BidKV: Utility-Guided Preemption Scheduling for KV-Pressure LLM Serving

Anonymous Authors
Anonymous Institution

Abstract

When KV-cache demand exceeds GPU memory capacity during online LLM serving, serving engines reclaim cache from active requests to admit waiting ones. Poor victim selection wastes recomputation, delays admission of queued requests, and degrades first-token latency—a metric directly visible to users. Existing policies rely on coarse request-order heuristics (LIFO, FCFS) or independent per-request scoring rules, lacking both explicit reclamation-cost signals and cross-request coordination. We present BidKV, a utility-guided reclamation policy that improves *admission responsiveness* under KV pressure. Each active request produces a bid encoding recoverable capacity and estimated disruption cost; at each pressure event, an online utility-ranked selection policy identifies the intended highest-utility victim and preempts it via the framework's native mechanism. The current prototype instantiates disruption cost from request-lifecycle proxies (completion progress, prompt length, preemption history) under recompute-fallback semantics. BidKV integrates non-invasively with vLLM and SGLang via a portable adapter layer. Evaluated with Llama-3.1-8B-Instruct on an NVIDIA RTX A6000 under ShareGPT traces, BidKV reduces P95 TTFT by 89% and improves 300 ms SLO attainment by 14.9 pp over vLLM's native policy, at a modest throughput cost.

1 Introduction

Online large language model (LLM) serving must sustain a continuously changing population of concurrent requests on limited GPU memory. In production workloads, requests are highly heterogeneous in prompt length, generation budget, and arrival pattern; they also progress asynchronously as some requests are still in prefill while others are already deep into decoding [11, 22, 24]. Under such mixed-length, bursty workloads, the key-value (KV) cache becomes a dominant runtime resource [16, 19]: new arrivals allocate KV state, ongoing requests expand it token by token, and completed or preempted requests release it at different points in their lifecycle. Aggregate KV demand therefore fluctuates on short timescales and routinely approaches hardware capacity limits; modern serving engines reclaim KV state from active requests—triggered at a high-watermark threshold before saturation—to preserve scheduling flexibility and ensure that low-cost victim choices remain available as new requests arrive.

The central question in this setting is not whether KV capacity must be reclaimed, but which active request should be chosen as the *victim*. Two requests may occupy comparable amounts of KV memory while imposing very different costs when interrupted. Under the *recompute-fallback* execution model used by current serving engines [11], reclaiming an active request releases its KV state but forces the request to restart through the framework's native recovery path when it is rescheduled. The cost of reclamation therefore depends not only on how much memory is freed, but also on how much useful computation is discarded and must later be redone. Reclaiming a request that is near completion can waste substantial prior work and incur expensive recomputation, whereas reclaiming a request that has only recently entered decoding may free similar capacity at much lower system disruption. Poor victim choices can also cascade: recomputation re-inflates KV demand, which in turn triggers further reclamation—a cycle that keeps KV capacity occupied by recomputing victims rather than available for waiting requests. The direct consequence is admission-latency degradation: queued requests wait longer to receive their first token, driving up time-to-first-token (TTFT) and eroding service-level objective (SLO) attainment. Active-KV reclamation is therefore a cross-request victim-selection problem whose quality directly governs admission responsiveness under sustained KV pressure.

Because reclamation costs are heterogeneous, the scheduler needs *explicit per-request cost signals* to distinguish cheap victims from expensive ones; because selection is cross-request, these signals must feed into *coordinated cross-request selection* rather than independent per-request rules. No existing system provides both. Current scheduling heuristics—LIFO preemption [11], FCFS admission [22], EDF-style waiting policies [8]—determine victim order without any per-request cost signal; broader serving innovations such as chunked prefills [1] and prefill–decode disaggregation [15, 26] advance throughput and resource placement but inherit the same order-based victim selection, leaving the signal gap intact. Token-level scoring methods such as H2O [23] and StreamingLLM [21] do produce request-specific importance signals, yet each applies a fixed scoring rule independently—without coordination across the batch, a locally reasonable per-request choice can be globally costly. What is missing is a *signal pathway*: a structured interface through which per-request reclamation sensitivity can be surfaced to the scheduler and used for coordinated cross-request victim selection.

In this study we present BidKV, a *bid-based scheduling abstraction* for active-KV reclamation. The key idea is to make reclamation sensitivity an explicit, structured signal rather than an implicit consequence of scheduling order. Each active request produces a *bid* that summarizes the KV capacity recoverable upon reclamation and a surrogate disruption estimate reflecting the relative penalty of reclaiming that request. Given bids from all active requests, the scheduler performs an online utility-ranked cross-request victim selection: at each pressure event, the request offering the most recoverable KV capacity per unit of estimated disruption is selected as the intended victim and preempted via the framework's native mechanism. In this way, BidKV turns victim selection from an opaque runtime behavior into an explicit interface between a *scoring layer*—which estimates per-request reclamation sensitivity—and a *coordinated selection layer*—which ranks candidates by cross-request reclamation utility.

In the current prototype, the disruption estimate is instantiated from request-lifecycle features (completion progress, prompt length, preemption history) under recompute-fallback semantics [11]. BidKV integrates non-invasively with both vLLM [11] and SGLang [25] via a portable adapter layer that fully reuses each framework's native reclamation and recovery paths—the fact that both integrations require no source-code modification confirms that the bid interface is cleanly separated from framework-specific scheduling internals. BidKV controls *which* request is reclaimed, not *how*. Under recompute-fallback execution, the policy does not alter final outputs; the differentiation lies entirely in victim selection. The primary benefit is improved admission responsiveness: by selecting victims whose reclamation frees adequate capacity at low reclamation cost, BidKV reduces the time waiting requests spend queued for KV allocation, yielding lower TTFT and higher SLO attainment. This comes at a modest cost in throughput and per-token decode latency, a tradeoff that favors latency-sensitive deployments. The benefit is operating-regime dependent: under sustained mixed-length KV pressure, utility-guided selection provides substantial gains; under low pressure, victim selection remains governed by the framework's native ordering, with BidKV's reorder activating only under sustained high-pressure conditions.

