REFRAME: ELASTIC EXPERT–KV MEMORY SHARING FOR MIXTURE-OF-EXPERTS SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

In large mixture-of-experts serving systems with limited GPU memory, expert weights and key–value (KV) state compete for capacity, yet most existing systems statically partition this memory between them. Since only a subset of resident experts is activated per token, substantial expert memory can remain underutilized. Therefore, static GPU memory partitioning prevents the underutilized memory from being used by the KV cache, causing requests to wait for KV capacity and reducing overall memory efficiency. To address this limitation, we present ReFrame, a serving runtime that dynamically shares GPU memory between expert weights and KV cache while preserving sufficient expert residency. First, typed frame rebinding enables GPU memory to alternate between expert and KV cache while retaining the native tensor and block views required by each consumer. Second, a bounded allocation policy coordinates when and how much memory to borrow, confines each loan to the triggering request, and uses private capacity first to leave shared blocks available for other requests. Finally, we evaluate ReFrame on two mixed workloads with Qwen3-235B-A22B. Against CPU-offloading baselines, the complete system achieves geometric mean speedups of $2.75\times$ over vLLM 0.29 and $2.94\times$ over SGLang 0.5.17 in trace completion time. An independent matched-batch comparison with vLLM 0.29 yields $2.74\times$, and a separate comparison with its 2,048-token batch budget yields $3.15\times$.

1 INTRODUCTION

Mixture-of-experts (MoE) has become a widely adopted architecture for scaling large language models (LLMs) due to its sparse activation. Specifically, MoE can increase parameter capacity without a proportional increase in per-token computation using sparse expert activation (Fedus et al., 2022; Du et al., 2022; Rajbhandari et al., 2022). In memory-constrained deployments, expert weights can occupy a substantial portion of GPU memory, leaving limited capacity for other runtime states such as key–value (KV) caches. To address this problem, existing systems commonly offload the weights of less frequently used experts to CPU memory and load them into GPU memory on demand via host-to-device transfers (Eliseev & Mazur, 2023; Leyang et al., 2025; Du et al., 2024). However, only a subset of experts is activated for each token, leaving the GPU memory occupied by resident experts underutilized. Such underutilized memory, which could instead provide additional capacity for the KV cache (Kwon et al., 2023; Li et al., 2025), represents a primary bottleneck for serving long-context and concurrent requests.

Existing research that alleviates this limitation can be broadly divided into two categories. The first improves expert supply through caching, prefetching, and CPU–GPU scheduling (Leyang et al., 2025; Cao et al., 2025; Song et al., 2024), and the second focuses on KV allocation (Kwon et al., 2023; Prabhu et al., 2025) and reuse (Zheng et al., 2024; Gim et al., 2024). Although these approaches improve memory efficiency, they still operate with separate and fixed memory pools that cannot adapt to changing demand. A more flexible approach is to dynamically reassign GPU memory between different objects (Xu et al., 2025; Liu et al., 2026). For example, eLLM unifies physical memory management for runtime tensors and KV caches, allowing memory to be reallocated between them as demand changes (Xu et al., 2025). In this paper, we mainly focus on MoE serving systems, which still divide GPU memory into fixed pools for experts and KV caches. As a result, for efficient use of GPU memory, MoE serving requires bidirectional GPU memory reallocation to alleviate KV cache pressure when needed and restore expert residency afterward.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

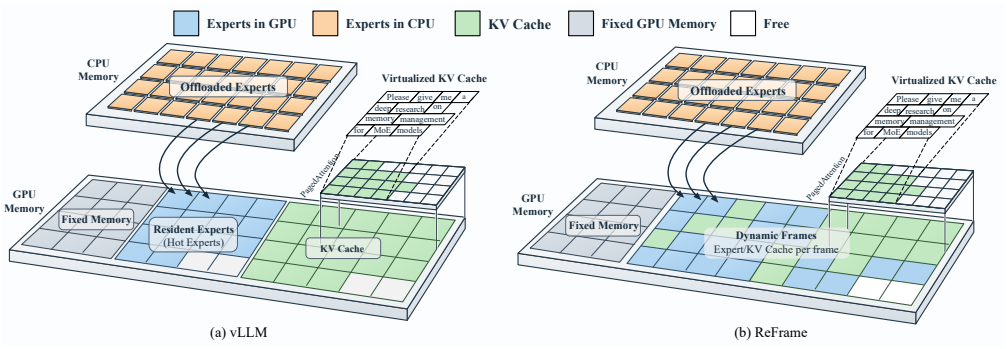

Figure 1: From fixed pools to typed frames. The vLLM-style layout assigns separate GPU regions to resident experts and KV cache. ReFrame retains separate logical views while assigning the physical frames to either type.

Therefore, we present ReFrame, a serving runtime that enables temporary expert–KV memory sharing through typed frame rebinding with bounded and request-isolated allocation. Typed frame rebinding provides the foundation for safely converting memory between expert-weight and KV-cache usage through a shared physical frame arena, native logical views, and an ownership protocol, as shown in Figure 1. Building on this, the allocation policy determines when and how much memory to borrow, which requests can use the borrowed capacity, and how blocks are prioritized during allocation. In addition, to support expert execution as residency changes, a rank-wide expert cache and bounded host preparation supply weights throughout the borrow-and-return cycle. To evaluate ReFrame, we measure end-to-end runtime performance on Qwen3-235B-A22B with four A100 80-GB GPUs across two source-disjoint workloads. Compared with vLLM 0.29 CPU offload, ReFrame achieves a $3.15\times$ completion-time speedup, accompanied by p95 time-to-first-token (TTFT) and time-per-output-token (TPOT) speedups of $5.63\times$ and $3.10\times$. Compared with vLLM 0.11 CPU offload, the corresponding speedups increase to $12.45\times$, $27.96\times$, and $20.73\times$, respectively. The experimental results are detailed in Section 5.

We make three contributions:

1. We introduce typed frame rebinding, a memory abstraction that enables safe, bidirectional Expert–KV memory reuse by combining shared physical frames, native logical views, and an ownership transition protocol.
2. We develop a bounded, request-isolated allocation policy that selects the smallest feasible candidate-frame prefix, reserves borrowed capacity for the triggering request, and prioritizes private blocks to preserve shared capacity.
3. We implement ReFrame and evaluate it on Qwen3-235B-A22B, achieving geometric-mean trace-completion speedups of $2.75\times$ over vLLM and $2.94\times$ over SGLang with CPU offloading, alongside a $4.90\times$ p95 first-token speedup over SGLang.

2 Related Work

MoE architectures and expert execution. Gated MoE, GShard, Switch Transformers, and Mixtral expand capacity through sparse activation (Shazeer et al., 2017; Lepikhin et al., 2020; Fedus et al., 2022; Jiang et al., 2024). DeepSeekMoE uses fine-grained/shared experts; DeepSeek-V2 compresses latent KV (Dai et al., 2024; Liu et al., 2024). These choices shape expert demand and KV. FlexGen schedules CPU/GPU/disk transfers, MoE-Infinity caches and prefetches experts, Fiddler chooses CPU execution or GPU weight transfer, and FluxMoE pages experts (Sheng et al., 2023; Leyang et al., 2025; Kamahori et al., 2025; Liu et al., 2026). Efficient expert supply, however, must coexist with evolving KV demand under the same GPU memory budget. Our study examines how expert residency can adapt to this competition while preserving model weights and routing.

KV allocation and serving schedules. PagedAttention allocates KV blocks, vAttention separates virtual contiguity from physical allocation, and CachedAttention retains KV across turns (Kwon

et al., 2023; Prabhu et al., 2025; Gao et al., 2024). Orca schedules iterations and selective batches; Sarathi-Serve chunks prefill to limit decode interference; DistServe separates prefill/decode resources under latency constraints (Yu et al., 2022; Agrawal et al., 2024; Zhong et al., 2024). These approaches improve KV allocation, state reuse, and scheduling. A complementary challenge arises when requests wait for KV capacity while memory remains assigned to expert residency. Our study focuses on making that capacity usable for request admission through native KV interfaces.

Memory sharing across types. SLoRA pools adapter/KV pages; Punica shares base models and batches LoRA adapters (Sheng et al., 2024; Chen et al., 2024). eLLM shares runtime/KV storage through virtual tensors, while Oneiros repurposes parameters, including inactive models' storage (Xu et al., 2025; Li et al., 2025). These approaches show how memory can be shared across storage roles and models. In our setting, the question extends beyond reclaiming capacity: a temporary transfer from experts to KV cache must safely produce usable KV blocks, and its admission benefit must be weighed against expert reloads and displaced work.

3 Typed Frame Rebinding

As shown in Figure 2, we introduce a mechanism called typed frame rebinding to safely reassign physical frames between expert and KV ownership and return them after use. The mechanism preserves native logical views through an ownership transition protocol.

3.1 Physical Frames and Logical Views

Expert slots retain kernel-compatible tensor layouts; KV retains native block identities and block tables. Rebinding changes frame eligibility while keeping active requests' block tables in place.

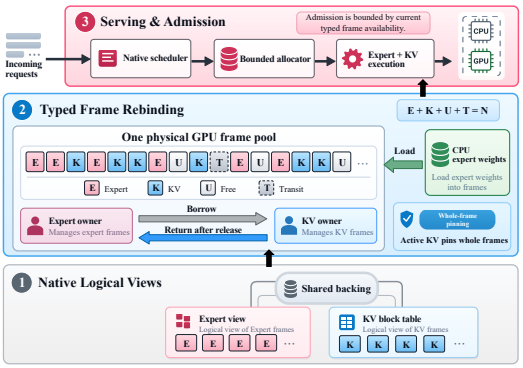

Figure 2: Typed frame rebinding connects native logical views to a shared frame arena. Ownership transactions publish ready backing for request-eligible native execution. Active KV pins whole frames. Tiles are schematic.

For one rank, let $\mathcal{G}$ be an arena of $N \in \mathbb{N}_{>0}$ physical frames, each of $F \in \mathbb{N}_{>0}$ bytes. Let $E(t)$, $K(t)$, $U(t)$, and $T(t)$ count Expert-owned, KV-owned, free, and in-transit frames. At time $t \geq 0$, the four frame states partition $\mathcal{G}$ into disjoint, exhaustive subsets, with nonnegative integer counts satisfying $E(t) + K(t) + U(t) + T(t) = N$. Expert weights are reconstructible, whereas live KV pins its backing frames. Borrowing and return redistribute capacity between E and K without changing the total arena capacity NF. Non-expert weights and workspaces remain outside this accounting but within the device budget.

