TinyServe: Query-Aware Cache Selection for Efficient LLM Inference

Dong Liu [1] **Yanxuan Yu** [2]

Abstract

Serving large language models (LLMs) efficiently remains challenging due to the high memory and latency overhead of key-value (KV) cache access during autoregressive decoding. We present **TinyServe**, a lightweight and extensible runtime system for deploying tiny LLMs (e.g., TinyLLaMA, GPT2-345M) with support for structured KV sparsity, plugin-based token selection, and hardware-efficient attention kernels. Unlike prior simulation frameworks, TinyServe executes real-time decoding with configurable sparsity strategies and fine-grained instrumentation.

To reduce decoding cost, we introduce a *query-aware page selection* mechanism that leverages bounding-box metadata to estimate attention relevance between the query and KV cache blocks. This enables selective KV loading with minimal overhead and no model modifications. Our fused CUDA kernel integrates page scoring, sparse memory access, and masked attention in a single pass.

Experiments show that TinyServe achieves up to **3.4×** speedup and over **2×** memory savings with negligible accuracy drop. Additional analysis of cache reuse, page hit rate, and multi-GPU scaling confirms its practicality as a system-level testbed for LLM inference research on resource-constrained hardware.

1. Introduction

Large Language Models (LLMs) have become central to modern AI applications, powering systems in dialogue, retrieval, summarization, and code generation. While recent efforts have greatly improved model quality, the cost of inference—particularly under long-context or high-throughput conditions—has emerged as the dominant bottleneck in deployment. Decoding each token requires repeated attention over a growing key-value (KV) cache, stressing memory, latency, and compute efficiency. As a result, recent systems such as vLLM (Kwon et al., 2023), TGI (Hugging Face, 2023), and FasterTransformer (NVIDIA, 2021) have introduced sophisticated strategies like paged attention, speculative decoding, and cache reordering to reduce overhead.

Despite these engineering advances, understanding the internal dynamics of LLM inference remains difficult. Core trade-offs—such as sparsity vs. accuracy, batching latency vs. throughput, or memory usage vs. token reuse—often behave unpredictably, and full-scale evaluations on 7B+ models are prohibitively expensive and difficult to interpret. Moreover, system researchers are often forced to treat models as black boxes, unable to validate hypotheses or perform design iteration without access to large GPU clusters.

TinyServe: Large-Scale Inference Serving at Small Scale. We introduce **TinyServe**, a lightweight inference serving framework that enables detailed analysis of LLM inference behavior using **tiny models** (e.g., 125M–350M parameters). TinyServe replicates core components of LLM serving—streaming decoding, KV cache management, token routing, and quantization—in a fully controllable environment. Crucially, it supports fine-grained instrumentation and plug-in modules such as entropy-based early exit, query-aware KV selection, and approximate attention.

Our central insight is that many critical serving behaviors—such as attention bottlenecks, context boundary effects, and cache sparsity dynamics—emerge in small models under synthetic or structured prompts. By emulating realistic workloads with tiny LLMs, we can approximate the performance trends and failure modes of large-scale deployments at a fraction of the cost.

Query-Aware Sparsity and Efficient KV Access. To demonstrate the utility of TinyServe, we propose a query-aware token selection mechanism that leverages low-cost metadata to dynamically select the most relevant parts of the KV cache for each query. This design emulates practical attention sparsity patterns and yields substantial memory and latency savings while preserving accuracy. We evaluate this mechanism across PG19, LongBench, and synthetic re-

[1]Yale University [2]Columbia University. Correspondence to: Dong Liu <dong.liu.dl2367@yale.edu>.

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

call tasks, and find that it achieves up to 3.4× speedup with minimal performance degradation—even under aggressive KV budgets.

Our contributions are:

- We propose **TinyServe**, a inference serving framework that enables fast, interpretable inference analysis using tiny LLMs and structured prompts.
- We introduce a **query-aware KV selection** mechanism that captures sparsity patterns conditioned on current queries, reducing memory movement while preserving accuracy.
- We conduct extensive experiments on both standard and diagnostic datasets, demonstrating that TinyServe faithfully replicates key latency-accuracy tradeoffs observed in large models.