In summary, this paper makes the following contributions:

(1) **Bid-based reclamation interface.** We introduce a structured bid abstraction that allows per-request reclamation sensitivity to be surfaced to the runtime scheduler, replacing implicit victim ordering rules with an explicit decision interface.
(2) **Decoupled scoring-to-selection architecture.** We separate request-level disruption estimation from coordinated cross-request victim selection through a structured bid interface, so that selection operates on uniform cost–capacity pairs regardless of how disruption estimates are produced.
(3) **Portable runtime integration.** We realize BidKV as a non-invasive scheduling layer with a `FrameworkAdapter` abstraction, demonstrating integration with vLLM and SGLang that reuses each framework's native reclamation and recovery mechanisms without source-code modification.
(4) **Structured evaluation under controlled conditions.** We evaluate five strategies that vary across scheduling dimensions (admission ordering, running-queue reordering, victim selection logic, and proactive reclamation) under frozen request traces and calibrated arrival rates, supporting structured attribution across these co-varying dimensions.

2 Background and Motivation

2.1 KV Cache Memory Model

Each autoregressive decoding step caches key and value projections [19] for all previously generated tokens. For a model with L layers, H KV heads (via grouped-query attention [2]), and head dimension d_h, a single request of length n stores $2LHd_hn$ elements. With bf16 precision, Llama-3.1-8B-Instruct (L=32, H=8_{kv}, d_h=128) allocates 128 KiB per token [7]. vLLM [11] manages this state through PagedAttention: physical GPU memory is partitioned into fixed-size *blocks* that are mapped to logical positions via a block table.

At this per-token cost, a single 4 096-token request already consumes ∼512 MiB of KV state. Total KV demand scales as $O(B \cdot \bar{n})$, where B is the batch size and $\bar{n}$ the mean sequence length, while the device memory available for KV allocation is fixed after model weights, activation buffers, and runtime overhead are subtracted. Concurrency and sequence length therefore compete for the same finite pool, and production-level serving loads routinely drive KV utilization to near-saturation, forcing the engine to *reclaim* KV state from active requests. The quality of this reclamation decision—*which* requests to evict—directly affects tail latency and throughput, as we formalize next.

2.2 The Victim-Selection Problem

Under recompute-fallback semantics [11], each reclamation event releases the victim's KV blocks and re-queues the request; when rescheduled, its prompt is recomputed from scratch. The scheduler's task at each such event is to choose a *victim set* $V \subseteq \mathcal{R}$ from the set of active requests $\mathcal{R}$ such that the total freed capacity meets the shortfall while minimizing aggregate reclamation cost:

$$\min_{V \subseteq \mathcal{R}} \sum_{i \in V} c_i \quad \text{s.t.} \quad \sum_{i \in V} r_i \;\geq\; N, \tag{1}$$

where r_i is the KV capacity freed by reclaiming request i, c_i is the reclamation cost—encompassing discarded computation, requeuing delay, and any cascading pressure from recomputed requests re-entering the running queue—and N is the capacity shortfall. Under recompute-fallback execution, the dominant component of c_i is the work already invested in the victim; Section 4.2 instantiates c_i with a surrogate derived from request-level lifecycle features. Minimizing aggregate reclamation cost is monotonically connected to admission responsiveness: cheaper victims keep less KV capacity occupied by non-productive re-prefill recomputation, accelerating queue drainage and shortening the wait before queued requests are admitted.

This formulation is an instance of the minimum-cost covering problem [10]. With heterogeneous requests, the choice of V matters: two requests may free comparable capacity yet differ by an order of magnitude in reclamation cost. Existing engines do not solve this selection problem explicitly. vLLM selects the most recently scheduled request (LIFO) [11]; other systems apply FCFS [22] or EDF ordering [8], none of which accounts for per-request reclamation cost. This analysis motivates two requirements. First, the scheduler should accept *explicit per-request reclamation-cost signals* that capture heterogeneity across candidates, rather than relying solely on arrival or scheduling order. Second, victim selection should perform *cross-request coordination*—considering the batch-level trade-off between capacity freed and cost incurred—rather than scoring each request in isolation.

Execution model. Throughout this paper we adopt the standard recompute-fallback execution model used by current serving engines [11]: reclaiming a request releases its KV blocks and re-queues it for full prompt recomputation when rescheduled; the reclamation policy does not alter final output tokens.

Empirical evidence. Figure 1 illustrates the victim-selection gap at a KV-pressure snapshot from a long-context ShareGPT trace (rate=0.5 req/s; the same effect arises under mixed-length workloads at smaller absolute scale, Section 6). Panel (a) shows 10 concurrent requests at peak KV utilization, sorted by tokens held. LIFO targets

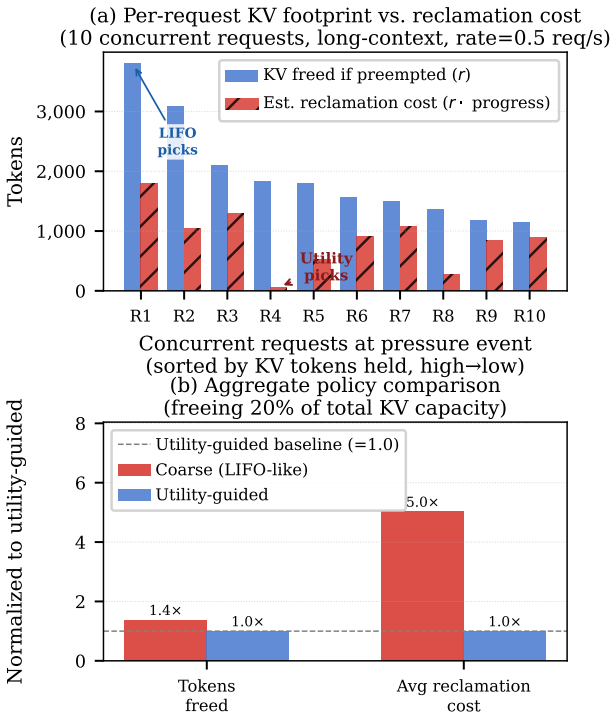

Figure 1: Empirical motivation (long-context trace, ShareGPT, rate=0.5 req/s). (a) Per-request KV footprint (blue) and estimated reclamation cost (red hatched) at a KV-pressure snapshot; 10 concurrent requests sorted by KV tokens held, high→low. LIFO targets R1–R2 (34–47% complete); utility-guided selects R4 (3% complete, 29× lower reclamation cost). (b) Aggregate comparison freeing 20% of total KV capacity: LIFO imposes 5.0× higher average reclamation cost per event relative to utility-guided selection.

R1 and R2—the requests with the largest KV footprints—yet these requests are 47% and 34% complete, with estimated reclamation costs of ∼1,800 and ∼1,050 tokens of discarded work respectively (red hatched bars). A utility-guided policy instead selects R4, which is only 3% complete: it frees 1,825 tokens while incurring a reclamation cost of just 62 tokens—a 29× improvement in cost efficiency for this single victim choice.