For native block identities $b \in \mathcal{B}$, let $\varnothing \neq \mathcal{F}(b) \subseteq \mathcal{G}$ be their backing frames. The scheduler reserves logical blocks; each rank converts its backing before model execution. Device access requires published, ready backing.

3.2 Ownership Transitions and Return

Each frame carries a published type and generation that govern consumer access. An ownership transition *quiesces* source users, *retires* their publication, *materializes* the destination, and *publishes* a valid generation, as shown in Figure 3. Neither consumer can access a frame in Transit. Expert ownership requires reconstructed contents. KV publication requires backing ready for native writes, followed by token-state population before dependent attention reads. A failed transition invokes validated recovery or terminates the affected execution.

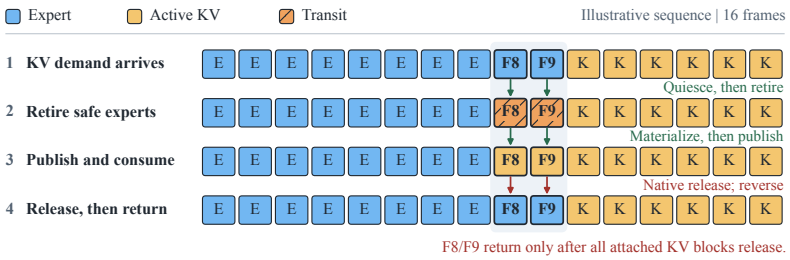

Figure 3: Schematic borrowing and return in a 16-frame arena. F8/F9 become KV-usable only after source retirement, materialization, and publication. Return follows release of all attached KV blocks and reconstruction of the Expert owner.

Proposition 1 (ownership invariant). For $f \in \mathcal{G}$ after step $n \in \mathbb{N}_0$, let $\Pi_n(f) \subseteq \{\text{Expert}, \text{KV}\} \times \mathbb{N}_0$ be the published owner–generation pairs and $\mathcal{R}_n(f)$ the protected uses. Assume $|\Pi_0(f)| \leq 1$ with valid initial protection, serialized per-frame transitions, fresh generations, and atomic validation and protection until use completes. A retirement may occur only when $\mathcal{R}_n(f) = \varnothing$, at which point publication is atomically cleared. Materialization does not require publication, whereas publication requires $\Pi_n(f) = \varnothing$ and destination readiness, and atomically installs a single owner–generation pair. Live KV references protect every backing frame, and recovery obeys these rules. Then $|\Pi_n(f)| \leq 1$ for all $n \geq 0$, and live KV references prevent retirement of their backing frames. Appendix I.2 proves these properties of the transition model.

Releasing a frame's last live KV reference makes it available for Expert reconstruction. Subsequent expert materialization rebuilds residency toward the startup capacity target, with contents determined by current routing and cache placement.

4 Request-Isolated Borrowing and Allocation

Building on the above mechanism, we introduce a request-isolated borrowing and allocation policy that decides when to borrow, how much capacity to borrow, which request receives the borrowed capacity, and which blocks are consumed first. The policy invokes typed frame rebinding only after these decisions are made, separating allocation decisions from ownership transitions.

4.1 Borrowing Triggers, Size, and Request Isolation

A loan supplies native KV blocks according to their backing frames and the requesting owner. For request identities $i \in \mathcal{I}$, let $\mathcal{B}_{\text{free}}(t) \subseteq \mathcal{B}$ be the free block identities and $\mathcal{K}_{\text{ready}}(t) \subseteq \mathcal{G}$ the published, ready KV frames. The single-valued lease map $\lambda_t : \mathcal{G} \to \mathcal{I} \cup \{\perp\}$ assigns i to a frame leased to request i and $\perp \notin \mathcal{I}$ to unrestricted frames. The request-eligible set is

$$\mathcal{A}_i(t) = \{b \in \mathcal{B}_{\text{free}}(t) : \mathcal{F}(b) \subseteq \mathcal{K}_{\text{ready}}(t), \quad \forall f \in \mathcal{F}(b),\ \lambda_t(f) \in \{\perp, i\}\}. \tag{1}$$

For demand $d_i(t) \in \mathbb{N}_0$ additional blocks, $|\mathcal{A}_i(t)| \geq d_i(t)$ is an execution-ready capacity condition; native execution checks still apply. Every backing frame must be ready and authorized. Loan size, usable blocks, and virtual address range are distinct quantities.

ReFrame triggers borrowing after native allocation fails and allocator-visible pressure reaches 1,000 ms, with at most six decoding requests. This pressure interval differs from arrival-to-admission wait. At a fixed allocation snapshot with a capacity shortage $|\mathcal{A}_i(t)| < d_i(t)$, let C_k contain the first k distinct borrowable frames outside the request-visible KV pool in the frozen candidate order and let $\widehat{\mathcal{A}}_i(C_k, t)$ denote Eq. 1 with those frames hypothetically ready and leased to i. The controller selects the smallest feasible prefix,

$$k^* = \min\{k \in \{1, \ldots, r\} : |\widehat{\mathcal{A}}_i(C_k, t)| \geq d_i(t)\}, \qquad r \in \{0, 1, 2\}, \tag{2}$$

where r is capped by the remaining episode allowance and the available candidate count. An empty feasible set, including $r = 0$, rejects borrowing. The hypothetical sets hold other state fixed and

count only complete native blocks; Appendix I.3 defines them and proves prefix minimality. The scheduler reserves blocks against this projected eligibility. Each rank converts Expert-owned backing through typed frame rebinding and validates readiness before execution; already ready KV frames need no conversion.

The active and pending request sets define each busy episode, which permits one activation. Free-count queries and block selection reserve borrowed frames for the triggering request. Each activation is sized for that owner's current demand, and the loan persists while the owner retains live KV references. Decode-edge borrowing is disabled; the owner can nevertheless use an existing loan for growth. Final KV release permits return through typed frame rebinding. The native scheduler governs chunked prefill, decoding, and preemption. Request isolation reserves capacity for its owner; future growth remains subject to native allocation.

4.2 Private-First Allocation

Fix request i and time t. Initial reservation uses projected eligibility $\widehat{\mathcal{A}}_i(C_{k^*}, t)$ with selected frames leased to i; the following rule applies to either eligible set. Partition $\mathcal{A}_i(t)$ into finite disjoint sets $\mathcal{P}$ and $\mathcal{S} = \mathcal{A}_i(t) \setminus \mathcal{P}$, with $p = |\mathcal{P}|$ and $s = |\mathcal{S}|$.

Proposition 2 (minimum shared consumption). For independently selectable blocks at this fixed state and integer demand $d = d_i(t)$ satisfying $0 \le d \le p + s$, private-first selection uses the minimum possible number of shared blocks,

$$s_{\text{used}} = \max(0, d - p), \qquad s_{\text{remaining}} = s - \max(0, d - p). \tag{3}$$

Appendix I.4 proves the bound and covers zero demand and empty private or shared sets.

Request isolation excludes blocks touching frames leased to i from request $j \neq i$ throughout the lease. Consuming owner-private blocks first avoids occupying substitutable shared blocks and preserves the largest shared pool at that state.

Block-to-frame and eligible-free indexes exclude other owners' leases and inspect bounded candidates. Without leases or new candidates, a maintained count suffices. Caches retain request and eligibility context; indexing and ownership transactions incur execution cost.

A rank-global cache and four host preparation workers supply experts through disjoint staging regions within a bounded pinned-memory budget, separately from GPU publication. The prototype uses a single decoder KV group, disabled prefix caching, eager execution, and serialized per-rank engine entry. The internal controls retain this expert-supply path. Static ReFrame also differs in configuration, whereas ReFrame without rebinding toggles borrowing within the same configuration. Appendix H describes these controls and their costs.

Service effects. For a completed request, TTFT $Y_i = W_i + P_i$ comprises pre-admission wait and time from admission to first content. Let a_i, t_i^{adm}, and t_i^{first} be the planned arrival, first admission, and first content times on a common time axis, with $a_i \le t_i^{\text{adm}} \le t_i^{\text{first}}$. Then $W_i = t_i^{\text{adm}} - a_i$ and $P_i = t_i^{\text{first}} - t_i^{\text{adm}}$. P_i includes scheduling, execution, and delivery. For the same request and planned arrival, define $\Delta X = X^{\text{borrow}} - X^{\text{fixed}}$ for $X \in \{Y_i, W_i, P_i\}$. Then

$$\Delta Y_i = \Delta W_i + \Delta P_i, \qquad \Delta Y_i < 0 \iff \Delta P_i < -\Delta W_i. \tag{4}$$

Reduced waiting improves TTFT when it outweighs post-admission time from prefill interference and expert reloads. Effects on decoding and recovery motivate evaluating TTFT, TPOT, and completion time. Appendices I.1 and H give the derivation, timing assumptions, and service accounting.

5 Experiments

5.1 Experimental setup

Baselines. ReFrame is implemented on vLLM 0.11.0. We compare **ReFrame** with **vLLM 0.11**, **vLLM 0.29** (Kwon et al., 2023), and **SGLang 0.5.17** (Zheng et al., 2024). All three baselines use 40-GiB/GPU CPU offload after native configurations fail resource qualification. Internal comparisons use Static ReFrame with a fixed KV pool and a same-configuration ReFrame control with rebinding disabled; both retain the expert-supply path.

Models. Our primary model is **Qwen3-235B-A22B-Instruct-2507** (Yang et al., 2025; Qwen Team, 2025), a MoE model with 235 billion total and 22 billion active parameters. We use BF16 weights, whose approximately 437.90-GiB footprint exceeds aggregate GPU capacity.

Testbed. All primary systems run on four NVIDIA A100 80-GB GPUs, with tensor parallelism (TP=4) and expert parallelism (EP=4) configured over the same four GPUs. All systems support 16,384-token sequences and at most 12 concurrent requests, and greedily generate exactly 128 tokens while ignoring EOS. Table 1 summarizes the profiles.

Workloads. Mixed workloads 1 and 2 use disjoint sources. Each combines 16 long GovReport prompts (Huang et al., 2021), drawn from LongBench (Bai et al., 2024) (12,153–12,224 tokens), with 20 short WildChat prompts (Zhao et al., 2024) (256–2,040 tokens). Ten early short requests arrive at 0–135 s, the long burst at 300–303.75 s, and ten late short requests at 660–795 s. Systems receive identical text, tokenization, and planned arrivals within each workload. Each three-system comparison run starts a fresh service process: vLLM, ReFrame, then SGLang on workload 1, and the reverse order on workload 2. We conduct an independent matched-batch comparison between ReFrame and the same strengthened vLLM configuration. A separate short-request workload contains 36 new requests at 15-s intervals.

