Tail-Optimized Caching for LLM Inference

Wenxin Zhang [1] **Yueying Li** [2] **Tianyi Peng** [1] **Ciamac C. Moallemi** [1]

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 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 40% 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. 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.

Tail latency drives user experience but is hard to optimize. In real-time user-facing applications, companies care about high-percentile response time, e.g., 95% of requests complete within 200 ms. In LLM applications, Users arrive to request services through an alternating sequence of prompts and responses that we call *turns*. Each prompt, along with all previous chat 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 algorithms mostly focus on maximizing *cache hit rate*, not *tail latency*. Existing caching designs for LLM utilize Least-Recently-Used (LRU) as the eviction policy, including vLLM (Kwon et al., 2023), SGLang (Zheng et al., 2024), and Mooncake (Qin et al., 2025). LRU evicts the least recently accessed item in the cache, but it does not distinguish between long or short conversations that could have different implications for tail latency: evicting 10 tokens from a 2000-token chat is far more likely to hurt tail latency than from a 10-token chat. Therefore, LRU can perform arbitrarily poorly on tail latency.

Our approach: Tail-Optimized LRU. To bridge this gap, we introduce *Tail Excess Latency*: TEL =

[1]Columbia University [2]Cornell University. Correspondence to: Wenxin Zhang <wz2574@columbia.edu>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

$\sum_i \max\{\text{TTFT}(i) - \xi, 0\}$, where $\text{TTFT}(i)$ denote the Time to First Token experienced by the i^{th} request and $\xi \geq 0$ is a user-specific latency threshold. We approximate TTFT by the number of uncached blocks.

For a conversation with chat history length L and whose next prompt is expected to add Q blocks, caching more than $L + Q - \xi$ blocks 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" (Figure 1). Motivated by this observation, we propose Tail-Optimized LRU, a two-line addition of LRU. Upon cache overflow, Tail-Optimized LRU works in two phases: 1) Proactive trimming: first evict blocks from conversations whose cache size exceeds the TEL-safe budget $L + Q - \xi$; 2) If space is still needed, evict blocks using LRU.

In practice, the next-prompt length is unknown; T-LRU can use an estimate $\hat{Q}$ such as the empirical average or a model prediction. Implementation requires only one extra bookkeeping: mark proactively trimmed blocks as "infinitely old," after which any existing LRU engine can evict them in the usual way.

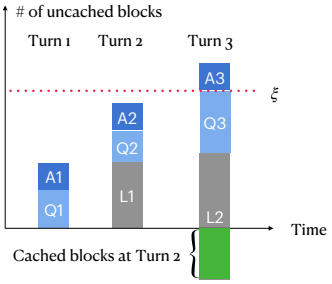

Figure 1: Proactive trimming: A three-turn conversation $\{(Q_1, A_1), (Q_2, A_2), (Q_3, A_3)\}$ is shown, L_i denotes chat history (i.e., $L_1 = Q_1 + A_1, L_2 = L_1 + Q_2 + A_2$). Bars indicate the number of uncached blocks each turn incurs; the red dashed line marks the latency threshold ξ. Turn 1: 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.

Contributions.

- We derive the hindsight optimal policy for Tail Excess Latency: evict down to the TEL-safe budget $L + Q - \xi$ then use furthest-in-future policy (Theorem B.1).
- We introduce a novel policy, Tail-Optimized LRU (Algorithm 1), which dominates LRU under any arrival trace (Theorem 3.1) and requires minimal change to LRU in implementation.
- We propose a novel stochastic model for multi-turn conversations that characterize uncertainties in LLM workload, which should be of broad interest to the caching community. Within it, we prove a generalized Tail-Optimized LRU is optimal (Theorem D.2); demonstrating the optimality of classic LRU (for average latency) and Tail-Optimized LRU as special cases.
- We evaluate variants of Tail-Optimized LRU, LRU, and hindsight optimal policy on real multi-turn chat traces. Our policy achieves up to 23.9% reduction in P95 tail latencies compared with LRU, along with up to 38.9% decrease in SLO violations compared with the strongest baseline, and offers insights on how to choose the latency threshold ξ (Section 4).

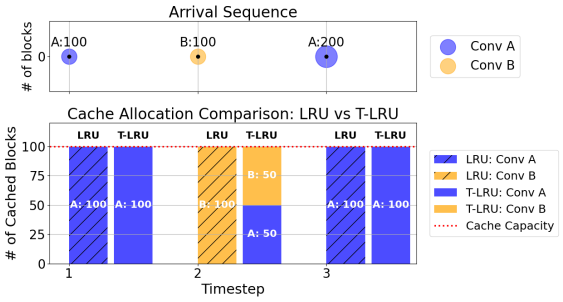

Figure 2: Difference between LRU and T-LRU: In this example, there are two conversations with a total of three turns. The top panel shows the total job size (chat history plus user prompt) at each step. For simplicity we assume response length is zero. The bottom panel shows the number of cached blocks updated after each turn arrival. The cache capacity is 100 blocks and user prompt length does not exceed 100 blocks. LRU evicts all of conversation A's cache blocks immediately after step 2, causing a maximum uncached blocks of 200 at step 3. In contrast, Tail-Optimized LRU (T-LRU) with $\xi = 160$ partially caches conversation B due to proactive trimming—reserving cache space for conversation A and brings the maximum uncached blocks to 150. That's a 50-block improvement over LRU, closing about 50% of the gap to the hindsight optimum of 100 uncached blocks.

2. Preliminary

Caching Decision. When a request arrives, the server can reuse any KV cache blocks from its chat history; evicted blocks must be recomputed.[1] As keeping KV blocks is strictly beneficial while space remains, the policy only needs to specify what to evict when the cache overflows. We focus on *optional caching*, where computation uses a separate workspace from the KV cache memory. After serving the request and updating its chat history, the server decides which newly produced blocks to save and evict to satisfy cache capacity. This allows us to treat the number of blocks to evict as a fixed target, as one can make caching decisions

[1] In practice, evicted blocks may be retrieved from other storage layers (e.g., CPU DRAM or SSD). We focus on a single-layer cache and do not consider inter-layer transfer.

after serving the request. See Appendix E for discussion of forced caching and how our results extend to that setting.

