with capacity $C = 1000$ under LRU, medium, P90, P95, P99 tail latencies are roughly 209 ms, 1415 ms, 2447 ms, 3649 ms), thus percentage improvements shrink.

Table 4: Relative latency improvement of T-LRU over LRU with various ξ_s (ShareGPT)

Capacity	$\xi_s = 50\text{ms}$		$\xi_s = 100\text{ms}$		$\xi_s = 200\text{ms}$		$\xi_s = 300\text{ms}$		$\xi_s = 500\text{ms}$	
	p90	p95	p90	p95	p90	p95	p90	p95	p90	p95
1000	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.9%	0.6%	0.9%	0.9%
2000	0.7%	0.0%	0.7%	0.2%	0.9%	1.5%	1.4%	2.0%	2.7%	2.3%
4000	0.6%	0.6%	0.7%	1.8%	2.0%	2.5%	2.5%	3.0%	4.2%	3.5%
6000	0.7%	1.3%	2.1%	2.9%	4.9%	3.6%	8.1%	4.2%	10.0%	4.7%
8000	1.5%	0.9%	2.6%	1.8%	3.5%	2.5%	4.8%	3.6%	9.6%	5.0%
10000	0.9%	0.7%	3.6%	1.7%	4.3%	2.9%	5.1%	3.6%	9.0%	6.9%

Table 5: Relative latency improvement of T-LRU over Threshold-LRU with various ξ_s (ShareGPT)

Capacity	$\xi_s = 50\text{ms}$		$\xi_s = 100\text{ms}$		$\xi_s = 200\text{ms}$		$\xi_s = 300\text{ms}$		$\xi_s = 500\text{ms}$	
	p90	p95	p90	p95	p90	p95	p90	p95	p90	p95
1000	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.9%	0.6%	0.9%	0.9%
2000	0.7%	0.0%	0.7%	0.2%	0.9%	1.5%	1.4%	2.0%	2.7%	2.3%
4000	0.6%	0.6%	0.7%	1.8%	2.0%	2.5%	2.5%	3.0%	4.2%	3.5%
6000	0.7%	0.8%	2.1%	2.4%	4.9%	3.1%	8.1%	3.7%	10.0%	4.2%
8000	0.8%	0.3%	2.0%	1.2%	2.9%	2.0%	4.1%	3.0%	9.0%	4.5%
10000	0.9%	0.6%	3.6%	1.6%	4.3%	2.8%	5.1%	3.5%	9.0%	6.8%

Using a 200 ms SLO, T-LRU cuts the share of requests above the budget by 2–8% across capacities (Table 6). Improvements again peak when ξ_s is near the desired percentile; extremely high ξ_s trades those mid-tail wins for heavier protection of the extreme tail, echoing the WildChat pattern.

Table 6: Relative improvement of T-LRU: % reduction in requests with latency $> 200\text{ms}$ (ShareGPT)

Capacity	$\xi_s = 50\text{ms}$		$\xi_s = 100\text{ms}$		$\xi_s = 150\text{ms}$		$\xi_s = 200\text{ms}$		$\xi_s = 500\text{ms}$	
	LRU	Thre-LRU	LRU	Thre-LRU	LRU	Thre-LRU	LRU	Thre-LRU	LRU	Thre-LRU
1000	0.7%	0.0%	1.7%	1.0%	2.3%	1.6%	2.3%	1.6%	-3.5%	-4.2%
2000	1.1%	1.0%	1.9%	1.8%	2.6%	2.5%	3.1%	3.0%	-8.6%	-8.7%
4000	1.8%	1.3%	3.9%	3.4%	4.7%	4.2%	4.8%	4.3%	-15.6%	-16.2%
6000	1.9%	1.1%	4.7%	3.9%	6.0%	5.2%	4.7%	3.9%	-26.2%	-27.2%
8000	1.2%	0.9%	4.3%	4.0%	4.9%	4.6%	2.9%	2.6%	-39.3%	-39.7%
10000	2.2%	1.8%	6.5%	6.1%	7.9%	7.6%	3.3%	3.0%	-50.7%	-51.2%

Lastly, in spite of the extra foresight, End-Aware T-LRU and Length-Aware T-LRU show only marginal gains over T-LRU, and all three policies perform very closely to Tail-Optimized Belady, the optimal hindsight policy that minimizes the Tail Excess Latency. This is exactly what our stochastic model predicts: under Poisson arrivals, LRU’s recency order is already a near-perfect proxy for “furthest in the future”, the rule the hindsight policy uses for eviction, so extra foresight offers diminishing returns.

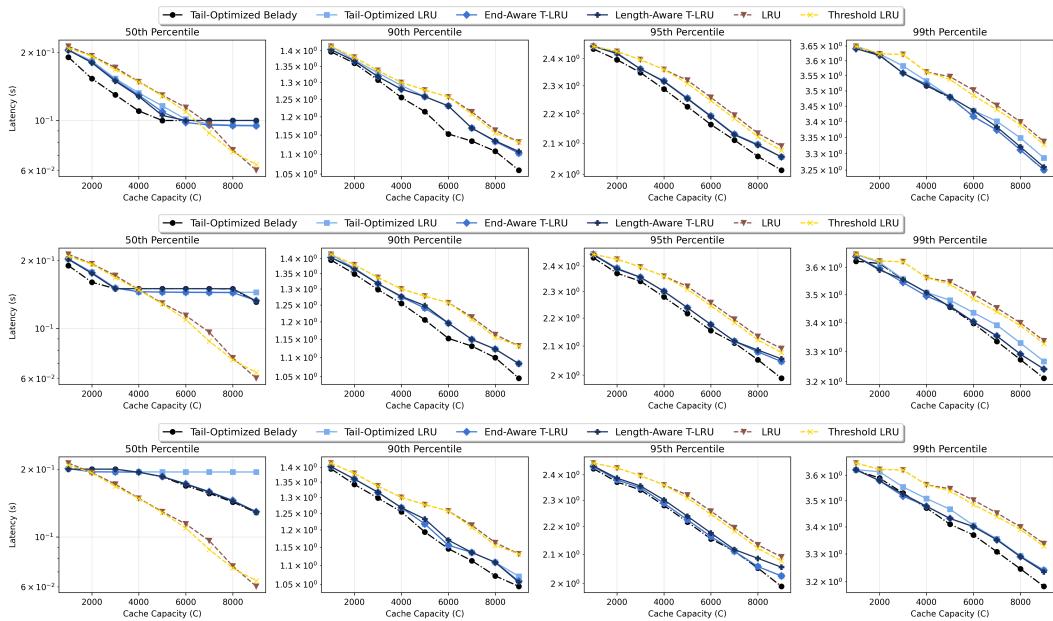


Figure 7: Latency results for various settings (threshold latency $\xi_s = 100, 200, 300$ ms) from top to bottom panels (ShareGPT)