Metrics. The primary metric is **trace completion time**, measured from the first planned request arrival to the final HTTP completion. To characterize request-level latency, we also report **TTFT**, measured from the planned arrival time to the first returned token, and **TPOT**, computed as the interval between the first and last content tokens divided by 127. We report request-level percentiles using the nearest-rank method and compute speedups by dividing each baseline metric by the corresponding ReFrame metric within the same source. The headline results aggregate the two source-disjoint workloads using an equally weighted geometric mean. Because each source is evaluated with a single run per system in the three-system comparison, the reported request distributions characterize within-run request behavior rather than variability across independent system repetitions. Appendix A details timing checks and internal-ablation endpoints.

5.2 Complete-system comparison with frameworks

We compare complete ReFrame with vLLM 0.11, vLLM 0.29, and SGLang 0.5.17 on the two mixed workloads to evaluate serving performance when model weights exceed aggregate GPU memory. In this baseline order, the equal-workload geometric-mean speedups are $12.45\times$, $2.75\times$, and $2.94\times$ for completion, $27.96\times$, $4.58\times$, and $4.90\times$ for TTFT p95, and $20.73\times$, $3.09\times$, and $1.42\times$ for TPOT p95, as shown in Figure 4. Each comparison retains its source-matched paired ReFrame runs and the resource profiles in Table 1. Table 2 reports absolute latencies, and every displayed run completes 36/36 requests and 4,608 output tokens. These speedups characterize complete serving systems.

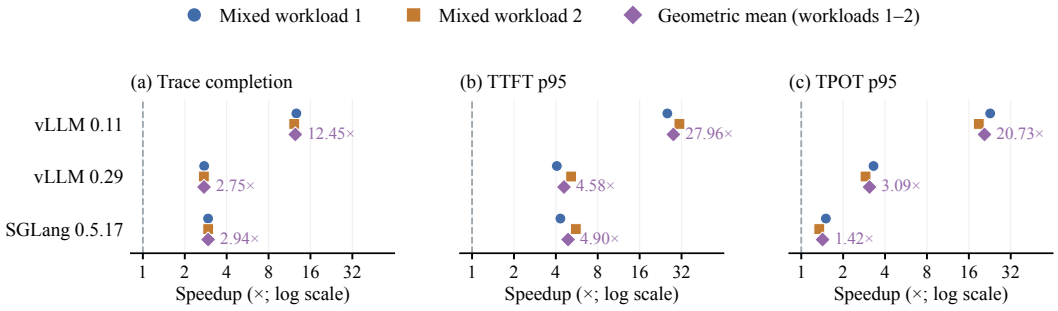

Figure 4: Source-matched ReFrame speedups with the profiles in Table 1. Geometric means cover workloads 1–2; additional-source results compare with vLLM 0.29 and SGLang.

Repeated runs and two vLLM configurations allow us to examine the consistency of the observed gains and their sensitivity to the baseline's batch-token budget. The independent matched-batch comparison gives a $2.74\times$ completion speedup over the same vLLM profile, with $4.55\times$ TTFT p95 and $3.04\times$ TPOT p95 speedups. The three-system comparison yields similar results with fresh service processes and reversed method order. These repetitions assess consistency on the same two workload sources. Under the 2,048-token vLLM 0.29 batch budget, completion speedup is $3.15\times$; increasing this budget to 16,384 tokens narrows the advantage while preserving it. Appendix B reports the vLLM comparisons and internal static references; Appendices C and D detail the con-

Table 1: Primary execution profiles. Memory controls have framework-specific meanings.

System	Batch / prefill chunk	GPU-memory control	Execution
vLLM 0.11	16,384 batch	0.98 utilization	CUDA graphs enabled
vLLM 0.29	16,384 batch	0.98 utilization	CUDA graphs enabled
SGLang 0.5.17	4,096 prefill chunk	0.92 static fraction	Decode-only CUDA graphs
ReFrame	16,384 batch	0.98 utilization	Eager

Table 2: Complete-system results in seconds; lower is better. ReFrame is shown once per workload. Bold marks the lowest value per metric. Every row completes 36 requests and 4,608 output tokens.

System	Completion (s)	TTFT p95 (s)	TPOT p95 (s)
Mixed workload 1			
vLLM 0.11	16,360.30	14,701.27	65.91
vLLM 0.29	3,533.38	2,352.61	9.69
SGLang 0.5.17	3,768.73	2,494.81	4.40
ReFrame	**1,281.42**	**578.75**	**2.93**
Mixed workload 2			
vLLM 0.11	16,091.94	14,453.61	64.29
vLLM 0.29	3,423.16	2,236.21	9.23
SGLang 0.5.17	3,678.66	2,416.27	4.30
ReFrame	**1,247.78**	**434.12**	**3.19**

figurations, qualification outcomes, and failures. The completion advantage therefore persists under the vLLM budget and also holds against SGLang.

To assess whether the advantage extends to another long-text source, we replace the 16 GovReport prompts with SummScreenFD prefixes, retaining the 20 WildChat prompts and planned arrivals. ReFrame achieves completion speedups of $3.05\times$ and $3.54\times$ over vLLM 0.29 and SGLang 0.5.17, respectively, as shown in Table 3, with lower TTFT p95 and TPOT p95. The advantage therefore extends to this additional long-text source.

Table 3: Additional-source results on SummScreenFD + WildChat, in seconds; lower is better. Each system completes 36 requests and 4,608 output tokens. Bold marks the lowest value per metric.

System	Completion (s)	TTFT p95 (s)	TPOT p95 (s)
vLLM 0.11	16,174.77	14,505.27	66.22
vLLM 0.29	3,361.54	2,213.67	9.39
SGLang 0.5.17	3,902.10	2,610.67	4.53
ReFrame	**1,102.06**	**412.76**	**2.75**

5.3 Request latency and progress through the workload

Beyond aggregate endpoints, the empirical TTFT and TPOT distributions show how latency varies across individual requests. As shown in Figure 5, the empirical distributions cover all 36 requests per system and workload across ReFrame and the three baselines. ReFrame has lower TTFT p95 and TPOT p95 than each paired baseline. SGLang has lower TPOT p95 than vLLM 0.29, yet its TTFT p95 and trace completion are higher. Decoding latency alone therefore does not capture the completion behavior of these mixed traces. The gain thus spans first-token responsiveness and decoding tails as well as overall completion.

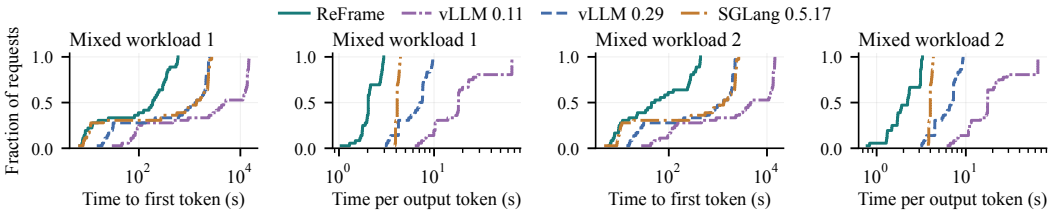

Figure 5: Empirical latency distributions for ReFrame and all three baselines. Each curve contains 36 requests; farther left is better on logarithmic axes.

Cumulative request completions provide a temporal view of serving progress through each workload. As shown in Figure 6, the completion trajectories follow requests from the first planned arrival

on a common linear time axis, preserving the early short wave, long burst, and late short wave of each trace. ReFrame reaches final completion earlier than each paired baseline. The trajectories and distributions show the same eight runs, adding no new runtime measurements. Faster completion therefore reflects an earlier drain of the same workload.

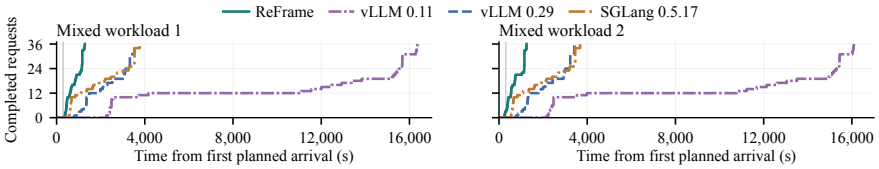

Figure 6: Completion trajectories under matched planned arrivals. Shading marks the long burst at 300–303.75 s. Every run completes all 36 requests; axes retain the full trace duration.

We then examine how ReFrame's latency advantage varies across the early-short, long-burst, and late-short arrival cohorts. Table 4 applies the three prespecified arrival phases to all baseline comparisons. This post-hoc descriptive analysis reports ratios of nearest-rank medians, using ten early short, sixteen long, and ten late short requests per workload. ReFrame improves TTFT, TPOT, and request end-to-end latency in every phase against all three baselines. For late short requests, SGLang TTFT speedups reach $10.64\times$ and $33.70\times$, while TPOT speedups are $1.94\times$ and $1.24\times$. These phase-level differences help locate the service advantage without assigning it to a single runtime component. The latency advantage extends to requests arriving during and after the long burst.

Table 4: Cohort-wise median latency. TTFT and E2E are in seconds; TPOT is in seconds per output token. Lower is better. Bold marks the lowest value within each matched comparison.

System	Early short (10 requests)			Long burst (16 requests)			Late short (10 requests)		
	TTFT	TPOT	E2E	TTFT	TPOT	E2E	TTFT	TPOT	E2E
Mixed workload 1									
vLLM 0.11	77.83	18.02	2,368.07	4,710.77	25.79	13,075.68	13,714.75	8.28	15,015.84
vLLM 0.29	23.65	9.05	1,172.75	1,274.84	7.56	2,227.52	2,127.77	3.50	2,664.06
SGLang 0.5.17	9.39	4.31	556.98	1,351.27	4.09	1,870.99	2,327.92	3.93	2,827.28
ReFrame	**8.27**	**2.84**	**371.76**	**307.93**	**1.96**	**530.42**	**218.77**	**2.03**	**476.82**
Mixed workload 2									
vLLM 0.11	95.22	17.53	2,321.39	4,582.86	26.23	12,749.38	13,381.28	8.24	14,779.36
vLLM 0.29	19.29	8.64	1,116.58	1,208.52	7.27	2,135.93	1,999.37	3.62	2,569.39
SGLang 0.5.17	9.32	4.21	544.18	1,310.87	4.00	1,818.20	2,262.53	3.84	2,750.40
ReFrame	**7.75**	**2.23**	**292.28**	**249.90**	**1.70**	**414.37**	**67.14**	**3.11**	**452.78**

We vary request spacing to assess ReFrame's serving performance under different arrival densities. At relative arrival densities of 0.5, 1, and 2, ReFrame achieves completion speedups of $1.98 - -2.89\times$ over vLLM 0.29 and $2.07 - -3.00\times$ over SGLang 0.5.17. It also maintains lower TTFT p95 and TPOT p95 at every density, as shown in Figure 7, indicating that its advantage extends across the tested arrival schedules. Because request spacing determines the trace's arrival span, Appendix F complements these results with drain tail and mean request latency to distinguish overall completion from the waiting experienced by individual requests.