Latency Objective: Tail Excess Latency. Percentile tail latency is notoriously hard to optimize. Instead, we introduce the Tail Excess Latency (TEL) metric as an amenable approximation: $\text{TEL} = \sum_i \max\{\text{TTFT}_i - \xi, 0\}$, where TTFT_i denote the Time to First Token experienced by the i^{th} request and $\xi \geq 0$ is a user-chosen threshold. TEL mirrors Conditional Value at Risk (CVaR) but with an explicit, user-chosen threshold (Bäuerle & Ott, 2011; Chow et al., 2015); it also resembles SLO attainment metrics (Zhong et al., 2024), but TEL weighs larger violations proportionally more. Setting $\xi = 0$ recovers the average-latency objective. Although our theory centers on TEL, in the experiments we report conventional metrics (tail latency and SLO violation rate) to allow straightforward comparisons in future studies. Section 4 also discusses how to pick ξ.

Why does caching matter for TEL? Empirically, TTFT grows roughly linearly with the number of *uncached* token blocks that must be processed during the prefill (Figure 3). This is potentially due to (1) parallelization in the prefill stage and (2) the dominance of the feed-forward linear layers during inference (Kamath et al., 2025; Zhu et al., 2024; Ye et al., 2025). Therefore, we adopt this approximation: TTFT = chat history length + current prompt length − cached history length, all in units of the number of KV blocks cached.

3. Tail-Optimized LRU

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.

Pseudocode is given in Algorithm 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 3.1) eliminates adoption risk, while the lightweight modifications from standard LRU (Section 3) minimize adoption costs.

Theorem 3.1 (Dominance of Tail-Optimized LRU)**.** *Let $\bar{Q}$ be a known upper bound on prompt lengths. For every possible arrival trace and identical initial cache state, Tail-Optimized LRU runs with $\bar{Q}$ incurs lower Tail Excess Latency than LRU.*

Theorem 3.1 ensures that our proposed policy never degrades system performance compared to LRU. Such a guarantee stems from two properties of T-LRU. First, by using a conservative upper bound $\bar{Q}$ as the next-prompt length, our "free eviction" phase never increases latency above the latency threshold ξ than LRU. In fact, by setting $\bar{Q} \geq \xi$, Tail-Optimized LRU is reduced to LRU. Second, proactively trimming cache blocks within TEL-safe budget frees up space to cache conversations that would otherwise contribute to tail latency.

Algorithm 1 Tail-Optimized LRU Policy

Input: Number of conversations N, Time to Last Turn $\{\tau_i\}$, cache sizes $\{X_i\}$, 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 $\{X_i\}$

$L_\theta \leftarrow L'_\theta, X_\theta \leftarrow L_\theta, \tau_\theta \leftarrow 0$ `// Update system state for arriving conversation`
while $\sum_{i\in[N]} X_i > C$ **do**
  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 $j = \arg\max_{i\in[N]:X_i\geq 1} \tau_i$ `// 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 the paged KV cache technique, which stores keys and values in non-contiguous memory space.

4. Experiments

4.1. Datasets and Metrics.

We conduct experiments on two chat datasets: ShareGPT and WildChat (Figure 4). We sample the first 1000-2000 turns based on a subset of the conversation traces and calculate the (medium, P90, P95, P99) latency under different caching policies. The model we used is Vicuna-7b and Mixtral-8x7B, and the default latency is measured on A100 with Vicuna with TP = 1 (no tensor parallelism) and without mixed bathing. Results on ShareGPT can be found in Appendix G.2.

4.2. Tail Latency Reduction.

We measure the tail latency reduction of our policy against LRU and *Threshold LRU*—caches *only if* the conversation

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

	ξ = 50ms		ξ = 100ms		ξ = 200ms		ξ = 300ms		ξ = 500ms		
Capacity	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%

Table 2: Relative improvement of T-LRU: % reduction in requests with latency $>$ 200ms

Capacity	ξ = 50ms		ξ = 100ms		ξ = 150ms		ξ = 200ms		ξ = 500ms	
	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%

length exceeds a fixed threshold, then fall back to LRU for further evictions. We fix the input parameter next-prompt length $\hat{Q}$ for Tail-Optimized LRU to be the average prompt length (200 for WildChat), and use 1024 as the threshold for Threshold-LRU following the one used by OpenAI (OpenAI, 2024). Across cache capacities C and tail-latency threshold ξ, T-LRU lowers P90 tail TTFT by up to 27.5% and P95 tail TTFT by up to 23.9% relative to LRU, and achieves similar gains over Threshold-LRU (Tables 1–3).

Sensitivity to latency threshold ξ. The benefit peaks when latency threshold ξ matches the target tail percentile. *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.* Therefore, setting $\xi = 200$ ms yields the largest improvement in P90 tail latency; setting $\xi = 300$ ms yields the largest improvement in P95 tail latency. A high value, e.g. $\xi = 500$ ms, relaxes protection for moderate tails and only provides protection for extreme tails —up to 3% of increase in P99 tail latency.

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

Comparing Threshold-LRU and Tail-Optimized LRU. Threshold-LRU is a straightforward patch for LRU's blindness to conversation length—in industry systems such as at OpenAI, prompt-cache is enabled only when the running chat history exceeds a fixed length. The heuristic is simple but ignores the forthcoming prompt length, so it can underprotect a short history followed by a long prompt. The two policies differ in the question they ask at eviction time: Threshold-LRU asks if the chat history exceeds a threshold whereas Tail-Optimized LRU asks if the projected next request length (chat history + next prompt length) exceeds a threshold. In fact, Theorem 3.1 generalizes: if Threshold-LRU uses a threshold cutoff of $\xi - \bar{Q}$ with $\bar{Q}$ denotes the upper bound on user prompt length, then T-LRU using $\bar{Q}$ incurs less Tail Excess Latency on any arrival trace.