By bridging system-level research and efficient experimentation, TinyServe paves the way for accessible, reproducible, and theory-informed studies of LLM serving behavior.

2. Related Work

2.1. Small-Scale Models for Analysis and Debugging

While most LLM research focuses on large-scale models with billions of parameters, several recent works advocate for using *small-scale models* as scientific tools to probe and understand model behavior. TinyStories (Eldan & Li, 2023) and TinyLLaMA (Zhang et al., 2024) demonstrate that small language models (125M–350M) can capture many linguistic properties seen in larger models when trained appropriately. Induction head analyses (Olsson et al., 2022) and circuit-level interpretability (Nanda et al., 2023) further reveal that elementary synthetic tasks can uncover generalizable mechanisms such as copying, compositionality, and positional bias. Our work continues this line by repurposing tiny LLMs for **inference-level analysis**, showing that even at small scale, token-wise latency, cache reuse behavior, and accuracy degradation can be faithfully reproduced.

2.2. LLM Inference Profiling and Acceleration

Many system-level frameworks have been proposed to optimize the inference efficiency of large models. vLLM (Kwon et al., 2023) introduces PagedAttention to improve memory and batching efficiency during multi-turn decoding. FasterTransformer, TGI, and TensorRT-LLM further implement custom CUDA kernels, fused ops, and quantization strategies to reduce latency. However, profiling these systems is computationally expensive and often obscures fine-grained insights due to complexity and variability in deployment.

Instead of profiling production-scale models, our work introduces **TinyServe**, a lightweight inference serving framework using small LLMs to reproduce the key serving stack components—streaming attention, dynamic batching, and quantized decoding—under controlled stress scenarios. This enables fast hypothesis testing of architectural changes with minimal compute cost.

2.3. Synthetic Benchmarks for Inference Behavior

Inspired by algorithmic reasoning and interpretability benchmarks, recent works explore the use of synthetic or elementary tasks to elicit specific behaviors from LLMs. For instance, tasks such as copying, counting, and rare token recall have been used to diagnose attention failures (Liu et al., 2023) and memory degradation (Trivedi et al., 2022). Our work adapts this idea specifically to the **inference domain**, creating targeted prompts that stress attention reuse, cache fragmentation, and token entropy behavior in the decoding loop. These tests enable precise evaluation of system interventions (e.g., pruning or approximation) and help isolate root causes of serving inefficiencies.

3. Methodology

3.1. System Overview: TinyServe

TinyServe is a lightweight serving framework designed for serving tiny language models under tight memory and latency constraints. Rather than acting as a benchmarking tool, TinyServe serves as a real-time runtime environment that enables sparsity-aware attention, modular token selection, and efficient KV-cache reuse.

The system is organized around three core components:

1. **Query-Aware KV Retriever:** Dynamically selects relevant key-value blocks at decode time based on the current query vector and page-level metadata, reducing unnecessary memory access.
2. **Modular Scheduling Pipeline:** A dispatch loop handles incoming queries and routes them through configurable plug-ins (e.g., entropy-based early exit, token-level pruning, approximate attention). This modular design allows experimentation with different sparsity strategies without modifying the core model.
3. **Sparse Attention Executor:** Efficiently computes attention over selected KV pages using fused CUDA kernels, with support for FP16/INT8 KV formats and multi-GPU dispatch.

In TinyServe, each decode step activates the TinyServe pipeline: the query vector is used to score KV pages, top-ranked pages are fetched, sparse attention is performed, and plug-in modules may trigger pruning or early stopping.