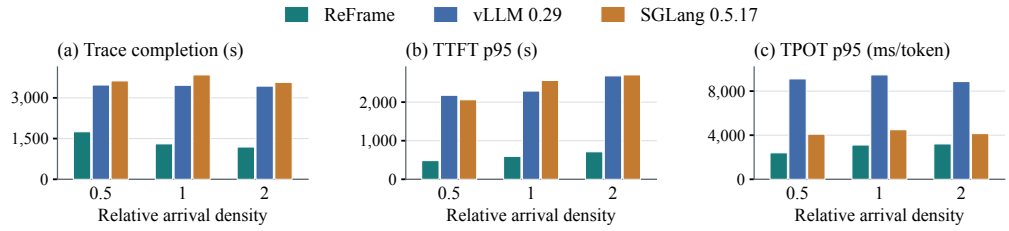

Figure 7: Arrival-density sensitivity on Mixed workload 1; lower is better. Each group compares the three systems at the same relative density. Each bar represents one fresh service process serving 36 requests with 128 output tokens per request.

5.4 RUNTIME CONTROLS AND OPERATING BOUNDARIES

The short-request workload evaluates low-pressure serving when borrowing is inactive. On the same 36 WildChat prompts and 15-s planned arrivals, ReFrame's paired completion speedups are $3.35\times$, $1.54\times$, and $1.28\times$ over vLLM 0.11, vLLM 0.29, and SGLang 0.5.17, respectively. Each speedup retains its campaign's paired ReFrame reference. Table 5 displays all five methods with one ReFrame row from the supplemental campaign. These results remain outside the two-source headline aggregate. The advantage therefore extends beyond mixed traces to this short-only workload.

Table 5: Source-matched short-request latencies; lower is better and bold marks the lowest displayed value per metric. All rows complete 36/36 requests and 4,608 output tokens. ReFrame, vLLM 0.29, and SGLang share one campaign; vLLM 0.11 and Static ReFrame retain their separately measured absolute values. Speedup claims use the corresponding campaign's paired ReFrame run.

Method	TTFT (s)		TPOT (ms/token)		Completion (s)
	p50	p95	p50	p95	
vLLM 0.11	1,401.80	2,825.42	11,055.67	11,726.77	4,534.47
vLLM 0.29	512.13	1,008.16	5,018.35	5,118.37	1,997.60
SGLang 0.5.17	341.94	676.49	3,953.27	3,956.90	1,661.05
Static ReFrame	310.82	585.13	3,468.84	3,732.60	1,405.79
ReFrame	**267.95**	**483.99**	**3,050.44**	**3,441.93**	**1,299.65**

The borrowing audit verifies a completed lend-and-restore cycle without loan-owner preemption in Appendix G.1. The selected internal pair is documented in Appendix G.

Figure 8 shows complete internal pairs selected retrospectively by the lowest paired long-request TTFT ratio among four repetitions for each control. On mixed workload 1, the selected pairs reduce this metric by 12.78% relative to Static ReFrame and 12.17% relative to the same-configuration ReFrame control without rebinding; both also improve TPOT p95 and completion. Selection on one metric does not estimate a typical gain: across all four ReFrame/Static ReFrame pairs, long-request TTFT improves by 1.24% while whole-trace TPOT p95 regresses by 2.29%.

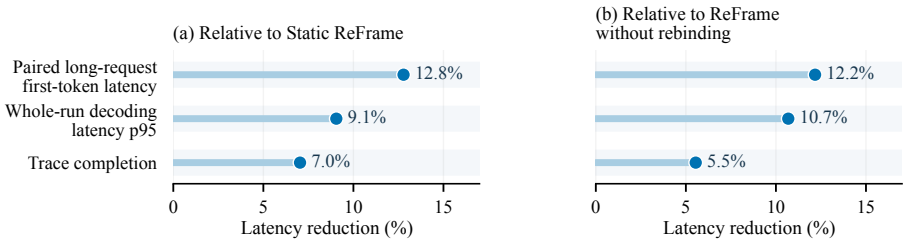

Figure 8: Retrospectively selected complete internal pairs; larger latency reductions are better. Each control has four repetitions, and selection uses the lowest paired long-request TTFT ratio. All endpoints from each selected pair are shown.

6 CONCLUSION

Expert residency and KV allocation are coupled resource demands in memory-constrained MoE serving. This work shows how a shared physical arena can coordinate these demands while preserving native expert-tensor and KV-block views. Typed frame rebinding separates physical ownership from logical representation, and bounded, request-isolated allocation connects capacity reassignment to request progress. Together, these mechanisms make expert storage and KV capacity jointly manageable within an existing serving runtime. The complete ReFrame runtime achieves $2.75\times$ and $2.94\times$ geometric-mean completion speedups over vLLM 0.29 and SGLang 0.5.17 with CPU offload on two mixed workloads, alongside lower TTFT and TPOT tails. An independent vLLM comparison yields a consistent $2.74\times$ speedup. These results support that coordinating memory ownership, expert supply, and request admission is an effective approach to serving MoE models whose weights exceed aggregate GPU memory. Future work will extend ReFrame to broader MoE architectures and sustained serving workloads with variable generation lengths. We will also investigate allocation policies that jointly account for expert reload cost, host-memory pressure, and request-level latency objectives.

7 GENERATIVE AI USAGE

The authors used generative AI tools solely for grammar correction, language polishing, and improving the clarity of manuscript drafts. These tools were not used to design the methodology, generate data, conduct experiments, train models, produce numerical results, or derive scientific conclusions. All scientific content, experimental results, interpretations, and claims were independently reviewed and verified by the authors.

REFERENCES

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 117–134, 2024.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 3119–3137, 2024.

Shiyi Cao, Shu Liu, Tyler Griggs, Peter Schafhalter, Xiaoxuan Liu, Ying Sheng, Joseph E Gonzalez, Matei Zaharia, and Ion Stoica. Moe-lightning: High-throughput moe inference on memory-constrained gpus. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pp. 715–730, 2025.

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant lora serving. *Proceedings of Machine Learning and Systems*, 6:1–13, 2024.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 1280–1297, 2024.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International conference on machine learning*, pp. 5547–5569. PMLR, 2022.

Zhixu Du, Shiyu Li, Yuhao Wu, Xiangyu Jiang, Jingwei Sun, Qilin Zheng, Yongkai Wu, Ang Li, Hai H Li, and Yiran Chen. Sida: Sparsity-inspired data-aware serving for efficient and scalable large mixture-of-experts models. *Proceedings of Machine Learning and Systems*, 6:224–238, 2024.

Artyom Eliseev and Denis Mazur. Fast inference of mixture-of-experts language models with offloading. *arXiv preprint arXiv:2312.17238*, 2023.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. {Cost-Efficient} large language model serving for multi-turn conversations with {CachedAttention}. In *2024 USENIX annual technical conference (USENIX ATC 24)*, pp. 111–126, 2024.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt Cache: Modular attention reuse for low-latency inference. In *Proceedings of Machine Learning and Systems*, volume 6, pp. 325–338, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/a66caa1703fe34705a4368c3014c1966-Abstract-Conference.html.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. In *Proceedings of the 2021 conference of the north American chapter of the association for computational linguistics: Human language technologies*, pp. 1419–1436, 2021.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Keisuke Kamahori, Tian Tang, Yile Gu, Kan Zhu, and Baris Kasikci. Fiddler: Cpu-gpu orchestration for fast inference of mixture-of-experts models. In *International Conference on Learning Representations*, volume 2025, pp. 56099–56115, 2025.

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

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

XUE Leyang, FU Yao, LU Zhan, et al. Moe-infinity: Offloading-efficient moe model serving. *arXiv preprint arXiv:2401.14361*, 2025.

Ruihao Li, Shagnik Pal, Vineeth Narayan Pullu, Prasoon Sinha, Jeeho Ryoo, Lizy K John, and Neeraja J Yadwadkar. Oneiros: Kv cache optimization through parameter remapping for multi-tenant llm serving. In *Proceedings of the 2025 ACM Symposium on Cloud Computing*, pp. 88–101, 2025.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

Qingxiu Liu, Cyril Y He, Hanser Jiang, Zion Wang, Alan Zhao, and Patrick PC Lee. Fluxmoe: Decoupling expert residency for high-performance moe serving. *arXiv preprint arXiv:2604.02715*, 2026.

Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, and Ashish Panwar. vattention: Dynamic memory management for serving llms without pagedattention. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pp. 1133–1150, 2025.

Qwen Team. Qwen3-235B-A22B-Instruct-2507 model card, 2025. URL `https://huggingface.co/Qwen/Qwen3-235B-A22B-Instruct-2507`. Accessed September 15, 2026.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International conference on machine learning*, pp. 18332–18346. PMLR, 2022.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pp. 31094–31116. PMLR, 2023.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, et al. Slora: Scalable serving of thousands of lora adapters. *Proceedings of machine learning and systems*, 6:296–311, 2024.

Xiaoniu Song, Zihang Zhong, Rong Chen, and Haibo Chen. Promoe: Fast moe-based llm serving using proactive caching. *arXiv preprint arXiv:2410.22134*, 2024.

Jiale Xu, Rui Zhang, Yi Xiong, Cong Guo, Zihan Liu, Yangjie Zhou, Weiming Hu, Hao Wu, Changxu Shao, Ziqing Wang, et al. ellm: Elastic memory management framework for efficient llm serving. *arXiv preprint arXiv:2506.15155*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX symposium on operating systems design and implementation (OSDI 22)*, pp. 521–538, 2022.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. Wildchat: 1m chatgpt interaction logs in the wild. In *International Conference on Learning Representations*, volume 2024, pp. 34590–34605, 2024.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. {DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 193–210, 2024.