4.3. SLO Violation Reduction.

We now measure the *count* of requests that a service-level objective, another objective similar to Tail Excess Latency—the *amount* of latency beyond a threshold. Table 2 reports the improvement on SLO attainment ratio (relative drop in requests whose TTFT exceeds 200 ms). With the latency threshold ξ 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. When ξ is much lower than the SLO ($\xi = 50$ or 100 ms), the improvement is modest because both baselines already satisfy most requests; when ξ is far higher ($\xi = 500$ 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, yet the same trimming logic also cuts the *number* of SLO violations whenever ξ aligns or is slightly below the target latency budget.

5. Future Directions and Conclusions

We propose a simple modification to the Least Recently Used caching policy that improve tail latency in multi-turn conversation LLM service. Our focus is on a single storage

layer, and exploring hierarchical KV caching architectures such as in (Qin et al., 2025), could provide insights into better managing multi-layer resources. Another interesting direction is how to jointly optimize caching and load balancing. (Srivatsa et al., 2024) has initiated work on this topic, and further studies using a queuing theory perspective could systematically analyze the trade-offs involved.

References

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

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

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

Berger, D. S., Berg, B., Zhu, T., Sen, S., and Harchol-Balter, M. {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)*, pp. 195–212, 2018.

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

Borodin, A., Irani, S., Raghavan, P., and Schieber, B. Competitive paging with locality of reference. *Journal of Computer and System Sciences*, 50(2):244–258, 1995.

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

Chow, Y., Tamar, A., Mannor, S., and Pavone, M. Risk-sensitive and robust decision-making: a cvar optimization approach. *Advances in neural information processing systems*, 28, 2015.

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

Coffman, E. G. and Denning, P. J. *Operating systems theory*, volume 973. prentice-Hall Englewood Cliffs, NJ, 1973.

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

Dan, A. and Towsley, D. 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*, pp. 143–152, 1990.

Fiat, A., Karp, R. M., Luby, M., McGeoch, L. A., Sleator, D. D., and Young, N. E. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

Gim, I., Chen, G., Lee, S.-s., Sarda, N., Khandelwal, A., and Zhong, L. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338, 2024.

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

Jin, C., Zhang, Z., Jiang, X., Liu, F., Liu, X., Liu, X., and Jin, X. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *arXiv preprint arXiv:2404.12457*, 2024.

Jin, Y., Wu, C.-F., Brooks, D., and Wei, G.-Y. s^3: Increasing gpu utilization during generative inference for higher throughput. *Advances in Neural Information Processing Systems*, 36:18015–18027, 2023.

Kamath, A. K., Prabhu, R., Mohan, J., Peter, S., Ramjee, R., and Panwar, A. 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*, pp. 897–912, 2025.

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

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Leonardi, E. and Torrisi, G. L. Least recently used caches under the shot noise model. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pp. 2281–2289. IEEE, 2015.

Lin, J., Zhang, S. Q., and Leon-Garcia, A. sllm: Accelerating llm inference using semantic load balancing with shared memory data structures. In *2024 25th International Symposium on Quality Electronic Design (ISQED)*, pp. 1–6. IEEE, 2024.

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

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

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.

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

Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Heek, J., Xiao, K., Agrawal, S., and Dean, J. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5:606–624, 2023.

Qin, R., Li, Z., He, W., Cui, J., Ren, F., Zhang, M., Wu, Y., Zheng, W., and Xu, X. Mooncake: Trading more storage for less computation—a kvcache-centric architecture for serving llm chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pp. 155–170. USENIX Association, 2025.

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

Srivatsa, V., He, Z., Abhyankar, R., Li, D., and Zhang, Y. 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.

Ye, Z., Chen, L., Lai, R., Lin, W., Zhang, Y., Wang, S., Chen, T., Kasikci, B., Grover, V., Krishnamurthy, A., et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. *arXiv preprint arXiv:2501.01005*, 2025.

Zhao, W., Ren, X., Hessel, J., Cardie, C., Choi, Y., and Deng, Y. Wildchat: 1m chatgpt interaction logs in the wild. *arXiv preprint arXiv:2405.01470*, 2024.

Zheng, L., Yin, L., Xie, Z., Sun, C. L., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., et al. Sglang: Efficient execution of structured language model programs. *Advances in Neural Information Processing Systems*, 37:62557–62583, 2024.

Zhong, Y., Liu, S., Chen, J., Hu, J., Zhu, Y., Liu, X., Jin, X., and Zhang, H. {DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210, 2024.

Zhu, B., Sheng, Y., Zheng, L., Barrett, C., Jordan, M. I., and Jiao, J. On optimal caching and model multiplexing for large model inference. *arXiv preprint arXiv:2306.02003*, 2023.

Zhu, K., Zhao, Y., Zhao, L., Zuo, G., Gu, Y., Xie, D., Gao, Y., Xu, Q., Tang, T., Ye, Z., et al. Nanoflow: Towards optimal large language model serving throughput. *arXiv preprint arXiv:2408.12757*, 2024.

Appendix

A. Related Work

Classic Caching (Paging). The caching problem is well-studied from the competitive analysis perspective, with the hindsight optimal (Belady, 1966), optimal deterministic (Sleator & Tarjan, 1985), and randomized algorithms (Fiat et al., 1991) are known. Beyond worst-case competitive ratio, many works model structured request processes to capture the temporal locality normally observed in practice. These include access graph model (Borodin et al., 1995; Chrobak & Noga, 1999), independent reference model (Coffman & Denning, 1973; King, 1972; Dan & Towsley, 1990; Che et al., 2002), Shot Noise Model (Leonardi & Torrisi, 2015), LRU Stack and Working Set Model (Jain, 1990), and Inter-Reference Gap Model (Phalke & Gopinath, 1995). Our model differs by allowing new object arrivals with independent request processes per object. For broader overviews, see (Borodin & El-Yaniv, 2005), and for caching with predictions, see (Lykouris & Vassilvitskii, 2021; Mhaisen et al., 2022).