Panel (b) aggregates the comparison over a target of freeing 20% of total KV capacity. LIFO recovers 1.4× more tokens per event (by preferring larger requests), but at a 5.0× higher average reclamation cost. Coarse victim selection trades marginal extra KV recovery for a disproportionate cost burden, compounding tail-latency degradation at every subsequent scheduling step.

3 Related Work

LLM serving systems. vLLM [11] introduced PagedAttention and handles KV pressure through request-level preemption (swap or recompute). Orca [22] pioneered iteration-level scheduling for continuous batching; DistServe [26] and Splitwise [15] disaggregate prefill and decode phases; Sarathi-Serve [1] introduces chunked prefills to reduce pipeline bubbles. FlexGen [17] offloads KV state to CPU/disk to extend effective capacity, while LoongServe [20] applies elastic sequence parallelism for long contexts. Prompt Cache [5] and Hydragen [9] exploit prefix sharing to avoid redundant KV computation. These systems advance batching, placement, and memory management along complementary dimensions, yet none treats victim selection as a first-class optimization target: systems that implement preemption still default to order-based heuristics (LIFO or FCFS) that do not incorporate per-request reclamation-cost signals.

Token-level KV cache compression. H2O [23] identifies "heavy-hitter" tokens via cumulative attention scores. StreamingLLM [21] preserves attention-sink tokens for infinite streaming. Scissorhands [13] exploits persistence-of-importance. SnapKV [12] clusters attention features to compress prefix cache. KIVI [27] applies asymmetric 2-bit KV quantization. PyramidKV [3] assigns layer-adaptive budgets via pyramid allocation. These methods effectively reduce per-request KV footprint, but each operates within a single request: the retention rule is applied independently of the rest of the batch. In their standard formulations, they do not coordinate across requests to decide *which request* should yield capacity, nor do they expose reclamation-cost information to the scheduler.

Scheduling theory and multi-resource allocation. Dominant Resource Fairness [4] and multi-resource packing [6] formalize allocation for heterogeneous cluster resources. The knapsack literature [10, 14] provides algorithmic foundations for constrained subset selection—the same combinatorial structure that underlies the idealized victim-selection formulation (Eq. 1). These frameworks assume that item costs are given inputs and that an offline solver is available. In LLM serving, per-request reclamation costs must be *estimated* at runtime from serving state, and selection must complete within the engine's scheduling loop under strict latency constraints. Rather than solving the combinatorial problem exactly, BidKV uses an online utility-ranked approximation: requests are sorted by reclamation utility at each scheduling step, and the top-ranked entry is consumed at each pressure event (Section 4.3).

Positioning. Existing serving systems implement the preemption mechanism but select victims by scheduling order; token-level compression methods produce per-request importance signals but apply them independently within each request. To our knowledge, no prior system combines explicit per-request reclamation-cost signals with coordinated cross-request victim selection—the two requirements identified in Section 2.2. BidKV fills this gap with a scorer-agnostic, framework-portable interface: per-request signals—whether derived from token-level scorers or request-lifecycle features—are structured as bids, and a utility-ranked online selection pass coordinates the choice across the batch (Section 4).

4 Design

The victim-selection problem identified in Section 2 imposes two requirements: explicit reclamation-sensitivity signals and cross-request coordination. Two additional design goals guide our architecture: *scorer-agnostic interface*—because the best importance

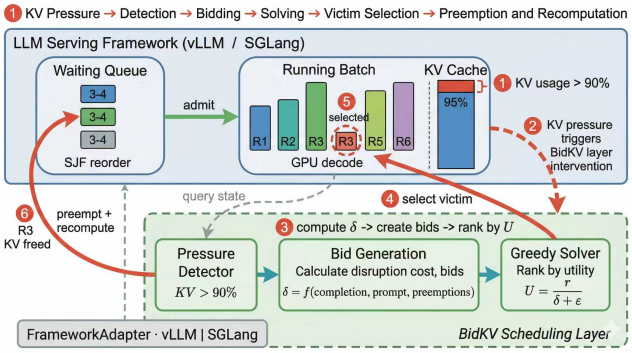

Figure 2: BidKV scheduling overview. When KV usage exceeds the pressure threshold (①), the scheduling layer activates (②): the pressure detector triggers bid generation, which computes disruption cost δ for each running request and ranks candidates by utility $U = r/(\delta + \varepsilon)$ (③). The selection layer identifies the top-ranked victim (④); the framework adapter preempts the chosen request, freeing its KV blocks (⑤). The preempted request re-enters the waiting queue for later recomputation (⑥).

signal may vary across workloads and models, the scoring strategy should be interchangeable without modifying the selection algorithm or the runtime integration; and *framework portability*—the mechanism should integrate with heterogeneous serving engines (*e.g.*, vLLM, SGLang) by reusing their native reclamation and recovery paths, without requiring source-code modifications.

These goals separate naturally into a *general architecture*—the bid-based data path and layered decomposition that are execution-model agnostic—and a *current instantiated path*—the specific disruption estimator (δ, Eq. 3) that uses request-lifecycle features under recompute-fallback semantics. The architecture prescribes how scoring signals flow to the selection layer; the instantiation determines which features populate δ in this paper.

BidKV organizes its logic into four layers (Figure 2): the *Runtime Adapter Layer* detects KV pressure and executes reclamations via the framework's native mechanism; the *Bid Generation Layer* computes disruption cost from request-lifecycle features and produces structured bids; the *Utility-Ranked Selection Layer* produces a cached reclamation-priority ordering; and the *Bid Signal Layer* defines the shared data structures used across all layers.

4.1 Bid Signal Layer

The atomic unit of communication is the *scheduling bid*, a frozen (immutable) data class with the following fields:

- `bid_id` — unique identifier.
- `request_id` — the inference request this bid applies to.
- r (`tokens_freed`) — the number of KV tokens freed if this request is preempted.
- δ (*disruption estimate*) — a surrogate scheduling cost (≥ 0) estimating the system-level disruption of reclaiming this request. Lower values indicate requests whose preemption incurs less scheduling disruption. Under recompute fallback (this work), δ is derived from request-level features (completion progress, prompt length, and preemption history; Eq. 3); other request-level scorers can populate δ without architectural change.
- `confidence` $\in [0, 1]$ — the scorer's confidence in δ.
- `metadata` — strategy-specific information.