A Workloads and Experimental Protocol

Runtime and input contract. The primary model, hardware, and request contract are specified in Section 5.1; earlier profiles appear in Appendix B. Each ReFrame variant uses four host preparation workers and begins with 94 key/value frames (4,608 native blocks) in a 141-frame addressable range (6,912 blocks); each frame occupies 36 MiB. vLLM uses its unmodified entry point, native key/value sizing, chunked prefill, and graph-capable execution. The later vLLM 0.29 profile omits the partial-prefill limit arguments unavailable in that version.

The mixed workloads use disjoint source identities; identified historical and calibration inputs are excluded. Together with the separate 36-prompt short-request workload, they contain 108 distinct inputs. Rerunning a source does not create a new workload.

Execution order. Every method uses a fresh process. On mixed workload 1, the first repeat executes Static ReFrame, ReFrame without rebinding, and ReFrame, with the reverse order in the second repeat. Mixed workload 2 uses the opposite orders. In the initial comparison, each vLLM version runs later on the same inputs and is paired with the first ReFrame runs. The independent matched-batch comparison instead uses new ReFrame and processes, reversing their order across the two workloads (Appendix C).

Timing and aggregation. For planned arrival a_i, first and last nonempty content times t_i^{first} and t_i^{last}, HTTP completion t_i^{end}, and output count n_i, the client records monotonic timestamps and computes

$$\text{E2E}_i = t_i^{\text{end}} - a_i, \qquad \text{TPOT}_i = \frac{t_i^{\text{last}} - t_i^{\text{first}}}{n_i - 1} \quad (n_i > 1). \tag{5}$$

TTFT is $t_i^{\text{first}} - a_i$, and trace completion is $Z = \max_i t_i^{\text{end}} - \min_i a_i$. ReFrame uses nearest-rank percentiles over all 36 requests; p95 is the 35th sorted value. Its primary rebinding endpoints are the median of 16 paired long-request TTFT ratios and the ratio of whole-run TPOT p95 values. Geometric means weight repeats within each source and then sources equally. Arrival replay is checked against the fixed inputs and planned timestamps; zero-loan runs remain included. Requests are descriptive observations within a run, not independent system repetitions.

Resource semantics and reproduction. Device memory is sampled NVML allocation, not live key/value occupancy. Process-tree resident memory can double-count shared mappings; whole-host memory can include other activity. PCIe receive rate is aggregate sampled traffic, not an expert-only transfer counter. The native recovery resource windows also include loading. Five-second sampling does not capture intervening peaks. The supplied source archive retains the measured snapshots, per-client timing tables, and figure generators. The supplemental views below elaborate the main-text comparisons; they are not an exhaustive account of every development experiment.

A.1 Source selection and prompt construction

The workload design separates three sources of variation: prompt identity, the timing of memory demand, and generation length. Mixed workloads 1 and 2 use different documents and conversations but share the same request counts and arrival schedule. Table 6 gives the exact token counts of the selected inputs, including chat formatting, rather than the broader eligibility ranges in Section 5.1. These counts are reproduced from the frozen input manifests; the three-framework mixed-workload records also retain server-reported token counts.

Table 6: Frozen workload inventory. Token counts refer to inputs; each request generates exactly 128 output tokens. The SummScreenFD variant reuses the 20 short requests of Mixed workload 1.

Workload	Short / long	Short tokens	Long tokens	Total tokens
Mixed workload 1	20 / 16	267–761	12,222–12,224	205,593
Mixed workload 2	20 / 16	270–804	12,221–12,224	206,789
Short-request workload	36 / 0	260–1,958	–	21,195
SummScreenFD + WildChat	20 / 16	267–761	12,224–12,224	205,597

Long documents. The two mixed workloads each contain 16 government reports from the GovReport portion of LongBench (Huang et al., 2021; Bai et al., 2024). The construction procedure retains a contiguous prefix of each original document, ending at a natural word boundary and preserving its whitespace. It renders the summarization instruction and chat template before checking the 12,224-token ceiling. This procedure does not pad a document, concatenate unrelated documents, or repeat text to reach a target length. The selected reports occupy 12,222–12,224 tokens in Mixed workload 1 and 12,221–12,224 in Mixed workload 2. The replay records retain the document group, original source identity, retained prefix boundary, rendered-prompt hash, and token count.

Conversational inputs. Short requests come from WildChat (Zhao et al., 2024). A conversation prefix ending at a user turn is rendered with the model's chat template and generation prompt; eligible prefixes contain 256–2,040 tokens. A source-group key identifies the conversation, so separate turns from the same conversation cannot silently become independent inputs across the three primary sources. The two mixed workloads and the short-request workload contain 108 unique rendered-prompt hashes and 108 distinct source groups. The selected mixed-workload short prompts are narrower than the eligibility envelope, as reported in Table 6. These are frozen systems traces constructed from public data, not newly sampled inputs each time a method is run.

Replay identity. A workload record fixes the prompt text, prompt hash, source-group identity, sequence position, planned arrival, and decoding parameters. The same rendered input is replayed across systems within a comparison; a second chat template is not applied at the serving endpoint. Source seeds are 2026092201 and 2026092202 for the mixed workloads and 2026092203 for the short-request workload. Seeds identify the frozen source construction, not independent runtime trials. Later framework comparisons reuse these traces with fresh service processes, while the same-configuration ablation repeats both sources. Their different roles are retained when aggregating results.

A.2 Arrival schedule and pressure phases

Let a_i denote the planned arrival in seconds relative to the first formal request. The mixed workloads use the following deterministic open-loop schedule:

$$\begin{aligned} a_j &= 15j, && j = 0, \ldots, 9, && \text{early short requests,} \\ a_{10+j} &= 300 + 0.25j, && j = 0, \ldots, 15, && \text{long-request burst,} \\ a_{26+j} &= 660 + 15j, && j = 0, \ldots, 9, && \text{late short requests.} \end{aligned} \tag{6}$$

Arrivals do not wait for earlier requests to complete. The serving limit of 12 concurrent requests therefore does not cap the number of submitted requests: the 16-request burst can create pending work. The long burst spans 3.75 s, whereas each short-request phase spans 135 s. The late short phase measures service after the long inputs have entered the system; it does not assume that every long request has already completed.

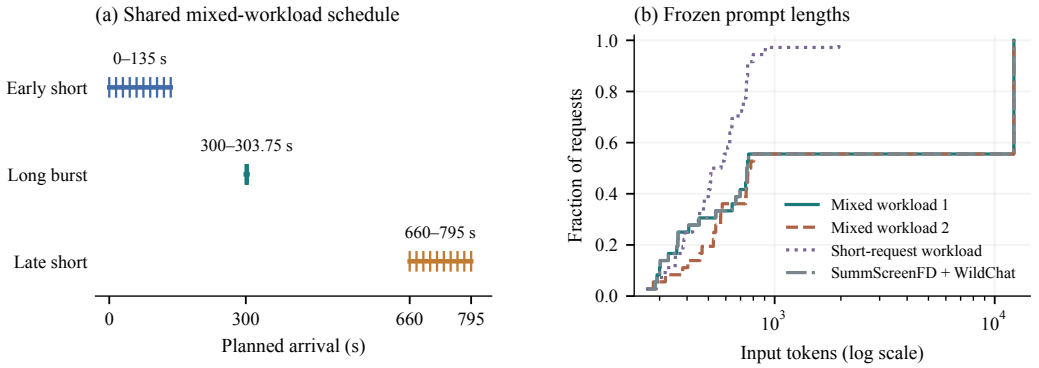

Figure 9: Arrival geometry and empirical prompt-length distributions. Both mixed workloads share the schedule in panel (a). Panel (b) includes all 36 inputs per workload; near-overlapping mixed and SummScreenFD curves reflect their deliberately similar token budgets. The short-request workload instead uses 36 arrivals at 15-s intervals.

The imbalance between request count and input volume is substantial, as shown in Figure 9. Long requests constitute $16/36 = 44.4\%$ of requests and generated tokens, but contribute 95.13% and 94.58% of input tokens in Mixed workloads 1 and 2. Thus, the burst changes prefill demand much more than the request count alone suggests. Mixed workload 2 has 11,210 short-input tokens versus 10,013 in Mixed workload 1, while their long-input totals differ by only one token. The nominal pressure pattern is consequently similar, but the documents, conversation histories, and resulting execution paths differ. Aggregate token counts alone cannot determine expert routing, residency, or which requests benefit from an ownership transition.

The separate short-request workload removes the long-document burst. It submits 36 distinct WildChat requests at $a_i = 15i$ for $i = 0, \ldots, 35$, ending at 525 s, for a total of 21,195 input tokens. "Short-request" describes its input composition, not an assumption that the serving system remains unqueued. Each formal trace has 4,608 generated tokens. Loading, qualification, and warmup are outside the client latency window; the repeated framework services perform four-request warmups before their formal traces. Raw monotonic clocks retain actual-send offsets, allowing replay timing to be checked separately from serving latency.

A.3 Controlled workload variants

The source and schedule transformations are applied before launching a comparison. They change one specified dimension of Mixed workload 1 while retaining the other request identities and controls.

Table 7: Construction of the supplemental traces. A construction contract does not by itself imply a completed performance comparison; qualified coverage is stated in Appendix F.

Variant	Changed property	Preserved properties
SummScreenFD + WildChat	Replace the 16 long reports with 16 episode transcripts	All 20 short prompts, 36 arrivals, and 128-token outputs
Arrival density $\rho \in \{0.5, 1, 2\}$	Set $a_i^{(\rho)} = a_i/\rho$	All 36 prompts, their order, and 128-token outputs
256-token generation	Set the fixed output count to 256	All 36 prompts and the original arrival schedule

The SummScreenFD variant draws long inputs from 16 distinct television-episode transcripts. Its frozen rendering rule takes a 12,224-token prefix of the rendered chat input, then retains the corresponding replay text and token identities. Unlike the GovReport word-boundary procedure, this

token-prefix rule can end inside the transcript; it is used consistently across the compared systems. The archive records the episode identity, original full-prompt hash, retained-prompt hash, and prefix rule. Because the 20 short prompts are unchanged, the experiment tests a change in long-document content under nearly identical input volume, rather than a wholly independent conversational workload.

Arrival density rescales the entire clock, including the gaps between phases. At densities 0.5, 1, and 2, the final arrival occurs at 1,590, 795, and 397.5 s, respectively. These are finite, phased traces, not Poisson arrival streams or sustained-load saturation tests. The 256-token construction fixes both minimum and maximum output lengths at 256 with end-of-sequence termination disabled; it would yield 9,216 output tokens if all 36 requests qualified. The retained manuscript snapshot does not contain a complete qualified three-system comparison for that condition, so no output-length scaling conclusion is drawn.