While most existing caching research focuses on maximizing hit rates, the closest related work to ours is (Berger et al., 2018), where they also use caching to reduce tail latency, but their design for web services does not apply directly to our single-machine LLM setting.

Prompt Caching. Prompt caching reuses precomputed KV states from chat history to reduce prefill computation. There have been many system-level works (Pope et al., 2023; Gim et al., 2024; Jin et al., 2024; Zhu et al., 2023; Lin et al., 2024; Srivatsa et al., 2024; Kwon et al., 2023; Zheng et al., 2024; Qin et al., 2025) and theory work (Zhu et al., 2023), but little is explored on optimizing tail latency through mathematical modeling.

B. Hindsight Optimal Policy for TEL

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. Let T denote the discrete time horizon, N denote the number of conversations, and C denote the cache capacity (in cache blocks). Define $A_i \subseteq [T]$ as the time steps when conversation i issues a request. For $t \in A_i$, let $q_{i,t}$ and $a_{i,t}$ denote the user prompt and model response lengths. The hindsight 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:

$$\min_{x_{i,t},u_{i,t}\in\mathbb{N}} \sum_{i\in[N]} \sum_{t\in\mathcal{A}_i} u_{i,t} \tag{1}$$

$$\text{s.t. } u_{i,t} \geq \sum_{j=1}^{t} q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - x_{i,t} - \xi, \qquad \forall i \in [N],\ t \in \mathcal{A}_i \quad \text{(slack variable)} \tag{2}$$

$$\sum_{i\in[N]} x_{i,t} \leq C, \qquad \forall t \in [T] \quad \text{(capacity constraint)} \tag{3}$$

$$x_{i,t+1} \leq \sum_{j=1}^{t} (q_{i,j} + a_{i,j}), \qquad \forall i \in [N],\ t \in \mathcal{A}_i,\ t < T \tag{4}$$

$$x_{i,t+1} \leq x_{i,t}, \qquad \forall i \in [N],\ t \notin \mathcal{A}_i,\ t < T \quad \text{(cannot conjure caches)} \tag{5}$$

Here $\sum_{j=1}^{t} q_{i,j} + \sum_{j=1}^{t-1} a_{i,j}$ is the total length of the chat history plus the prompt of the turn issued at step t, i.e., the number of blocks the model needs to process in the prefill stage, and $u_{i,t}$ describes the TEL objective as the optimal $u_{i,t} = (\sum_{j=1}^{t} q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - x_{i,t} - \xi)^+$. Constraint (4) requires one cannot cache more than total chat history; constraint (5) says a conversation's cache allocation can grow only when that conversation arrives.

Theorem B.1 (Hindsight Optimal Policy Structure)**.** *The hindsight optimal policy that minimizes TEL caps the number of KV cache blocks allocated to each conversation by TEL-safe budget; if additional evictions are required, it evicts cache*

blocks from conversations that will arrive furthest in the future. Specifically, let $\boldsymbol{x}^*$ denote an optimal solution to

$$\begin{aligned}
&\max_{x_{i,t}\in\mathbb{N}} \sum_{i\in[N]}\sum_{t\in\mathcal{A}_i} x_{i,t} \\
&\text{s.t. } (3),(4),(5) \\
&\quad x_{i,t} \le \left(\sum_{j=1}^{t} q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - \xi\right)^+, \forall i\in[N], t\in\mathcal{A}_i. \quad \textit{(cache just enough)}
\end{aligned}$$

Then $\boldsymbol{x}^*$ is an optimal solution to the TEL problem (1).

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.

Proof. Fix a feasible $\boldsymbol{x}$, the optimal $\boldsymbol{u}$ 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 $\boldsymbol{x}$.

We first show that the optimal solution to the optimization problem (1) must satisfies $x_{i,t} \le \left(\sum_{j=1}^{t} q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - \xi\right)^+$ for every $i\in[N], t\in\mathcal{A}_i$. Suppose not, i.e., the optimal $\boldsymbol{x}'$ to (1) satisfies $x'_{i,t} > \left(\sum_{j=1}^{t} q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - \xi\right)^+$ for some $i\in[N], t\in\mathcal{A}_i$. 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 objective value while releasing cache capacity that can be directed to other pages, which contradicts the optimality of $\boldsymbol{x}'$.

Thus the optimal solution satisfies

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

where the last equality holds as $x_{i,t} = 0$ for the case with $\sum_{j=1}^{t} q_{i,j} + \sum_{j=1}^{t-1} a_{i,j} - \xi < 0$.

Therefore, minimizing $\sum_{i\in[N]}\sum_{t\in\mathcal{A}_i} u_{i,t}$ is equivalent to maximizing $\sum_{i\in[N]}\sum_{t\in\mathcal{A}_i} x_{i,t}$ as the first term is a constant. □

C. Proof of Theorem 3.1

Proof. Let's fix an arbitrary sample path of user prompt traces $\{q_{i,t}\}$ and model response traces $\{a_{i,t}\}$. Let's assume the optional caching setting, as the proof for forced caching setting is similar and in fact simpler, as the caching policy doesn't consider the arriving conversation when making eviction decisions.

Suppose conversation θ arrives and the three policies start with the same cache states. For convenience let $L_\theta = L_\theta + Q + A$, and $\lambda_\theta = \lambda_{\mathsf{turn}}$ denote the length of conversation θ after service and the belief turn rate updated upon its arrival.