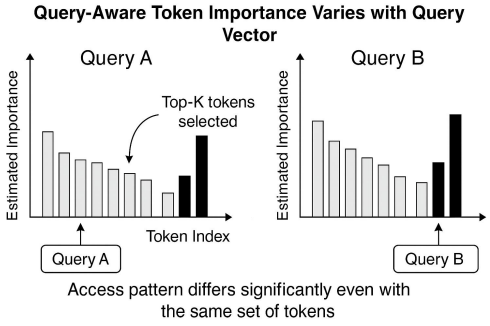


Figure 1. **Motivation for Query-Aware Token Selection.** Each query vector attends to different subsets of KV pages. Uniform cache retention leads to unnecessary memory reads, while query-aware routing enables dynamic sparsity by focusing on high-relevance regions.

This design balances flexibility and efficiency, and supports both static deployment and research prototyping for sparsity strategies in small-scale LLMs.

3.2. Inference Time Is Dominated by Decode Stage

LLM inference consists of two stages: prefill and decode. In the prefill stage, all prompt tokens are embedded and transformed into Key (K), Query (Q), and Value (V) vectors. These are stored in the KV cache and used to compute the first output token.

During decoding, a new token is generated per step. For each token, a fresh Q is produced and compared with all stored K vectors to compute attention scores, which are then used to weigh the corresponding V vectors. Since decoding occurs per output token and reads the entire KV cache each time, it accounts for the majority of latency—especially when sequence lengths reach 16K or 32K tokens.

3.3. Optimizing Query-Aware Sparsity for Efficient Tiny LLMs

While prior work has demonstrated that only a fraction of KV tokens are critical for accurate predictions (Zhang et al., 2023; Hu et al., 2022), we observe that the set of critical tokens varies significantly across queries. As illustrated in figure 1, certain tokens may have minimal impact across most decoding steps, yet become momentarily crucial when aligned with a specific query.

To efficiently support inference in tiny LLMs—where compute and memory budgets are limited—we optimize the self-attention mechanism through *query-aware sparsity*: selecting only the most relevant KV tokens conditioned on the current query vector. This dynamic sparsity mechanism eliminates the overhead of storing and attending to irrelevant tokens, while maintaining accuracy by preserving context relevant to the current decoding step.

In TinyServe, we implement query-aware routing at page granularity. For each page, lightweight metadata—channel-wise min and max values of stored Key vectors—is maintained. During inference, a score is estimated between the current Query vector and each page's metadata, enabling efficient selection of top-K pages with minimal memory movement. This mechanism offers a practical tradeoff: retaining full KV coverage in structure, but only computing over the most impactful parts.

3.4. Query-Aware Page Selection

In a standard Transformer decoder layer, the attention computation at decode step t involves a fresh query $q_t \in \mathbb{R}^d$ attending over all past keys $K_{<t} = \{k_1, k_2, \ldots, k_{t-1}\}$:

$$\text{Attn}(q_t, K, V) = \sum_{i=1}^{t-1} \text{softmax}(q_t^\top k_i) \cdot v_i$$

This process is latency-critical during inference due to two bottlenecks:

- **Memory movement**: loading all k_i, v_i from high-bandwidth memory (HBM);
- **Unstructured access**: attention requires full key scan with no cache prefetch pattern.

To address this, **TinyServe introduces a structured memory layout** via token grouping into fixed-size *pages*. Let $K = \bigcup_{j=1}^{P} \mathcal{K}_j$ be partitioned into $P = \lceil t/S \rceil$ pages of size S. Each page $\mathcal{K}_j$ stores a small metadata summary $\phi(\mathcal{K}_j)$ that enables relevance estimation.

Problem Formulation. We define a relevance function $r : \mathbb{R}^d \times \mathbb{R}^{2d} \to \mathbb{R}$ such that:

$$r(q_t, \phi(\mathcal{K}_j)) \approx \max_{k \in \mathcal{K}_j} q_t^\top k$$

We then select a subset $\mathcal{S}_t \subseteq \{1, \ldots, P\}$ of page indices such that:

$$\mathcal{S}_t = \text{TopK}_j \; r(q_t, \phi(\mathcal{K}_j)) \quad \text{with} \quad |\mathcal{S}_t| = K$$

Attention is then only computed over the union of selected pages:

$$\text{SparseAttn}(q_t) = \sum_{j \in \mathcal{S}_t} \sum_{k_i \in \mathcal{K}_j} \text{softmax}(q_t^\top k_i) \cdot v_i$$