Each bid carries a derived *utility* property:

$$U(b) = \frac{b.r}{b.\delta + \varepsilon}, \qquad \varepsilon = 10^{-3}. \tag{2}$$

A high utility ratio means preempting this request frees many tokens at low reclamation cost. By preferring high-utility victims, the selection policy reduces aggregate reclamation cost, keeping KV capacity available for queued requests and thereby improving admission responsiveness (TTFT and SLO attainment).

Bids are collected in a `BidPool`—an immutable snapshot of all active bids at a given instant. The selection output is a utility-ranked ordering of active bids; the runtime adapter consumes the current top-ranked entry at each pressure event.

4.2 Bid Generation Layer

The bid generation layer translates per-request state into a structured bid. The choice of features that populate δ is an instantiation decision; this paper uses request-lifecycle proxies under recompute-fallback semantics, as detailed below. Each running request contributes one bid with r equal to its current KV footprint and δ computed as follows. Let $\hat{r} = \max(\beta,\ n_{\text{prompt}}/n_0)$ be the prompt-normalized recompute cost (floored at β to prevent very short prompts from appearing free to evict). The disruption estimate sums three additive terms:

$$\delta = \max\Big(\delta_{\min},\ \underbrace{\hat{r}}_{\text{recompute}} + \underbrace{w_c\, c^2}_{\text{completion}} + \underbrace{w_s\, P}_{\text{starvation}}\Big), \tag{3}$$

where $c = \min(1,\ n_{\text{output}}/n_{\max})$ is the completion ratio and P counts prior preemptions.

All three terms share the same unit: *recompute cost of an n_0=256-token prompt* (i.e. $\hat{r}$=1 at baseline). Under this unit, w_c=2.0 encodes that a fully completed request is as disruptive to evict as re-prefilling a 512-token prompt; w_s=0.5 adds half a baseline recompute cost per prior preemption. The floor β=0.5 ensures even trivially short prompts carry non-trivial reclamation cost. The completion term is *quadratic*: evicting a request at 90% completion wastes four times more accumulated decode work than one at 45%, reflecting the sharply increasing value of near-complete requests. Because δ functions purely as a *ranking signal*, the ranking procedure requires only that the relative ordering be preserved; exact magnitudes need not be calibrated to absolute costs (Section 7).

4.3 Online Utility-Ranked Victim Selection

Equation 1 states the idealized batch reclamation objective; practical serving requires an online approximation that fits within the engine's tick-based scheduling loop. At each scheduling step, the `GreedyBidSolver` computes a reclamation utility score for every active bid and produces a utility-ranked ordering of all candidates (Algorithm 1). This ordering is cached and refreshed periodically; at each pressure event, the runtime adapter consumes the current top-ranked entry and executes one native preemption.

The utility score is drawn directly from Eq. 2: $U = r/(\delta + \varepsilon)$. Higher utility means the request frees more KV capacity per unit

Algorithm 1 Utility-Ranked Victim Ordering

Require: Bid pool $\mathcal{P} = \{b_1, b_2, \ldots, b_B\}$
Ensure: Utility-ranked ordering O
1: **for all** $b \in \mathcal{P}$ **do**
2: $\quad U(b) \leftarrow \dfrac{b.r}{b.\delta + \varepsilon}$ ▷ reclamation utility, $\varepsilon = 10^{-3}$
3: **end for**
4: Sort $\mathcal{P}$ by $U(b)$ in decreasing order $\rightarrow O$
5: **return** O ▷ cached; adapter pops top entry at each pressure event

of estimated reclamation disruption; the top-ranked request is the intended highest-utility victim. Because δ serves as an ordinal ranking signal, the selection requires only that relative ordering be preserved across bids—exact magnitude calibration is not needed.

The ranking procedure runs in $O(B \log B)$ time. In practice $B < 100$, so ranking takes microseconds—negligible compared with a decode step.

4.4 Runtime Adapter Layer

To achieve framework portability, BidKV defines a `FrameworkAdapter` ABC with five responsibilities:

(1) **KV stats visibility**: returns current and maximum token counts for pressure detection.
(2) **Pressure-event interception**: intercepts the framework's native preemption path and invokes BidKV's solver first.
(3) **Reclamation execution**: delegates to the framework's native preemption mechanism.
(4) **Scoring callback**: provides decode-step signals (*e.g.*, newly generated tokens) to the scoring strategy for priority updates.
(5) **Lifecycle management**: cleans up bids and state when a request completes.

This contract is intentionally minimal: any serving framework exposing KV statistics and a reclamation hook can implement the adapter.

5 Implementation

BidKV is implemented as a standalone Python package with **zero external dependencies** (Python ≥ 3.10 standard library only), totaling ∼12,000 lines of code. The system defaults to **disabled** and includes a global kill switch for instant bypass in production. Strategies are managed through a pluggable registry; adding a new strategy requires implementing a single interface and one registration call.

5.1 vLLM Integration

We integrate with vLLM 0.17.1 (v1 architecture) using its plugin system. BidKV's fundamental integration point is victim selection: the adapter ranks running requests by reclamation utility and delegates execution entirely to vLLM's native preemption mechanism—releasing all KV blocks and requeueing the request for full recomputation. BidKV does not modify vLLM's KV cache coordinator or block pool; it only controls *which* request is reclaimed. Under recompute-fallback semantics, the victim-selection policy does not alter final outputs.

The adapter intercepts scheduling at three points:

(1) **Pre-schedule.** Before each native scheduling call, the hook performs five sub-steps: (a) *Admission reorder*: strategies order the waiting queue by SJF (prompt length) or FCFS (see Table 1). (b) *Victim-priority refresh* (every 3 s): each strategy's `select_victims()` is invoked on the running set to produce a cached priority ordering. BidKV uses the full bid pipeline (Eq. 2–3), ranking candidates by utility $U{=}r/(\delta{+}\varepsilon)$; other strategies use simpler heuristics. (c) *Proactive preemption* (KV>90%, waiting queue non-empty, ≥3 running, 5 s cooldown): the lowest-priority running request from the cached ordering is preempted via vLLM's native mechanism. PE and PE-SJF skip this step (no-intervention baselines). (d) *SRPT preemption* (Static-Random, Largest-First only, KV>80%, 1.5 s cooldown): a running request whose estimated remaining work exceeds 1.2× the total cost of the cheapest waiting request is preempted. BidKV disables SRPT: under recompute-fallback semantics, the re-prefill cost of a long-prompt victim typically outweighs the scheduling gain. (e) *Running-queue reorder*: strategies with this enabled sort the running list by cached priority so that vLLM's native LIFO eviction removes the lowest-priority request. BidKV gates this reorder to KV utilization >95% and mean prompt length ≤500 tokens; the adapter maps high reclamation utility to low keep-priority, so the lowest-priority request removed by the native LIFO back-end corresponds to the intended highest-utility victim from the ranking.
(2) **Post-decode tracking.** After each decode step, newly sampled tokens are appended to per-request state for scoring.
(3) **Lifecycle cleanup.** On request completion, bids and state are cleared.