B Earlier System Configurations

The initial vLLM 0.29 profile uses a 2,048-token batch budget, 0.94 GPU-memory utilization, and 40-GiB/GPU CPU offload, versus 16,384 tokens and 0.98 utilization for ReFrame. In distinct complete 36-request runs, ReFrame achieves a $3.15\times$ geometric-mean completion speedup over that vLLM profile. Table 8 gives the absolute endpoints; the later framework comparison is in the main text.

Table 8: ReFrame and the earlier vLLM 0.29 profile. These are separate runs from the later three-system comparison in Table 2. Lower is better; bold marks the lowest value per workload and metric. Percentiles use nearest ranks; vLLM uses CPU offload.

	TTFT (s) ↓		TPOT (ms/token) ↓		Completion (s) ↓
Method	p50	p95	p50	p95	
Mixed workload 1					
vLLM 0.29	1,563.47	2,933.24	7,465.98	9,812.34	4,090.40
ReFrame	**190.40**	**580.90**	**2,011.73**	**2,892.97**	**1,290.87**
Mixed workload 2					
vLLM 0.29	1,537.32	2,938.96	7,329.87	9,657.45	4,111.34
ReFrame	**71.60**	**467.81**	**1,916.16**	**3,407.53**	**1,314.91**

The prespecified arrival phases also favor ReFrame in all three median endpoints. Table 9 gives the source-matched vLLM 0.29 and vLLM 0.11 contrasts, each with its paired ReFrame reference.

The internal Static ReFrame reference retains expert supply with a fixed KV pool. Its selected paired comparison is shown in Figure 8; Appendix H distinguishes it from the same-configuration control with rebinding disabled.

Table 9: Initial arrival-phase speedups over vLLM 0.29 (upper block) and vLLM 0.11 (lower block): baseline/ReFrame nearest-rank medians. Higher is better. E2E spans planned arrival to HTTP completion.

Arrival phase	Requests	TTFT p50	TPOT p50	End-to-end p50
Mixed workload 1				
Early short requests	10	2.52	2.77	2.75
Long-request burst	16	4.88	4.45	4.99
Late short requests	10	12.01	1.77	6.75
Mixed workload 2				
Early short requests	10	2.36	3.55	3.49
Long-request burst	16	5.69	4.77	6.02
Late short requests	10	26.28	1.13	6.27
vLLM 0.11 / original paired ReFrame				
Mixed workload 1				
Early short	10	9.02	6.44	6.49
Long burst	16	14.69	12.83	24.01
Late short	10	61.19	4.11	31.31
Mixed workload 2				
Early short	10	11.55	8.19	8.22
Long burst	16	16.96	14.07	28.45
Late short	10	131.70	2.47	28.43

C Matched-Batch Baseline Evaluation

The independent comparison uses fresh processes on the main-text sources and reverses method order between workloads. vLLM 0.29 uses a 16,384-token batch budget, 0.97 GPU-memory utilization, 40-GiB/GPU CPU offload, chunked prefill, and disabled prefix caching; the frozen ReFrame configuration is unchanged. All four formal cells complete 36 requests and 4,608 tokens and pass source, timing, configuration, and release checks. Table 10 gives absolute latencies and execution order.

Table 10: Qualified matched-batch comparison. TTFT starts at planned arrival; lower is better. Order is the actual sequence within each workload.

System	Order	Completion (s)	TTFT p95 (s)	TPOT p95 (ms)
Qwen3-235B-A22B-Instruct-2507; mixed workload 1				
vLLM 0.29	Second	3430.1	2260.78	9392.37
ReFrame	First	1275.8	572.50	3030.17
Qwen3-235B-A22B-Instruct-2507; mixed workload 2				
vLLM 0.29	First	3426.7	2238.11	9235.86
ReFrame	Second	1225.7	427.08	3089.41

Native configurations failed memory qualification. The qualified vLLM CPU-offload profile was fixed before the formal comparison; no partial or qualification-only request was used as a latency sample.

D Repeated Framework Comparison

Fresh processes repeat the two main-text sources with reversed method order. All six ReFrame, vLLM 0.29, and SGLang 0.5.17 cells complete 36 requests and 4,608 tokens. Table 11 adds median endpoints and actual order to the completion and p95 results in Table 2; requests within one run are descriptive observations, not independent system repetitions.

Table 11: Additional median endpoints and execution order for the repeated framework comparison. TTFT starts at planned arrival; lower is better.

System	Order	TTFT p50 (s)	TPOT p50 (ms/token)
Mixed workload 1			
vLLM 0.29	First	1,274.84	7,069.35
SGLang 0.5.17	Third	1,351.27	4,092.15
ReFrame	Second	186.16	2,031.84
Mixed workload 2			
SGLang 0.5.17	First	1,310.87	3,995.20
vLLM 0.29	Third	1,208.52	6,846.14
ReFrame	Second	44.36	2,224.18

SGLang's selected 0.92 static-memory fraction and 4,096-token prefill chunk passed qualification after larger chunks failed CUDA memory checks. The selected profile was unchanged for both formal sources.

D.1 Where the request latency is spent

The complete-system endpoints combine requests that encounter very different service conditions. We therefore decompose each cohort's mean end-to-end latency into the time before and after the first content token. For a cohort $\mathcal{C}$,

$$\overline{\mathrm{E2E}}_{\mathcal{C}} = \frac{1}{|\mathcal{C}|}\sum_{i\in\mathcal{C}}(t_i^{\mathrm{first}} - a_i) + \frac{1}{|\mathcal{C}|}\sum_{i\in\mathcal{C}}(t_i^{\mathrm{end}} - t_i^{\mathrm{first}}). \tag{7}$$

This identity permits an additive stacked plot; summing separately computed latency percentiles would not. The first interval includes prefill and any delay before the first token. The second includes subsequent serving, scheduling, and HTTP completion, so it is not a direct measure of GPU decode computation.

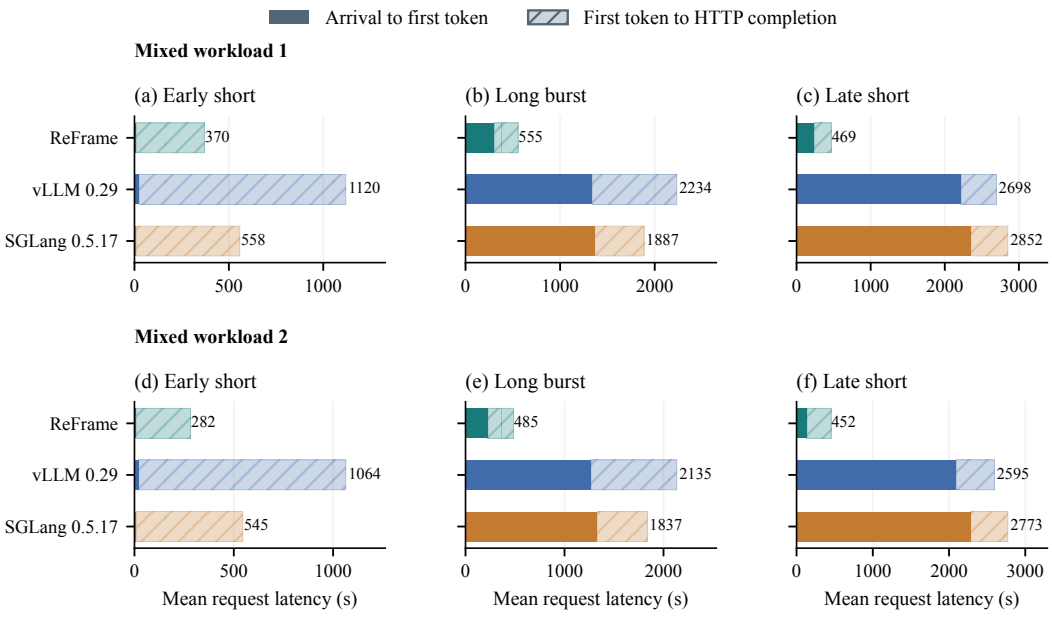

Figure 10: Mean request latency by arrival phase on the two mixed workloads. Solid segments measure planned arrival to first content token; hatched segments measure first content token to HTTP completion. End labels give the mean end-to-end latency in seconds. Each short cohort contains 10 requests and each long cohort 16; panels use different horizontal scales. These are the same six qualified runs used in the main three-framework comparison.

Early short requests spend only 1.7–2.9% of their mean end-to-end latency before the first token across the six runs, as shown in Figure 10. For example, on Mixed workload 1, ReFrame and SGLang have similar mean first-token times, 8.97 and 9.40 s, but their mean post-first-token intervals are 361.46 and 548.96 s. An early TTFT comparison alone therefore misses a substantial part of the complete-system service difference.

The long burst changes this balance. ReFrame's mean long-request TTFT is 305.75 and 226.93 s on the two sources, compared with 1,337.72/1,267.75 s for vLLM and 1,367.79/1,329.85 s for SGLang. Both latency intervals contribute to ReFrame's lower mean end-to-end time. For the late short requests, the pre-first-token interval accounts for 80.8–82.5% of baseline mean latency, versus 30.3–51.1% for ReFrame. The framework advantage thus extends to requests arriving after the burst, rather than being confined to the long requests themselves.

These observations locate the benefit along the service timeline. They do not isolate rebinding: expert supply, memory allocation, batching, and execution profiles all contribute to the complete systems. The selected same-configuration case in Appendix G examines enabling rebinding in one measured operating regime, while the zero-loan short-request experiment in Appendix E tests a regime in which no runtime borrowing occurs.

E Short-Request Framework Comparison

The supplement reuses the original 36 WildChat prompts and 15-s arrivals with fresh services and four-request warmups. Table 5 gives absolute endpoints. Paired median actual-send offset differences are 0.012 ms for vLLM 0.29 and 0.018 ms for SGLang 0.5.17, below the 20-ms limit.

ReFrame records zero runtime loans; all responses complete and the last live four-rank audit shows zero active requests or borrowed frames. This one trace per system does not estimate run-to-run variation or rebinding-component gains.