For conversations with length $L_i \le \xi - \bar{Q}$ (we assume $\xi > \bar{Q}$),

- Threshold LRU caches zero tokens by definition of the threshold,
- Tail-Optimized LRU also caches zero because $L_i + \bar{Q} - \xi \le 0$,
- LRU caches at least zero tokens.

By the next time such conversations arrive, the length of the chat history plus the user prompt is no longer than $L_i + \bar{Q} \le \xi$, thus the costs (i.e., number of uncached tokens) under all three policies are all zero.

For conversations with length $L_i \ge \xi - \bar{Q}$,

- Threshold LRU first stores the entire history (L_i tokens) then, if necessary, evicts according to LRU;
- Tail-Optimized LRU only stores $L_i + \bar{Q} - \xi$ and then, if necessary, also evicts according to LRU;

Hence, before the common LRU-eviction stage begins, Tail-Optimized LRU has already evicted more tokens that Threshold LRU. As the three policies start with the same cache state, the additional number of tokens to evicted according to LRU are ordered as

$$\mathsf{LRU} \ge \mathsf{ThLRU} \ge \mathsf{TLRU}.$$

Let X_i^{LRU} denote the number of blocks cached under LRU for conversation i.

- If $X_i^{\mathsf{LRU}} \ge L_i + \bar{Q} - \xi$, then we must have

  $$X_i^{\mathsf{ThLRU}} \ge X_i^{\mathsf{LRU}} \ge L_i + \bar{Q} - \xi = X_i^{\mathsf{TLRU}},$$

  and such an inequality holds until the next time conversation i arrives again, as the three policies use the same priority order at the LRU eviction phase, and the extra tokens evicted by Tail-Optimized LRU before using LRU is always no less than the other two. The costs under all three policies are all zero.

- If $X_i^{\mathsf{TLRU}} < L_i + \bar{Q} - \xi$, then we must have

  $$X_i^{\mathsf{TLRU}} \ge X_i^{\mathsf{ThLRU}} \ge X_i^{\mathsf{LRU}},$$

  due to the relative ordering of number to evicted according to LRU, and such an inequality holds until the next time conversation i arrives again as well. Thus, the costs under all three policies ordered as

  $$(L_i + Q_i - X_i^{\mathsf{TLRU}} - \xi)^+ \le (L_i + Q_i - X_i^{\mathsf{ThLRU}} - \xi)^+ \le (L_i + Q_i - X_i^{\mathsf{LRU}} - \xi)^+.$$

Because the comparison was carried out on an arbitrary sample path, the same ordering holds almost surely, which completes the proof. □

D. 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 Tail-Optimized LRU (Algorithm 1) and LRU as special cases.

Stochastic Conversation Model. Classic caching models fail to capture the nature of LLM workload: multi-turn conversations start, terminate, and grow in size 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.

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*. We capture this pattern by modeling the number of active conversations as a continuous-time birth-death process:

- New conversations are "born" at rate $\lambda_{\text{conv}} > 0$, and each active conversation "dies" at rate $\mu > 0$. We index conversations by their first-turn order, $i = 1, 2, \ldots$.
- 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.

For simplification, we assume that KV caches cannot be reused across different conversations. This assumption is also grounded by security and privacy concerns—e.g., vLLM implement cache isolation to prevent timing-based inference attacks.[2]

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 $(\boldsymbol{\lambda}(t), \boldsymbol{L}(t), \boldsymbol{X}(t))$, 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 $\boldsymbol{X}'$, which blocks to cache after serving each request, to minimize the Tail Excess Latency for M requests for arbitrary $M \in \mathbb{N}$.

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 D.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, $\boldsymbol{X}(t^-)$ denote the cache state *before* arrival, and $\boldsymbol{\lambda}(t^+), \boldsymbol{L}(t^+)$ denote the belief turn-arrival rates and chat history lengths updated *after* service ($\lambda_\theta(t^+) = \bar{\lambda}_\theta, L_\theta(t^+) = L_\theta(t^-) + Q + A$). ET-LRU chooses

$$\boldsymbol{X}^{\text{ETLRU}} \in \arg\min \sum_i \lambda_i(t^+) \cdot \mathbb{E}[(L_i(t^+) + Q_i - Y_i - \xi)^+] \tag{6}$$

$$\text{s.t. } Y_\theta \le L_\theta(t^+), \quad \text{(optional caching)} \tag{7}$$

$$Y_i \le X_i(t^-), \forall i \ne \theta, \quad \text{(cannot conjure caches)} \tag{8}$$

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

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 (7) implies that the decision-maker can cache at most the total chat history of conversation θ just served; constraint (8) says a conversation's cache allocation can grow only whenn it arrives.

[2] https://docs.vllm.ai/en/stable/design/v1/prefix_caching.html

Theorem D.2 (Optimality of Expected-Tail-Optimized LRU)**.** *Expected-Tail-Optimized LRU (Definition D.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 D.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 (6) reduces to $\max \sum_i \exp(-\mu(t - \text{last turn time}))Y_i$, thus ET-LRU reduces to LRU. Therefore, Theorem D.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 Algorithm 1 (with estimate $\hat{Q}$ replaced by deterministic Q). Theorem D.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.