Strategy differentiation. Table 1 summarizes how the five experimental strategies differ across scheduling dimensions. All share the same reclamation execution path (vLLM native preemption). BidKV's core differentiator is step (b): the bid-based utility ranking that determines reclamation priority. Steps (a), (c), and (e) configure admission ordering, proactive preemption triggers, and queue management that are shared across multiple evaluated strategies; step (d) is used only by heuristic baselines. Because these dimensions co-vary across strategies, Section 6 attributes performance differences to the combined scheduling configuration rather than to isolated mechanisms.

5.2 SGLang Integration

The `SGLangAdapter` integrates with SGLang's RadixAttention engine [25] by hooking the scheduling entry point. To test portability, we deploy BidKV's complete scheduling policy (SJF admission ordering, pressure-gated running-queue reorder, and utility-ranked victim selection) on SGLang without modifying SGLang's internals, and compare against Vanilla SGLang (SGLang's native LRU-based eviction with no managed scheduling) and Random-Evict (BidKV's adapter with random victim selection replacing the utility-ranked ordering).

Table 1: Strategy differentiation across scheduling dimensions. All share the same execution mechanism (vLLM native preemption). "prio" = completion-aware keep score; "gated" = reorder enabled only when KV utilization >95% and mean prompt length ≤500. SRPT = shortest-remaining-processing-time preemption (KV>80%). LIFO* = no managed victim-selection; vLLM's native last-in-first-out eviction determines which request is reclaimed.

Strategy	Wait	Run	Victims	Proactive	SRPT
PE (default)	FCFS	LIFO	LIFO*	✗	✗
PE-SJF	SJF	LIFO	LIFO*	✗	✗
Static-Random	SJF	prio	random	✓	✓
Largest-First	SJF	prio	capacity	✓	✓
BidKV	SJF	gated	$U=\frac{r}{\delta+\varepsilon}$	✓	✗

6 Evaluation

We evaluate BidKV along five dimensions: (1) overall serving performance (Section 6.2), (2) sensitivity to arrival rate (Section 6.3), (3) long-context workload (Section 6.4), (4) reclamation event analysis (Section 6.5), and (5) framework portability on SGLang (Section 6.6).

6.1 Experimental Setup

Hardware and model. All experiments run on a single NVIDIA RTX A6000 GPU (48 GiB) with CUDA 12.5. We serve Llama-3.1-8B-Instruct [7] in bf16 precision (~16 GiB model weight).

Inference engine. vLLM 0.17.1 (v1 architecture) with PagedAttention [11] serves as the primary quantitative platform. We limit GPU memory utilization to 0.5 and fix KV cache capacity at 600 blocks (9,600 tokens) to create sustained KV pressure across all request rates.

Workload and traces. The primary evaluation uses a *mixed-length* workload: 1,000 requests sampled from the ShareGPT Vicuna unfiltered dataset [18], whose length distribution is representative of real-world LLM conversations [24]. Prompt lengths range from tens to thousands of tokens, creating heterogeneous KV footprints that exercise the victim-selection problem formalized in Section 2.2: when the engine reclaims memory, requests differ substantially in both the KV capacity they free and the reclamation cost they impose. Requests arrive according to a Poisson process (open-loop) at three rates: {2.0, 3.8, 5.7} req/s. A supplementary *long-context* workload (500 requests, rates 0.35/0.5/0.7 req/s) tests behavior under extreme KV pressure, where individual requests regularly consume 2,000–4,000 tokens. All traces are generated with seed 42 and verified via SHA-256 checksums; the same traces are shared across all strategies.

SLO definition. We define SLO attainment as the fraction of requests whose TTFT falls below a given threshold. We report SLO at 300 ms for the mixed workload—the strictest threshold that produces measurable strategy differentiation (3–15 pp gaps)—and at 2,000 ms for the long-context workload.

Metrics. Four primary metrics capture serving quality across complementary dimensions: *Throughput* (req/s)—total completed requests per second, the standard capacity metric in LLM serving [11, 22, 25]; *SLO attainment*—the fraction of requests whose TTFT falls within the threshold above; *TTFT P95*—tail first-token latency, the most direct measure of admission responsiveness; and *TPOT P95* [1]—95th-percentile per-token output time, capturing decode-phase latency. Together these four metrics cover prefill (TTFT), decode (TPOT), throughput, and SLO attainment. All tail-latency metrics use P95 rather than P99: at our sample size (1,000 requests × 3 runs), P99 is determined by ~10 extreme values per run, introducing high run-to-run variance compared with the ~50 values that determine P95.

Baselines. Five strategies form a layered comparison that varies waiting-queue ordering, running-queue reordering, proactive reclamation, and victim selection logic (Table 1):

(1) **PE** (vLLM default): FCFS admission, LIFO preemption, no proactive reclamation.
(2) **PE-SJF**: PE plus SJF waiting-queue ordering; isolates the contribution of admission-queue reordering.
(3) **Static-Random**: SJF admission, proactive reclamation (KV > 90%), random victim selection, SRPT preemption (KV > 80%).
(4) **Largest-First**: SJF admission, proactive reclamation, capacity-greedy victim—evicts the request occupying the most KV blocks; SRPT enabled.
(5) **BidKV**: SJF admission, pressure-gated running reorder (KV > 95%), completion-aware cost estimation (δ, Eq. 3) → bids → utility-ranked victim ordering; proactive reclamation enabled (KV > 90%), SRPT explicitly disabled.

The comparison spans a range of reclamation philosophies: PE applies no active scheduling; PE-SJF isolates the contribution of admission ordering; Static-Random and Largest-First add proactive reclamation with simple heuristics; BidKV replaces all per-request heuristics with utility-ranked cross-request victim selection. Each step in this progression changes multiple dimensions simultaneously (Table 1); we attribute performance differences to groups of co-varying features rather than claim strict single-variable isolation. Each configuration is repeated three times; we report the mean across runs.