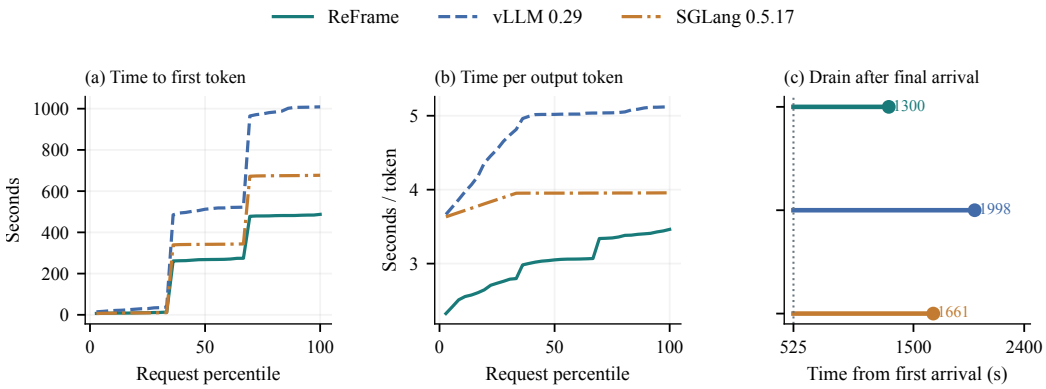

Figure 11: Short-request service without runtime borrowing. Panels (a) and (b) show the 36 request order statistics for each selected qualified run. In panel (c), colored lines span the final planned arrival at 525 s to HTTP trace completion; top-to-bottom rows are ReFrame, vLLM, and SGLang. The values are individual trace endpoints, not independent repeated-run averages.

The distributional view distinguishes first-token delay from the subsequent output pace, as shown in Figure 11. ReFrame's TTFT p95 is 483.99 s, compared with 1,008.16 s for vLLM and 676.49 s for SGLang; TPOT p95 is 3.44, 5.12, and 3.96 s per token, respectively. Subtracting the shared final arrival gives drain tails of 774.65, 1,472.60, and 1,136.05 s. The jumps in the request order statistics describe groups of requests receiving different service delays within a trace; they do not imply independent measurements of a scheduler cycle.

Because ReFrame borrows no frames in these runs, its lower endpoints demonstrate a complete-system advantage that does not require a dynamic loan on this workload. The vLLM 0.11 result in

Table 5 retains its own original ReFrame partner and is not mixed into these three-system distributions.

F Arrival Density and Long-Document Content

The supplemental traces distinguish a shorter arrival span from faster service. In addition to trace completion Z, we report the drain tail

$$D = \max_i t_i^{\text{end}} - \max_i a_i = Z - (\max_i a_i - \min_i a_i). \tag{8}$$

D is the time remaining after the final planned arrival. It is a trace-level endpoint, not a request latency percentile or a sustainable-throughput estimate.

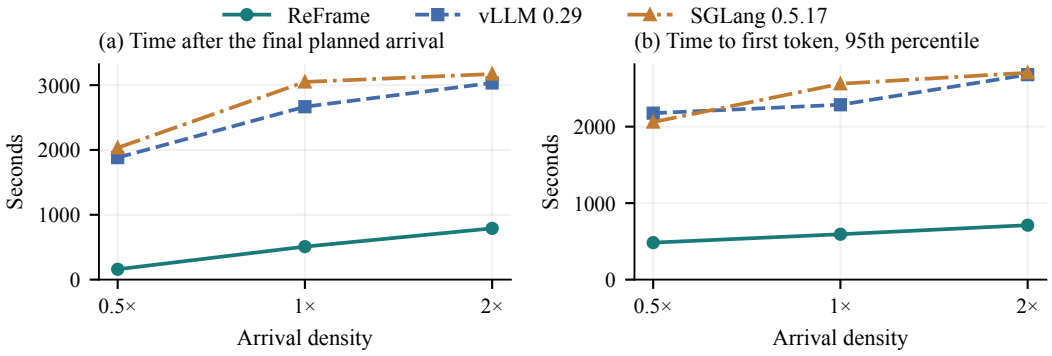

Figure 12: Finite-trace arrival-density sensitivity with all nine qualified system–density combinations. Lines connect measured endpoints. Prompt text and output lengths are fixed while all planned arrival times are divided by the density factor. Lower is better.

ReFrame retains lower completion time and latency tails at every tested density, as shown in Figure 12 and Table 12. Its completion speedup ranges from 1.98× to 2.89× over vLLM and from 2.07× to 3.00× over SGLang. ReFrame still has the shortest drain tail at each density; at 2×, the baselines require 3,034.05 and 3,171.60 s after the final arrival. These finite traces characterize sensitivity to burst compression, rather than a maximum sustainable arrival rate.

Table 12: Qualified arrival-density endpoints in seconds, all with 36 completed requests and 4,608 output tokens. Bold marks the lowest endpoint at each density. The final planned arrivals are 1,590, 795, and 397.5 s at 0.5×, 1×, and 2×.

Arrival density	System	Completion	Drain tail	TTFT p95	TPOT p95
0.5×	ReFrame	**1749.54**	**159.54**	**483.43**	**2.40**
0.5×	vLLM 0.29	3472.19	1882.19	2176.38	9.11
0.5×	SGLang 0.5.17	3623.36	2033.36	2060.20	4.09
1×	ReFrame	**1303.88**	**508.88**	**593.20**	**3.11**
1×	vLLM 0.29	3462.06	2667.06	2286.40	9.47
1×	SGLang 0.5.17	3844.38	3049.38	2560.49	4.49
2×	ReFrame	**1188.00**	**790.50**	**711.77**	**3.21**
2×	vLLM 0.29	3431.55	3034.05	2679.82	8.87
2×	SGLang 0.5.17	3569.10	3171.60	2704.83	4.14

Content at approximately fixed input volume. Replacing the 16 reports with SummScreenFD episode transcripts changes the mixed trace from 205,593 to 205,597 input tokens, while preserving all short prompts and arrivals (Appendix A.3). ReFrame's qualified completion and drain-tail endpoints are 1,102.06 and 307.06 s. The corresponding pairs are 3,361.54/2,566.54 s for vLLM and

3,902.10/3,107.10 s for SGLang. The complete-system advantage therefore also appears with conversational long-document content at essentially the same input volume. This comparison broadens the input domain; it does not estimate a content-independent rebinding effect or separate the contribution of different expert-routing patterns.

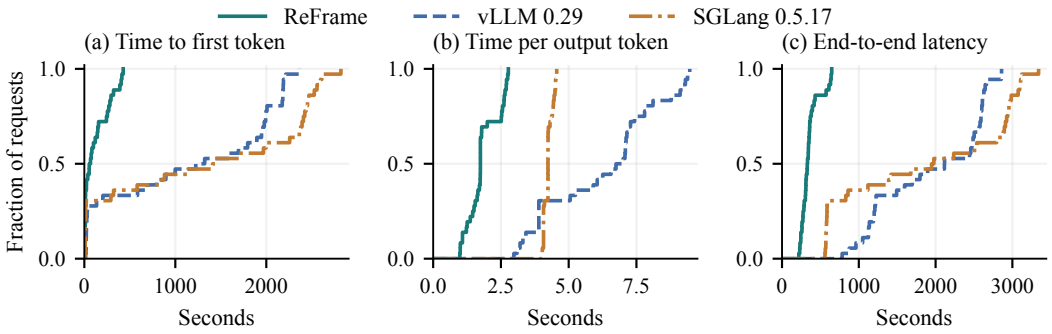

Figure 13: Request latency distributions for the mixed SummScreenFD/WildChat workload. Each empirical curve contains the 36 requests from one qualified system run. TTFT and end-to-end latency include waiting from planned arrival; output-token intervals use 128-token generations. Curves summarize request heterogeneity, not independent system repetitions.

The latency distributions extend the trace-level comparison to individual requests, as shown in Figure 13. ReFrame reduces TTFT p95 to 412.76 s from 2,213.67 s with vLLM and 2,610.67 s with SGLang; TPOT p95 is 2.75 s, compared with 9.39 and 4.53 s. Completion speedups are $3.05\times$ and $3.54\times$, respectively. The improvements therefore span arrival-to-first-token latency, output-token intervals, and overall completion for this source. As in the main framework comparison, these are complete-system results under the stated resource profiles.

Additional latency endpoints. Table 13 retains medians, mean end-to-end latency, and drain-tail endpoints for all twelve runs, complementing the plotted tail latencies.

Table 13: Supplementary latency metrics for the three-system extension; lower is better and bold marks each condition's minimum. Main-text tables and figures report completion and tail latencies.

System	TTFT p50 (s)	TPOT p50 (ms/token)	E2E p50 (s)	E2E p95 (s)	Mean E2E (s)	Drain tail (s)
SummScreenFD + WildChat						
vLLM 0.29	1,224.80	6,745.17	2,115.00	2,855.64	1,943.90	2,566.54
SGLang 0.5.17	1,410.00	4,235.80	1,947.93	3,122.10	1,856.42	3,107.10
ReFrame	**54.08**	**1,741.93**	**328.57**	**639.40**	**358.38**	**307.06**
Mixed workload 1, $0.5\times$ *density*						
vLLM 0.29	1,159.55	5,523.81	1,882.19	2,844.55	1,651.68	1,882.19
SGLang 0.5.17	1,042.84	3,729.07	1,516.57	2,526.18	1,460.96	2,033.36
ReFrame	**11.59**	**1,692.34**	**271.25**	**609.97**	**331.32**	**159.54**
Mixed workload 1, $1\times$ *density*						
vLLM 0.29	1,248.99	6,901.07	2,182.44	2,954.62	2,005.45	2,667.06
SGLang 0.5.17	1,384.28	4,175.05	1,914.41	3,064.38	1,825.50	3,049.38
ReFrame	**201.51**	**2,039.72**	**453.80**	**871.76**	**497.00**	**508.88**
Mixed workload 1, $2\times$ *density*						
vLLM 0.29	1,280.33	6,583.30	2,212.78	3,051.53	2,100.63	3,034.05
SGLang 0.5.17	1,418.91	3,845.27	1,907.02	3,171.60	1,784.49	3,171.60
ReFrame	**379.37**	**1,931.98**	**576.49**	**964.95**	**614.64**	**790.50**

Coverage of these views. The sealed supplemental matrix contains nine arrival-density runs and three SummScreenFD runs: 432 completed requests and 55,296 generated tokens. All displayed endpoints were recomputed from the raw streaming clocks with frozen input identities and matched planned arrivals. Each system–condition combination has one run. The deferred 256-token comparison is excluded from these views; earlier failed attempts remain separate in the evidence archive.

Newly scheduled vLLM 0.11 supplements are also excluded until qualified terminal records are available. The earlier vLLM 0.11 comparisons retain their original paired ReFrame references.