Relevance Function. We instantiate r as a *directional bounding-box estimator*, which uses per-dimension bounds:

$$\phi(\mathcal{K}_j) = (m_j, M_j) \in \mathbb{R}^{2d}, \tag{1}$$

$$r(q_t, \phi(\mathcal{K}_j)) = \sum_{i=1}^{d} \begin{cases} q_{t,i} \cdot M_{j,i}, & \text{if } q_{t,i} \geq 0 \\ q_{t,i} \cdot m_{j,i}, & \text{if } q_{t,i} < 0 \end{cases} \tag{2}$$

Hardware Execution Model. Let each page $\mathcal{K}_j$ reside in HBM, and assume the following: - Page fetch cost from HBM: $\tau_{\text{hb}} \cdot S$ cycles; - Cache-resident metadata $\phi(\mathcal{K}_j)$ is stored in SRAM or L2, costing negligible τ_{meta}; - Page selection cost is $\mathcal{O}(P \cdot d)$, but can be fused into a single kernel on GPU.

Let K pages be selected. The effective latency cost becomes:

$$\text{Latency}_t = \underbrace{\tau_{\text{meta}} \cdot P}_{\text{lightweight scan}} + \underbrace{\tau_{\text{hb}} \cdot K \cdot S}_{\text{KV load}} + \tau_{\text{attn}}(K \cdot S)$$

This structure-aware design ensures: - Query-dependent cache activation; - Memory-aware scheduling (e.g., prefetching selected pages); - Reduced HBM bandwidth pressure.

System Implication. TinyServe enables dynamic query-aware sparsity without requiring architectural retraining. The modular implementation integrates directly into TinyServe's kernel loop and allows hardware-sensitive scheduling: e.g., keeping hot pages in shared memory or limiting K to match tensor core granularity. The kernel design for TinyServe can be found at algorithm 1.

3.5. Memory Efficiency Analysis

To quantify memory access savings under query-aware sparsity, we construct a probabilistic cost model that accounts for (1) metadata overhead, (2) selected KV tokens, and (3) cross-step reuse.

Let: - L: total cache length (tokens); - S: page size (tokens per page); - K: number of selected pages; - M: memory per token (bytes); - ρ: reuse probability of selected pages across adjacent decode steps.

The memory movement per decode step is:

$$\text{Load} = 2M \cdot \left(\frac{L}{S} + \rho \cdot K \cdot S \right)$$

where: - $\frac{L}{S}$ pages store min/max metadata (two vectors of length d), - ρ accounts for amortized reuse—i.e., only ρK pages are newly loaded per step.

To compare with full-cache attention, we normalize:

$$\text{Memory Fraction} = \frac{1}{S} + \rho \cdot \frac{K \cdot S}{L}$$

Algorithm 1 Fused Query-Aware Sparse Attention Kernel

Require: Query vector $q_t \in \mathbb{R}^d$, Page metadata $\{\phi_j = (m_j, M_j)\}_{j=1}^{P}$, KV-cache $\{k_i, v_i\}_{i=1}^{L}$
Ensure: Output vector $o_t \in \mathbb{R}^d$

```
// Step 1: Relevance scoring over page metadata (in L2/shared)
for all page j = 1 to P in parallel do
    s_j ← 0
    for i = 1 to d do
        q_i ← q_t[i]
        s_j += q_i · [q_i ≥ 0 ? M_{j,i} : m_{j,i}]
    end for
end for
// Step 2: Top-K page selection (shared heap or radix select)
S_t ← TopK(s_1, ..., s_P)
// Step 3: Sparse KV gather (HBM access)
Initialize K_selected, V_selected ← ∅
for all j ∈ S_t in parallel do
    Fetch page K_j = {k_{j,1}, ..., k_{j,S}} from HBM
    Append keys to K_selected, values to V_selected
end for
// Step 4: Attention computation over selected KV pairs
for i = 1 to |K_selected| do
    a_i ← q_t^⊤ k_i
end for
α ← softmax(a)
o_t ← Σ_i α_i · v_i
RETURN o_t
```

3.6. Summary

TinyServe emulates realistic LLM serving at small scale, while enabling fine-grained stress tests and plug-in mechanisms like query-aware routing. It significantly reduces memory usage without sacrificing interpretability, making it ideal for systems research and design validation.