6.2 Main Comparison

We begin with the medium arrival rate (3.8 req/s) from Table 2, where KV pressure is sustained but the system is not saturated.

Key findings.

- **BidKV achieves the best TTFT P95 at rate = 3.8.** TTFT P95 is 631 ms, 7% below Largest-First (677 ms) and 48% below Static-Random (1,216 ms). SLO(300ms) reaches 83.2%, 0.9 pp below Static-Random (84.1%) but 3.3 pp above Largest-First (79.9%) and 14.8 pp above PE (68.4%).
- **SRPT-enabled strategies lead on throughput and TPOT.** Static-Random achieves the highest throughput (3.57 req/s vs. BidKV 3.42) and lowest TPOT P95 (90.9 ms vs. BidKV 107.2 ms), enabled by aggressive SRPT preemption (KV > 80%). However, its TTFT P95 is ~2× higher (1,216 ms vs. 631 ms), indicating that aggressive throughput reclamation comes at the expense of admission latency.

- **SJF admission is the single largest factor.** PE-SJF—PE plus SJF waiting-queue ordering, with no proactive reclamation or SRPT—improves SLO from 68.4% to 79.6% (+11.2 pp) and cuts TTFT P95 by 90% (6,769→666 ms). However, throughput drops 12% (3.48→3.06 req/s) and TPOT P95 degrades (97.0→153.5 ms), showing that admission reordering alone introduces a latency-throughput tradeoff.
- **BidKV disables SRPT by design.** SRPT aggressively preempts longer-running requests when KV usage exceeds 80%, freeing capacity for waiting requests but increasing recomputation for evicted victims. BidKV explicitly disables SRPT: BidKV's scoring criterion already accounts for estimated reclamation cost via δ, avoiding the cascading preemptions that aggressive reclamation can trigger.

Attribution chain. The layered comparison separates four groups of co-varying features: (1) *PE → PE-SJF* (+11.2 pp SLO, TTFT 6,769→666 ms): SJF admission alone produces the largest latency reduction in the comparison—TTFT drops by an order of magnitude because short-prompt requests no longer queue behind long ones. (2) *PE-SJF → Static-Random* (+4.5 pp SLO, throughput 3.06→3.57 req/s, TTFT 666→1,216 ms): adding proactive reclamation, running-queue reorder, SRPT, and random victim selection recovers throughput (+17%) and further improves SLO, though tail TTFT nearly doubles. (3) *Static-Random → Largest-First* (−4.2 pp SLO, TTFT 1,216→677 ms): replacing random with capacity-greedy victim selection cuts tail latency nearly in half, though SLO decreases because the more conservative selection frees less total capacity per event. (4) *Largest-First → BidKV* (+3.3 pp SLO, TTFT 677→631 ms): replacing capacity heuristics with utility-ranked cross-request victim selection and disabling SRPT, confirming that reclamation-cost-aware victim coordination provides incremental value beyond simple size-based heuristics.

6.3 Rate Sensitivity

Table 2 shows how performance scales across three arrival rates.

Three operating regimes. Low load (rate = 2.0): KV pressure is largely absorbed by rapid request turnover. BidKV leads on all four metrics (SLO 97.0%, TTFT 258 ms, TPOT 72.4 ms), though the absolute margins are small: the throughput spread is ≤0.01 req/s and all strategies achieve ≥92% SLO.

Medium load (rate = 3.8): Sustained KV pressure forces proactive reclamation; this is the primary differentiating regime (Table 2, rate = 3.8). Static-Random leads on throughput (3.57 req/s) and TPOT (90.9 ms), enabled by SRPT. However, its TTFT P95 (1,216 ms) is nearly 2× that of BidKV (631 ms). BidKV achieves the best TTFT P95 and second-best SLO (83.2%), 0.9 pp below Static-Random but 3.3 pp above Largest-First and 14.8 pp above PE. PE-SJF achieves 79.6% SLO without proactive reclamation—above PE (68.4%) but below strategies that actively preempt, confirming that admission ordering alone is insufficient for maximum SLO.

High load (rate = 5.7): All strategies degrade under intense pressure. BidKV leads on SLO (81.0%); PE-SJF achieves the lowest TTFT P95 (727 ms) through conservative scheduling (no proactive preemption), but at lower throughput (3.28 vs. BidKV 3.60 req/s) and lower SLO (74.6%). Static-Random achieves the highest throughput (4.09 req/s, +13.6% over BidKV) but its TTFT P95 is 2.2× higher (1,792 ms).

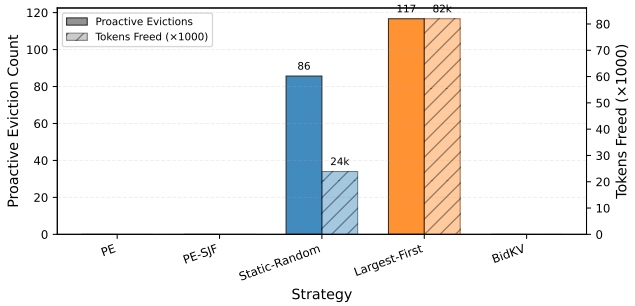

Figure 3: Reclamation event analysis (mixed workload, rate = 3.8). Left bars: proactive eviction count. Right bars: total tokens freed (×1000). BidKV balances eviction frequency and tokens freed per event, avoiding the extremes of too-aggressive or too-passive policies.

Cross-rate summary. Averaging across all three rates, BidKV achieves the best SLO (87.1%) and TTFT P95 (562 ms, 48% below Static-Random's 1,091 ms) at a throughput of 2.99 req/s—6.8% below Static-Random's 3.21 req/s. This tradeoff reflects BidKV's design point: ~7% throughput cost in exchange for substantially better admission responsiveness under sustained KV pressure.

6.4 Long-Context Workload

Table 3 extends the evaluation to a long-context workload (500 requests, rate = 0.70 req/s, SLO threshold 2,000 ms). KV pressure is more severe: typical requests consume 2,000–4,000 tokens, saturating the 9,600-token KV budget with just two to four concurrent requests. PE collapses (SLO 5.0%, TTFT ~95 s), confirming that unmanaged LIFO reclamation is untenable for long-context traffic. Among the four active strategies, Largest-First leads on SLO (64.4%)—when all requests are long, evicting the largest frees the most capacity, a regime where capacity-greedy selection excels. BidKV achieves the best TTFT P95 (30,143 ms, 5 % better than Static-Random, 25 % better than PE-SJF), with competitive SLO (62.3%, 2.1 pp below Largest-First). PE-SJF obtains the best TPOT P95 (145.2 ms) but the worst TTFT (37,551 ms), confirming that admission ordering alone is insufficient when KV pressure is extreme. The TTFT advantage of BidKV reflects the same mechanism as in the mixed workload: utility-ranked selection avoids wasteful recomputation, keeping prefill bandwidth available for queued requests.