Proof. Given system state $\boldsymbol{\lambda}, \boldsymbol{L}, \boldsymbol{X}$, 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

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

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

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

with $\mathsf{dim}(\boldsymbol{X}, \theta) = \mathsf{dim}(\boldsymbol{X}) + \mathbb{1}\{\theta > \mathsf{dim}(\boldsymbol{X})\}$, here $\mathsf{dim}(\boldsymbol{X})$ denotes the dimension of vector $\boldsymbol{X}$, and the dimension expands when a new conversation arrives; $\Phi(\boldsymbol{\lambda}, \theta, \tau)$ update the belief turn rate of conversation θ to $\bar{\lambda}_i$, then discount belief turn rates of all conversations by $\exp(-\mu\tau)$; $\Psi(\boldsymbol{L}, \theta, Q + A)$ updates the conversation length vector. Specifically, we increase the dimension of $\boldsymbol{L}$ 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 $\boldsymbol{\lambda}$, the conversation length vector as $\boldsymbol{L}$, and the cached token length vector as $\boldsymbol{X}$, the cost-to-go function is given by:

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

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

- $\Gamma(\boldsymbol{\lambda}, \theta)$ updates the return rate vector upon the arrival of conversation θ. Specifically, this operator changes the belief arrival rate of conversation θ to $\bar{\lambda}_i$.
- $\Phi(\boldsymbol{\lambda}, \tau)$ 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 $\boldsymbol{X}^{\mathsf{TLRU}}$, the cost will be higher. At last arrival M, $V_M(\boldsymbol{\lambda}, \boldsymbol{L}, \boldsymbol{X}, \theta, Q_\theta, A_\theta) = (L_\theta + Q_\theta - X_\theta - \xi)^+$ 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

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

Then we need to prove

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

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 $\boldsymbol{\lambda}$ 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 $\tilde{p}_j = \tilde{\lambda}_j/(\lambda_{\mathsf{conv}} + \sum_i \tilde{\lambda}_i)$ denote the probability that conversation j returns at the next arrival, and $\tilde{p}_{\mathsf{dim}(\boldsymbol{X}(1))+1} = \lambda_{\mathsf{conv}}/(\lambda_{\mathsf{conv}} + \sum_i \tilde{\lambda}_i)$ 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 $i(1), i(2), \ldots$ 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 $\boldsymbol{X}(1)$ denote the cache state that evicts a token from conversation $i(1)$, and $\boldsymbol{X}(k)$ 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}, \boldsymbol{X}(1)) \\
&= \sum_{i \in \mathsf{dim}(\boldsymbol{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_{\boldsymbol{X}'(1) \in \mathcal{X}(\boldsymbol{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)}), \boldsymbol{X}'(1)) \right] \\
&\quad + \tilde{p}_{i(k)} \mathbb{E}_{Q_{i(k)}, A_{i(k)}} \left[\min_{\boldsymbol{X}'(1) \in \mathcal{X}(\boldsymbol{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)}), \boldsymbol{X}'(1)) \right] \\
&\quad + \sum_{i \neq i(1), i(k)} \mathbb{E}_{Q_i, A_i} \left[\min_{\boldsymbol{X}'(1) \in \mathcal{X}(\boldsymbol{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), \boldsymbol{X}'(1)) \right] \\
&\leq \sum_{i \in \mathsf{dim}(\boldsymbol{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_{\boldsymbol{X}'(k) \in \mathcal{X}(\boldsymbol{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)}), \boldsymbol{X}'(k)) \right] \\
&\quad + \tilde{p}_{i(k)} \mathbb{E}_{Q_{i(k)}, A_{i(k)}} \left[\min_{\boldsymbol{X}'(k) \in \mathcal{X}(\boldsymbol{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)}), \boldsymbol{X}'(k)) \right] \\
&\quad + \sum_{i \neq i(1), i(k)} \mathbb{E}_{Q_i, A_i} \left[\min_{\boldsymbol{X}'(k) \in \mathcal{X}(\boldsymbol{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), \boldsymbol{X}'(k)) \right] \\
&= J_{k+1}(\tilde{\boldsymbol{\lambda}}, \tilde{L}, \boldsymbol{X}(k)),
\end{aligned}$$

where the inequality holds as

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

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

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

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

- by induction hypothesis, the optimal $\boldsymbol{X}'(1)^*$ and $\boldsymbol{X}'(k)^*$ are given by the optimization problem (6). Fix a user prompt length Q_i and a model response length A_i and we compare the cost-to-do under $\boldsymbol{X}(1)$ and $\boldsymbol{X}(k)$.
  - if conversation $i(1)$ arrives next, then one need to evict one more token from $\boldsymbol{X}(1)$ than from $\boldsymbol{X}(k)$. Suppose the extra token evicted from $\boldsymbol{X}(1)$ is from conversation $i(k)$, then $\boldsymbol{X}'(1)^*=\boldsymbol{X}'(k)^*$. If not, then this means the extra token evicted is from another conversation with better ranking criterion, thus we have

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

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

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

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

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

Therefore, by induction, the result holds for all $k\ge 1$. □

Greedy Implementation. The optimization problem (6) need not be solved explicitly as a token-by-token greedy procedure suffices. At a high-level, the algorithm ranks each token by arrival rates weighted by its counterfactual cost, i.e., the cost increase when we evict this token.

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

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

In the implementation of the algorithm, one can use min-heap to process which token to evict using the ranking criterion. The computational complexity of the algorithm 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 Algorithm 2 indeed returns a cache state that is an optimal solution to the optimization problem (6).

Lemma D.3. *Algorithm 2 returns an optimal solution to the optimization problem* (6).

Proof. We prove by contradiction. Note that it is possible for the algorithm to return multiple optimal solutions, and it is also possible for the optimization problem (6) to have multiple optimal solutions. Suppose not, then the two set of solutions do not intersect. Let $\boldsymbol{X}^*$ denote one optimal solution. Then there must exist two conversations i, j such that $X_j^*\ge 1$ and

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

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 $\leftarrow 0$
$L_\theta \leftarrow L_\theta + Q + A$ `// Update system state for arriving conversation` θ
$X_\theta \leftarrow L_\theta$
$\lambda_\theta \leftarrow \bar{\lambda}_\theta$
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 $\leftarrow$ evicted $+ 1$
  $v_j \leftarrow \lambda_j \cdot \mathbb{P}(L_j + Q_j - \xi \geq X_j)$ `// Update ranking criterion`
return $\{X_i\}$.