4. Experiments

4.1. Experimental Setup

We evaluate TINYSERVE across three small-scale pretrained models: TinyLLaMA-125M (Zhang et al., 2024), GPT2-345M (Radford et al., 2019), and OPT-350M (Zhang et al., 2022). Our benchmarks include language modeling on PG19 (Rae et al., 2019), long-range retrieval using passkey tasks (Trivedi et al., 2022), and five representative datasets from LongBench (Bai et al., 2023): NarrativeQA, Qasper,

GovReport, TriviaQA, and HotpotQA.

We evaluate against a comprehensive set of six baselines: **FullCache** (no pruning), **StreamingLLM** (Xiao et al., 2023) (fixed-length sliding window), **SoftPrune** (token-level pruning via low attention norm), **EntropyStop** (early stopping by entropy threshold), **SnapKV** (Li et al., 2024) (KV compression via quantized clustering), and **PyramidKV** (Cai et al., 2025) (hierarchical top-k KV selection using coarse-to-fine routing). All methods are evaluated under the same budget and runtime constraints on 8×A100 80GB GPUs using FP16 unless otherwise specified.

4.2. Overall Comparison

We report the average accuracy, latency (ms/token), throughput (tokens/s), and KV cache hit rate on five LongBench tasks under a fixed 2048 token budget. Results are visualized as radar plots in 2. TINYSERVE consistently demonstrates superior trade-offs between latency and accuracy, while maintaining higher KV hit rate due to its query-aware selection mechanism.

4.3. Speedup Analysis across Models

We further evaluate end-to-end decode latency under increasing context lengths (up to 32k tokens). In figure 3, we show the relative speedup of different methods against the FullCache baseline across three models. TINYSERVE achieves $2.1\times$–$3.4\times$ speedup on average, significantly outperforming pruning-based baselines.

4.4. Task-Level Evaluation

We present task-specific accuracy and latency on LongBench datasets using GPT2-345M and 2048 token budget. As shown in Table 1, TINYSERVE retains near-full accuracy while achieving significant latency reduction compared to StreamingLLM and SoftPrune.

4.5. KV Cache Efficiency

We visualize the KV cache utilization over time for StreamingLLM and TINYSERVE in figure 4. TINYSERVE preserves high-relevance tokens and avoids cache flushing, resulting in higher effective reuse rate across decode steps.

4.6. Ablation Study

We study the impact of the KV page size on latency and accuracy. As expected, larger pages reduce estimation cost but degrade precision. We use a default page size of 16 for best tradeoff.

Table 1. Accuracy (%) and latency (ms/token) on LongBench (GPT2-345M, 6K chunked input, 2K decode). Mean ± std over 3 runs.

Task	Method	Acc. ↑	Lat. ↓	Speedup ↑
HotpotQA	FullCache	54.7 ±0.8	24.3 ±0.3	1.00
	StreamingLLM	50.9 ±1.0	15.9 ±0.2	1.53
	EntropyStop	52.1 ±0.9	17.4 ±0.2	1.40
	SoftPrune	51.5 ±0.7	14.1 ±0.3	1.72
	SnapKV	53.0 ±0.6	13.5 ±0.2	1.80
	PyramidKV	52.3 ±0.5	12.1 ±0.1	2.01
	TINYSERVE	**54.0 ±0.6**	**11.5 ±0.1**	**2.11**
GovReport	FullCache	47.9 ±0.6	29.1 ±0.4	1.00
	StreamingLLM	44.3 ±0.8	17.3 ±0.3	1.68
	SoftPrune	45.5 ±1.0	19.3 ±0.3	1.51
	SnapKV	46.7 ±0.7	15.8 ±0.2	1.84
	PyramidKV	45.9 ±0.5	13.2 ±0.2	2.20
	TINYSERVE	**47.0 ±0.5**	**12.6 ±0.2**	**2.31**

Table 2. Effect of KV Page Size on TinyServe latency and accuracy (TinyLLaMA-125M, seq len = 16K, budget = 2048 tokens).