6.5 Reclamation Event Analysis

Figure 3 reveals the mechanism behind the rate = 3.8 performance differences:

- **PE** triggers zero proactive evictions, relying entirely on vLLM's reactive LIFO mechanism. This avoids reclamation cost but forfeits the opportunity to free memory for waiting requests, resulting in high TTFT and low SLO.

Table 2: Rate sensitivity (mixed workload, vLLM). Four primary metrics across three arrival rates. Best per rate in bold; second-best underlined. Mean of 3 runs.

Strategy	Rate (req/s)	SLO (300ms, %)	TTFT P95 (ms)	Throughput (req/s)	TPOT P95 (ms)
PE (default)	2.0	92.0	683	1.96	78.1
	3.8	68.4	6769	3.48	97.0
	5.7	56.2	8699	3.49	119.9
PE-SJF	2.0	94.2	330	1.96	84.6
	3.8	79.6	666	3.06	153.5
	5.7	74.6	**727**	3.28	150.6
Static-Random	2.0	96.5	266	1.96	72.8
	3.8	**84.1**	1216	**3.57**	**90.9**
	5.7	80.4	1792	**4.09**	**94.4**
Largest-First	2.0	95.5	291	1.96	74.5
	3.8	79.9	677	3.28	112.9
	5.7	77.7	793	3.53	113.2
BidKV	2.0	**97.0**	**258**	1.96	**72.4**
	3.8	83.2	**631**	3.42	107.2
	5.7	**81.0**	797	3.60	110.0

Table 3: Long-context workload at peak request rate (vLLM, Llama-3.1-8B-Instruct, A6000, 500 req/run, rate = 0.70 req/s, SLO threshold 2,000 ms, mean of 3 independent runs). At lower rates (0.35 and 0.50 req/s) all proactive strategies attain SLO > 70%; relative ordering is qualitatively consistent. Best in bold; second-best underlined.

Strategy	SLO (2000 ms, %)	TTFT P95 (ms)	Throughput (req/s)	TPOT P95 (ms)
PE (default)	5.0	94696	0.413	184.1
PE-SJF	63.6	37551	0.563	**145.2**
Static-Random	57.4	31693	**0.568**	208.3
Largest-First	**64.4**	33345	0.542	239.6
BidKV	62.3	**30143**	0.560	207.0

- **PE-SJF** triggers zero proactive evictions, relying entirely on vLLM's reactive LIFO mechanism—identical to PE in reclamation behavior. Yet SJF admission alone lifts SLO from 68.4% to 79.6%, demonstrating that admission ordering contributes more to scheduling quality than the passive reclamation policy.
- **Largest-First** selects victims by KV occupancy, targeting the request holding the most cache blocks. Without considering cross-request reclamation cost, this occasionally targets near-complete requests whose recompute is expensive, causing cascading pressure.
- **BidKV** maintains a cached utility-ranked ordering of victim candidates using $U = r/(\delta + \varepsilon)$; the top-ranked entry is consumed at each pressure event. The disruption term δ grows with completion progress and prior preemption count, steering selection toward requests whose eviction frees adequate capacity at low reclamation cost. This produces the best TTFT P95 at rate = 3.8 while maintaining competitive SLO.

6.6 Framework Portability

Table 4 reports results on SGLang's RadixAttention engine [25] at the most stressed rate (5.7 req/s, mixed workload). The SGLang adapter applies BidKV's complete scheduling policy—SJF admission ordering, pressure-gated running-queue reorder, and utility-ranked victim selection—mirroring the vLLM configuration without any modification to SGLang's internals.

Table 4: Framework portability: SGLang (mixed workload, rate = 5.7 req/s, Llama-3.1-8B-Instruct, A6000, 1,000 req/run, SLO threshold 300 ms, mean of 3 runs). Vanilla SGLang applies no managed victim selection; Random-Evict and BidKV deploy BidKV's adapter with random and utility-ranked victim selection respectively. Best among active strategies in bold; second-best underlined. †Vanilla SGLang's low TPOT reflects unobstructed decode throughput, not admission efficiency; its TTFT P95 of 6,246 ms indicates severe admission-latency degradation.

Strategy	SLO (300 ms, %)	TTFT P95 (ms)	Throughput (req/s)	TPOT P95 (ms)
Vanilla SGLang	55.2	6246	3.04	58.7†
Random-Evict	85.7	878	**5.57**	88.9
BidKV	**86.9**	**598**	5.53	**85.4**

Managed scheduling vs. native eviction. Vanilla SGLang's native LRU eviction yields 55.2% SLO and TTFT P95 of 6,246 ms—reasonable throughput (3.04 req/s) but severe admission latency. Deploying BidKV's full policy on SGLang raises SLO to 86.9% (+31.7 pp) and cuts TTFT P95 to 598 ms (10.4× reduction), demonstrating that the policy is functional and effective on an architecturally distinct serving engine.

Portability across frameworks. BidKV also outperforms Random-Evict (SLO 86.9% vs. 85.7%, TTFT P95 598 vs. 878 ms), with notably lower run-to-run variance: Random-Evict's TTFT P95 spans 460–1,700 ms across three runs, while BidKV's range is 475–667 ms. Throughput is within 0.7% (5.53 vs. 5.57 req/s). The key portability result is the pattern: on both vLLM and SGLang, BidKV's scheduling policy achieves the best SLO and TTFT P95 at negligible throughput cost, confirming that the complete policy, including its integration

hooks, transfers across serving frameworks without framework-specific tuning.

7 Discussion and Limitations

7.1 Surrogate Disruption Estimate

The disruption estimate δ (Eq. 3) is a *ranking signal* derived from request-lifecycle features, not a precise reclamation-cost oracle. Exact cost prediction is unnecessary: under recompute-fallback semantics the scheduler's task is to *rank* victim candidates so that requests whose reclamation is relatively cheap are preferred over expensive ones. The three features—completion progress, prompt length, and preemption history—capture the axes of reclamation heterogeneity established in Section 2 and need only preserve relative ordering across these dimensions to steer selection toward low-cost victims.