Then we can construct another solution $\boldsymbol{X}'$ 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 $\boldsymbol{X}'$ and $\boldsymbol{X}^*$ is given by

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

which contradicts the optimality of $\boldsymbol{X}^*$. □

E. 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 algorithm to evict more tokens if needed.

Hindsight optimal policy. To model forced caching, we replace optional caching constraint (4) with

$$x_{i,t+1} = \sum_{j=1}^{t} (q_{i,j} + a_{i,j}), \forall i \in [N], t \in \mathcal{A}_i \text{ and } t < T, \tag{9}$$

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

No-Worse-Than-LRU Guarantee Theorem 3.1 continues to hold under forced caching.

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

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

Theorem D.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 Algorithm 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.

F. Additional Figures

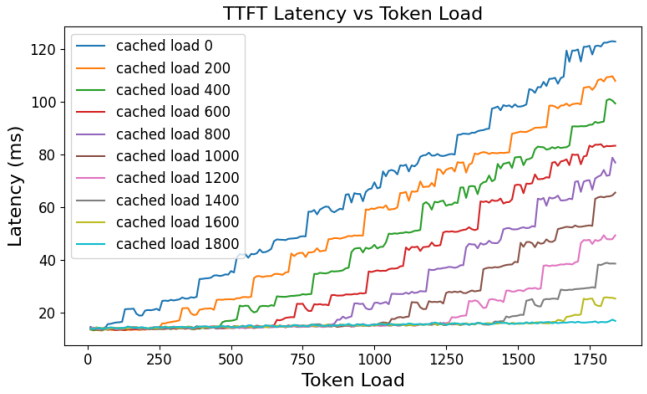

Figure 3: Experimental results demonstrating the linear fit. We used Vicuna-7B with vLLM's prefix-caching enabled on a Colab A100 GPU. The plot shows TTFT latency as a function of total prompt length and cached-prefix size.

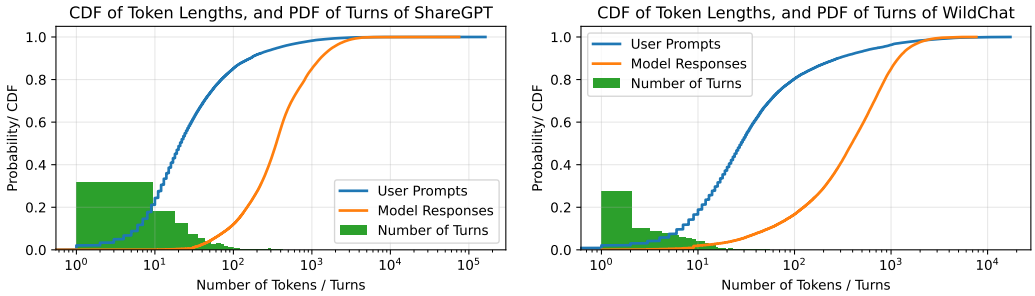

Figure 4: Distributions of turns and tokens of ShareGPT (Contributors, 2025) and WildChat (Zhao et al., 2024) datasets.

G. Additional Experiment Results

Table 3: Relative latency improvement of T-LRU over Threshold-LRU with various ξ

	ξ = 50ms		ξ = 100ms		ξ = 200ms		ξ = 300ms		ξ = 500ms		
Capacity	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%

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

In this section, we provide experiment results of Tail-Optimized Baledy (the hindsight-optimal policy for Tail Excess Latency), serving as the best possible benchmark, and two variants of T-LRU with future knowledge. *End-Aware T-LRU* knows whether a conversation will continue (return) or not, but does not know the length of future prompts. This variant

evicts all blocks from a conversation if it terminates. *Length-Aware T-LRU* knows both whether a conversation will continue or not AND the exact length of the next user prompt. This variant uses the exact user prompt to perform proactive trimming, while T-LRU uses the empirical average as the next prompt length. These variants create a predictability spectrum that helps us quantify maximum potential benefits from different levels of prediction.

We highlight two observations from Figure 5. First, tail improvement is achieved with a modest cost to the median. T-LRU (blue squares) consistently beats LRU and Threshold-LRU at the tail (around 300 ms), but its median latency is slightly higher (around 40 ms). The trade-off is expected and in fact intended: reduce the worst-case delays by sacrificing a few milliseconds on typical requests.

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 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.