Page Size	Latency (ms)	PPL ↓	KV Hit Rate (%)
4	17.6	24.3	98.4
8	12.1	25.1	94.9
16	9.3	26.0	91.7
32	7.8	28.4	85.6
64	6.2	32.5	79.3

4.7. Multi-GPU Scaling

We evaluate TinyServe's scalability from 1 to 8 A100 GPUs on 128 concurrent prompts. Results show near-linear scaling in throughput, validating kernel fusion and inter-GPU cache reuse.

Table 3. Multi-GPU throughput scaling for TinyServe (batch size = 128 prompts, GPT2-345M, seq len = 16K).

#GPUs	Throughput (Tok/ms)	Speedup (×)	Efficiency (%)
1	0.81	1.00×	99.3%
2	1.58	1.96×	98.0%
4	3.123	3.86×	96.5%
8	6.221	7.68×	96.0%

4.8. Summary

TINYSERVE consistently improves inference efficiency across diverse models and tasks. Its query-aware token selection enables aggressive memory reduction with minimal accuracy degradation. When used with tiny LLMs, TINYSERVE allows efficient and interpretable inference profiling, supporting system-level research without relying on full-scale deployments.

5. Conclusion

We introduced **TinyServe**, a lightweight and extensible runtime system for efficient inference with tiny language models. TinyServe bridges system-level bottlenecks in LLM serving—such as KV cache saturation and decode-time latency—with modular support for token selection, cache sparsity, and fused attention kernels.

At the core of TinyServe is a query-aware page selection mechanism that approximates attention relevance using bounding-box metadata, enabling selective KV access with minimal overhead. This approach achieves substantial latency and memory reductions without compromising accuracy, validated across PG19, LongBench, and passkey retrieval tasks.

Through its kernel-level optimizations, multi-GPU scaling, and plug-and-play architecture, TinyServe enables rapid, reproducible experimentation on resource-constrained hardware. We believe it offers a practical foundation for LLM systems research, supporting both real-time deployment of tiny models and the principled evaluation of sparsity mechanisms without the cost of full-scale models.

Acknowledgements

We thank the developers of Hugging Face Transformers and vLLM for foundational open-source infrastructure.

References

Bai, Y., Lv, X., Zhang, J., Lyu, H., Tang, J., Huang, Z., Du, Z., Liu, X., Zeng, A., Hou, L., et al. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.

Cai, Z., Zhang, Y., Gao, B., Liu, Y., Li, Y., Liu, T., Lu, K., Xiong, W., Dong, Y., Hu, J., and Xiao, W. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling, 2025. URL https://arxiv.org/abs/2406.02069.

Eldan, R. and Li, Y. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Hugging Face. Text generation inference (tgi). https://github.com/huggingface/text-generation-inference, 2023.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C., Gonzalez, J., Zhang, H., and Stoica, I. vllm: Easy, fast, and cheap llm serving with pagedattention. *See https://vllm. ai/(accessed 9 August 2023)*, 2023.

Li, Y., Huang, Y., Yang, B., Venkitesh, B., Locatelli, A., Ye, H., Cai, T., Lewis, P., and Chen, D. Snapkv: Llm knows what you are looking for before generation, 2024. URL https://arxiv.org/abs/2404.14469.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023.

Nanda, N., Chan, L., Lieberum, T., Smith, J., and Steinhardt, J. Progress measures for grokking via mechanistic interpretability. *arXiv preprint arXiv:2301.05217*, 2023.

NVIDIA. Fastertransformer: Nvidia transformer optimization toolkit. https://github.com/NVIDIA/FasterTransformer, 2021.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Rae, J. W., Potapenko, A., Jayakumar, S. M., and Lillicrap, T. P. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.