Because reclaimed requests are fully recomputed by the framework's native recovery path, the policy preserves output correctness; the policy governs which request is reclaimed at each pressure event. This choice directly shapes admission responsiveness: cost-aware victim ranking reduces wasted recomputation and keeps KV capacity available for queued requests, reducing TTFT and improving SLO attainment (Section 6).

7.2 Limitations

- **Single GPU.** All experiments use a single A6000. Multi-GPU deployments may introduce coordination overhead for bid collection.
- **Single model scale.** We evaluate on Llama-3.1-8B-Instruct. Larger models (70B+) with different KV-to-weight ratios may exhibit different pressure dynamics.
- **Execution model.** All experiments use recompute-fallback semantics, the default recovery path in vLLM and SGLang. Alternative execution models (e.g., swap-to-host, partial truncation) may change the cost structure; adapting the scorer to such models is future work.

7.3 Broader Impact

Because bids make per-request disruption sensitivity an explicit input to the scheduler, the interface can encode *SLO-differentiated serving*: higher-priority requests express higher disruption cost, receiving stronger protection against reclamation. A consequence is that lower-priority requests may absorb a disproportionate share of preemption events. Per-request preemption caps—limiting how often any single request can be reclaimed within a scheduling window—provide a straightforward guard; exploring richer fairness-aware scheduling extensions is left to future work.

8 Conclusion

We presented BidKV, a bid-based scheduling abstraction for active-KV reclamation in online LLM serving. BidKV surfaces per-request reclamation sensitivity as an explicit, structured signal and performs online utility-ranked cross-request victim selection, replacing implicit ordering heuristics with a utility-guided decision interface ($U = r/(\delta+\varepsilon)$). Under recompute-fallback semantics the policy preserves output correctness; the differentiation lies in *which* request is selected for reclamation at each pressure event—a choice that directly shapes how quickly queued requests receive KV allocation and, consequently, first-token latency and SLO attainment.

Through a five-strategy evaluation on vLLM with Llama-3.1-8B-Instruct under mixed-length ShareGPT workloads, we showed that BidKV achieves the best cross-rate SLO attainment (87.1%) and TTFT P95 (562 ms), outperforming both capacity-greedy heuristics and default scheduling baselines at a modest throughput cost (∼7%). At low load, BidKV leads on all four metrics. At medium and high load, strategies that enable aggressive SRPT preemption trade tail latency for throughput; BidKV provides the best admission responsiveness under sustained KV pressure. A supplementary long-context evaluation confirms that the benefit is most pronounced when heterogeneous request lifecycles create actionable differences in reclamation cost.

BidKV integrates with both vLLM and SGLang via a portable adapter abstraction, reusing each framework's native reclamation and recovery paths without source-code modification. The decoupled scoring-to-selection architecture enables independent evolution of scoring strategies and selection algorithms. Future work includes token-level truncation for partial KV release, fairness-aware scheduling extensions, and multi-GPU coordination.

References

[1] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[2] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. arXiv:2305.13245 [cs.CL] https://arxiv.org/abs/2305.13245

[3] Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, et al. 2025. PyramidKV: Dynamic KV Cache Compression based on Pyramidal Information Funneling. arXiv:2406.02069 [cs.CL] https://arxiv.org/abs/2406.02069

[4] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. 2011. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[5] In Gim, Guojun Chen, Seung seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. 2024. Prompt Cache: Modular Attention Reuse for Low-Latency Inference. arXiv:2311.04934 [cs.CL] https://arxiv.org/abs/2311.04934

[6] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. 2014. Multi-resource packing for cluster schedulers. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (Chicago, Illinois, USA) *(SIGCOMM '14)*. Association for Computing Machinery, New York, NY, USA, 455–466. doi:10.1145/2619239.2626334

[7] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[8] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving DNNs Like Clockwork: Performance Predictability from the Bottom Up. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 443–462.

[9] Jordan Juravsky, Bradley Brown, Ryan Ehrlich, Daniel Y. Fu, Christopher Ré, and Azalia Mirhoseini. 2024. Hydragen: High-Throughput LLM Inference with Shared Prefixes. arXiv:2402.05099 [cs.LG] https://arxiv.org/abs/2402.05099

[10] Hans Kellerer, Ulrich Pferschy, and David Pisinger. 2004. *Knapsack Problems*. Springer. https://doi.org/10.1007/978-3-540-24777-7

[11] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 611–626. doi:10.1145/3600006.3613165

[12] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. SnapKV: LLM Knows What You are Looking for Before Generation. arXiv:2404.14469 [cs.CL]

https://arxiv.org/abs/2404.14469

[13] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023. Scissorhands: Exploiting the Persistence of Importance Hypothesis for LLM KV Cache Compression at Test Time. arXiv:2305.17118 [cs.LG] https://arxiv.org/abs/2305.17118

[14] Silvano Martello and Paolo Toth. 1990. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons.

[15] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2025. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *Proceedings of the 51st Annual International Symposium on Computer Architecture* (Buenos Aires, Argentina) *(ISCA '24)*. IEEE Press, 118–132. doi:10.1109/ISCA59077.2024.00019

[16] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, et al. 2022. Efficiently Scaling Transformer Inference. arXiv:2211.05102 [cs.LG] https://arxiv.org/abs/2211.05102

[17] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, et al. 2023. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. arXiv:2303.06865 [cs.LG] https://arxiv.org/abs/2303.06865

[18] TeamShareGPT. 2023. ShareGPT Conversations Dataset.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[20] Bingyang Wu, Shengyu Liu, Yinmin Zhong, Peng Sun, Xuanzhe Liu, and Xin Jin. 2024. LoongServe: Efficiently Serving Long-Context Large Language Models with Elastic Sequence Parallelism. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles* (Austin, TX, USA) *(SOSP '24)*. Association for Computing Machinery, New York, NY, USA, 640–654. doi:10.1145/3694715.3695948

[21] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient Streaming Language Models with Attention Sinks. arXiv:2309.17453 [cs.CL] https://arxiv.org/abs/2309.17453

[22] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 521–538.

[23] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, et al. 2023. H_2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. arXiv:2306.14048 [cs.LG] https://arxiv.org/abs/2306.14048

[24] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, et al. 2024. LMSYS-Chat-1M: A Large-Scale Real-World LLM Conversation Dataset. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*.

[25] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, et al. 2024. SGLang: Efficient Execution of Structured Language Model Programs. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*.

[26] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[27] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2023. KIVI : Plug-and-play 2bit KV Cache Quantization with Streaming Asymmetric Quantization. (2023). doi:10.13140/RG.2.2.28167.37282