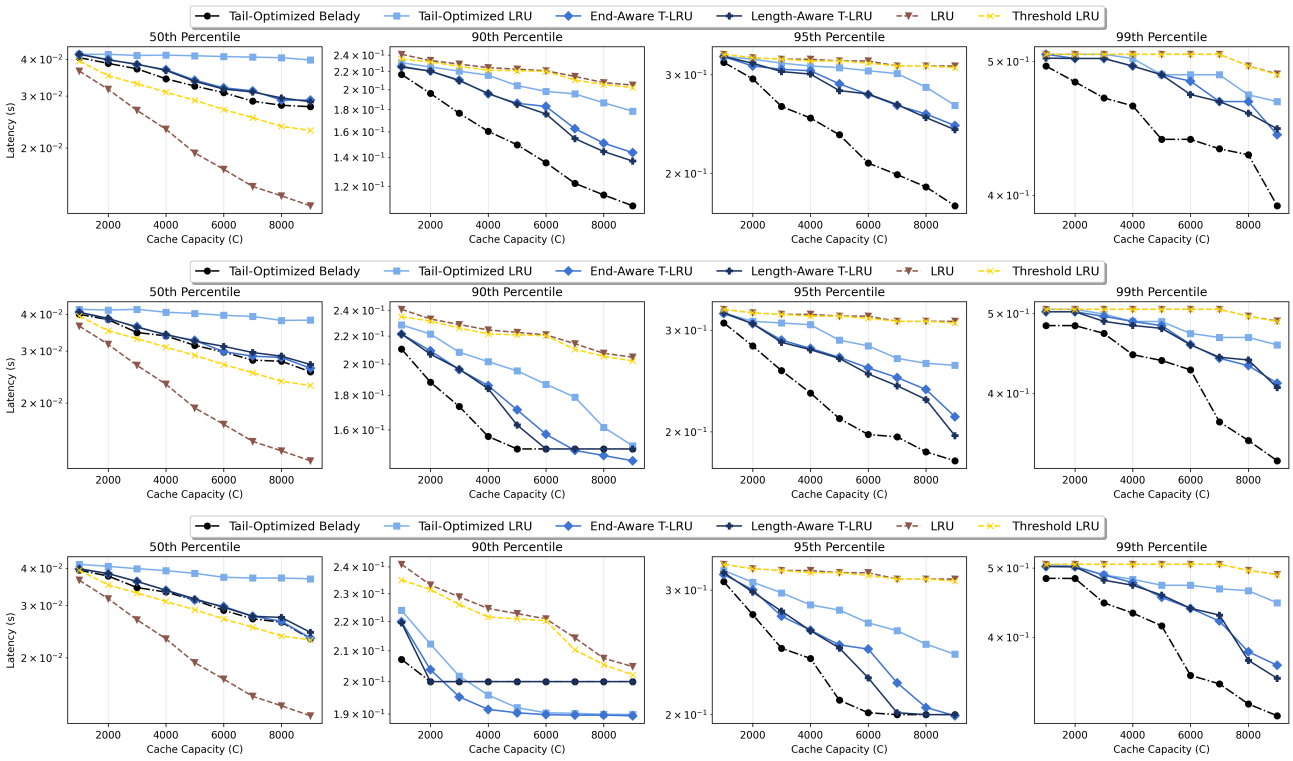

Figure 5: Latency results for various settings (threshold latency ξ = 100, 200, 300 ms) from top to bottom panels.

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 D. 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 1–3) 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 (ShareGPT)

	ξ = 50ms		ξ = 100ms		ξ = 200ms		ξ = 300ms		ξ = 500ms	
Capacity	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 (ShareGPT)

	ξ = 50ms		ξ = 100ms		ξ = 200ms		ξ = 300ms		ξ = 500ms	
Capacity	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 is near the desired percentile; extremely high ξ 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 $>$ 200ms (ShareGPT)

Capacity	ξ = 50ms		ξ = 100ms		ξ = 150ms		ξ = 200ms		ξ = 500ms	
	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.

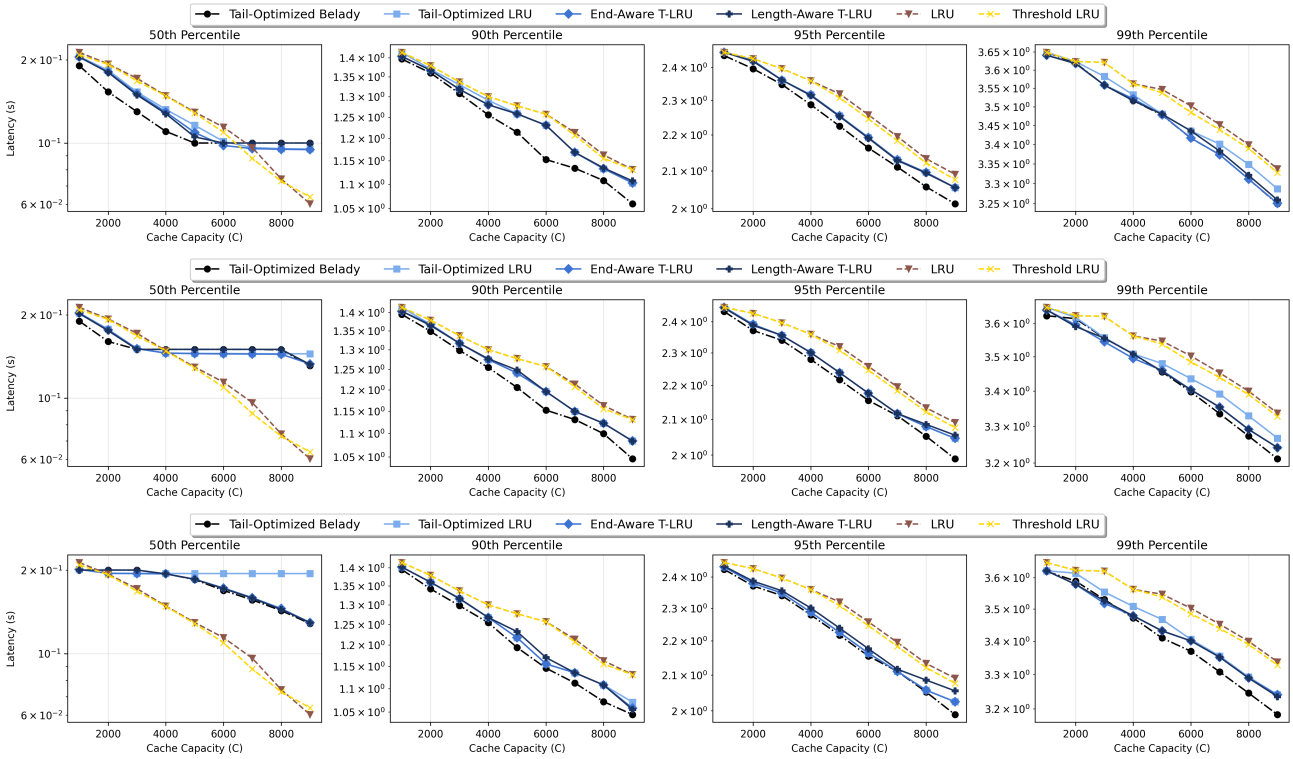

Figure 6: Latency results for various settings (threshold latency ξ = 100, 200, 300 ms) from top to bottom panels (ShareGPT)