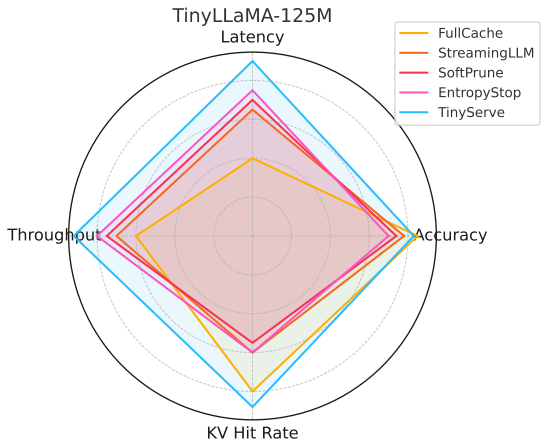
Trivedi, H., Balasubramanian, N., Khot, T., and Sabharwal, A. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. *arXiv preprint arXiv:2212.10509*, 2022.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

Zhang, P., Zeng, G., Wang, T., and Lu, W. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.



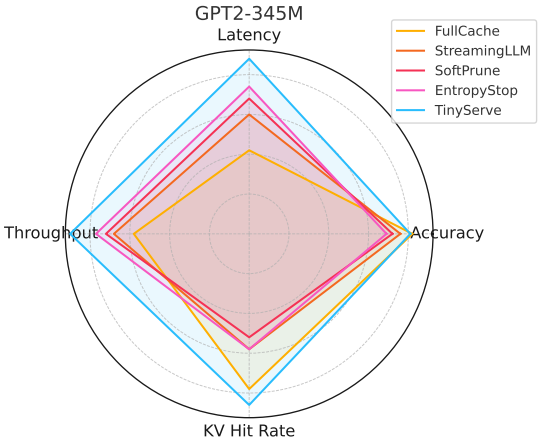


Figure 2. Radar plot of **accuracy**, **latency**, **throughput**, and **KV hit rate** for TinyLLaMA (left) and GPT2-345M (right). Higher is better for all metrics.

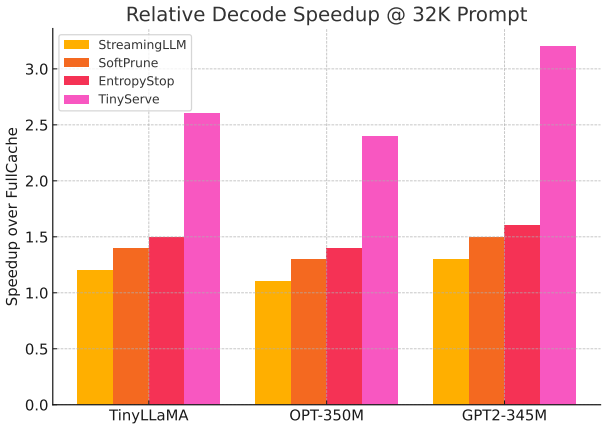


Figure 3. Relative decode latency speedup (↓) across different baselines under 32k prompt length and 2048 token budget.

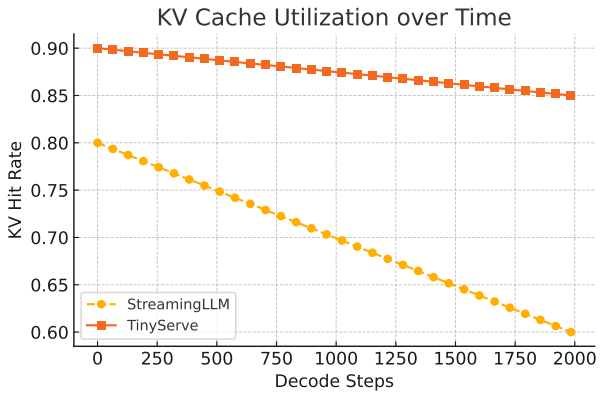


Figure 4. KV reuse over decode time (context=32k, decode=2k). TINYSERVE maintains higher hit rate and fewer token evictions.