G Selected Rebinding Case and Runtime Evidence

The retrospectively selected pairs are shown in Figure 8. Table 14 adds whole-run medians and borrowing counts for the selected same-configuration pair. Both arms complete 36 requests and 4,608 tokens. Selection by long-request TTFT does not estimate a typical effect.

Table 14: Whole-run medians and borrowing for the selected same-configuration pair on mixed workload 1, second repetition.

Method	TTFT p50 (s)	TPOT p50 (ms/token)	Loans	Completed
ReFrame without rebinding	264.18	2,239.24	0	36/36
ReFrame	189.16	2,083.80	1	36/36

The selected same-configuration pair lowers paired long-request TTFT by 12.17%, whole-run TPOT p95 by 10.69%, and completion by 5.55%. The complete repeated-pair and fixed-partition analyses are retained in the full experimental version.

G.1 Observed borrowing and restoration

Across fourteen qualified ReFrame runs, each borrowing run returns its frame on every rank, restores the initial 1,786 Expert frames per rank, and records zero loan-owner preemptions. The sampled cycle and repeated preemption counts are shown in Figure 14. Endpoint equality establishes closure for these runs, not continuous per-frame identity between samples.

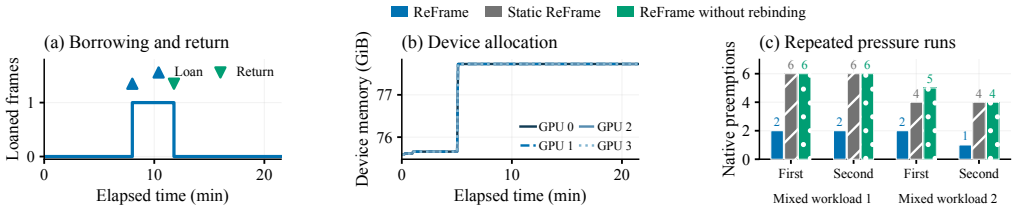

Figure 14: Ownership and device-memory samples in the first mixed-workload repetition (a,b), with internal native request-preemption counts across four repeats (c).

The four same-configuration pairs record 6, 6, 5, and 4 native preemptions without rebinding, versus 2, 2, 2, and 1 with rebinding. All four dynamic runs restore the initial Expert capacity and have zero loan-owner preemptions. These counters establish the observed capacity-return behavior; the latency endpoints remain separate measurements.

H Supporting Runtime Definitions

The definitions below support the propositions and timing interpretation in the main text. Detailed experiment construction and measured service behavior appear first in this supplement.

Frame accounting and eligibility. For an arena of N frames of F bytes, Expert-owned, KV-owned, free, and in-transit frames form a disjoint partition:

$$E(t) + K(t) + U(t) + T(t) = N. \tag{9}$$

Other weights and workspaces remain outside this arena but count toward device memory. KV ownership includes both occupied and reusable backing. A native block is eligible only when it is free, every backing frame is ready, and the request has the authorization in Eq. 1:

$$b \in \mathcal{B}_{\text{free}}(t) \ \wedge \ \mathcal{F}(b) \subseteq \mathcal{K}_{\text{ready}}(t). \tag{10}$$

A frame in transit is unavailable to both consumers. Retirement revokes the source publication before destination preparation; publication then validates destination readiness and exposes a fresh owner–generation pair. Expert restoration reconstructs weights from immutable host backing. KV readiness permits native writes, and attention reads additionally require populated token state. Live references protect their backing frames throughout use. Recovery follows the same transition rules or stops the affected execution.

Timing and internal controls. With admission time t_i^{adm}, the first-token latency satisfies

$$Y_i = (t_i^{\text{adm}} - a_i) + (t_i^{\text{first}} - t_i^{\text{adm}}) = W_i^{\text{admission}} + P_i^{\text{first}}. \tag{11}$$

The second interval includes scheduling, execution, and delivery. The experiments measure complete TTFT rather than these admission intervals separately. Static ReFrame retains a shadow controller; ReFrame without rebinding matches the dynamic configuration but disables borrowing. Both retain expert supply. The minimum-prefix and private-first properties concern a fixed allocation state; the following proofs state their precise assumptions.

I Proofs for Sections 3–4

We prove the accounting, ownership, and allocation properties under the assumptions stated in Sections 3–4.

I.1 Frame accounting and latency decomposition

Let $\mathcal{G}$ be the finite arena frame set, $|\mathcal{G}| = N$. At every valid state, its Expert-owned, KV-owned, free, and in-transit subsets form a disjoint partition. Taking cardinalities gives $E(t) + K(t) + U(t) + T(t) = N$; multiplying by the fixed frame size F gives the arena capacity NF. Publication and retirement move a frame between these states without creating or removing a frame. KV-owned capacity includes reusable and occupied backing, so $K(t)F$ is not an identity for the live-token footprint.

For a completed request, subtraction of timestamps gives

$$(t_i^{\text{adm}} - a_i) + (t_i^{\text{first}} - t_i^{\text{adm}}) = t_i^{\text{first}} - a_i = Y_i.$$

For ordered timestamps, both intervals are nonnegative. If an actual admission precedes the planned arrival, the algebra still holds but W_i is a signed scheduling offset rather than a nonnegative wait. Comparing the same request and arrival reference in the two executions gives $\Delta Y_i = \Delta W_i + \Delta P_i$ by subtraction. Subtracting ΔW_i from both sides of $\Delta Y_i < 0$ gives Eq. 4. The identity partitions elapsed time at admission; complete TTFT is the measured endpoint.

I.2 Proof of Proposition 1: ownership and live-reference protection

Fix $f \in \mathcal{G}$. A publication is an owner–generation pair (o, g) with $o \in \{\text{Expert}, \text{KV}\}$ and $g \in \mathbb{N}_0$. Generations are not reused for a frame. Initially, $|\Pi_0(f)| \leq 1$, and every protected use refers to that frame's valid publication. The permitted state transitions are:

- Quiescence and completion of protected uses leave $\Pi_n(f)$ unchanged. Acquiring a use atomically checks the current publication and readiness and registers protection before the use begins.
- Retirement atomically checks $\mathcal{R}_n(f) = \varnothing$ and sets $\Pi_{n+1}(f) = \varnothing$. It cannot race past a registered use.
- Materialization requires an unpublished frame and leaves its publication empty. Destination data are not exposed while they are being prepared.
- Publication requires $\Pi_n(f) = \varnothing$ and destination readiness, then sets $\Pi_{n+1}(f) = \{(o, g)\}$ for a fresh generation. Expert readiness includes reconstructed weights; KV readiness permits native writes, while attention reads additionally require populated token state.

Per-frame ownership transitions are serialized. By induction on $n \in \mathbb{N}_0$, quiescence and use operations preserve the bound on publication cardinality; retirement and materialization give cardinality zero; publication gives cardinality one. Hence $|\Pi_n(f)| \leq 1$ for every reachable finite state. Validation against the current pair rejects a retired generation, and a successful acquisition remains protected until completion.

For a live KV reference q to block b, the protection rule registers $q \in \mathcal{R}_n(f)$ for every $f \in \mathcal{F}(b)$ and retains this protection while q is live. Consequently $\mathcal{R}_n(f) \neq \varnothing$, and the retirement guard fails on each backing frame. Releasing the last reference removes this obstruction but does not itself reconstruct or publish an Expert owner. Recovery obeys the same transition relation or terminates the affected execution; it therefore preserves the invariant. The invariant holds for every finite sequence of permitted transitions. □

I.3 Candidate-prefix feasibility and minimality

Fix a request i, a snapshot t, the free block identities, the backing map, and the existing publications and leases. Candidate frames $f_1, \ldots, f_r$ are distinct borrowable frames outside the request-visible KV pool and can be assigned to i. Any Expert-owned candidate must first complete a valid transition; an already ready KV frame requires only authorization. Here $r \in \{0, 1, 2\}$ is capped by both the remaining episode allowance and the number of available candidates. Define $C_0 = \varnothing$ and $C_k = \{f_1, \ldots, f_k\}$. Holding all other state fixed, the hypothetical eligible set is

$$\widehat{\mathcal{A}}_i(C_k, t) = \{b \in \mathcal{B}_{\mathrm{free}}(t) : \mathcal{F}(b) \subseteq \mathcal{K}_{\mathrm{ready}}(t) \cup C_k, \quad \forall f \in \mathcal{F}(b) \setminus C_k,\ \lambda_t(f) \in \{\bot, i\}\}\,.$$

All frames in C_k are hypothetically ready and leased to i. A block spanning several frames is admitted only if each satisfies readiness and authorization. Since $C_k \subseteq C_{k+1}$, any block in $\widehat{\mathcal{A}}_i(C_k, t)$ remains in $\widehat{\mathcal{A}}_i(C_{k+1}, t)$; no previously valid backing or authorization is removed. Thus the eligible sets, and their cardinalities, are nondecreasing in k.

If the feasible index set in Eq. 2 is nonempty, it is a finite subset of $\{1, \ldots, r\}$ and has a least element k^*. Its definition ensures that k^* satisfies the demand and that every positive $k < k^*$ fails it. For the capacity-shortage branch, $|\mathcal{A}_i(t)| < d_i(t)$, so $k = 0$ also fails. If $r = 0$ or no prefix satisfies the demand, no loan is selected. The selected prefix is minimal within the fixed candidate order at that snapshot. If free blocks, readiness, or leases change before allocation, native eligibility is revalidated before the selected blocks are used. □

I.4 Proof of Proposition 2: minimum shared consumption

Fix the request and allocation state. Assume the finite disjoint eligible sets $\mathcal{P}$ and $\mathcal{S}$ contain independently selectable native blocks for this allocation. Write $p = |\mathcal{P}|$, $s = |\mathcal{S}|$, and let integer demand satisfy $0 \leq d \leq p + s$. Every feasible selection $X \subseteq \mathcal{P} \cup \mathcal{S}$ with $|X| = d$ obeys

$$|X \cap \mathcal{S}| = d - |X \cap \mathcal{P}| \geq d - p, \qquad |X \cap \mathcal{S}| \geq 0.$$

Therefore shared consumption is at least $\max(0, d - p)$. Select $\min(d, p)$ private blocks and $d - \min(d, p) = \max(0, d - p)$ shared blocks. This selection exists: when $d \leq p$, no shared blocks are needed; when $d > p$, feasibility gives $d - p \leq s$. Private-first attains the bound, leaving $s - \max(0, d - p)$ shared blocks. The construction includes $d = 0$, $p = 0$, and $s = 0$. It maximizes the number of shared blocks left in this fixed allocation state. □