ConServe: Fine-Grained GPU Harvesting for LLM Online and Offline Co-Serving

Yifan Qiao [1] **Shan Yu** [2] **Shu Anzai** [2] **Haoran Ma** [2] **Shuo Yang** [1] **Yang Wang** [3]
Miryung Kim [2] **Yongji Wu** [1] **Yang Zhou** [1 4] **Jiarong Xing** [1 5] **Joseph E. Gonzalez** [1] **Ion Stoica** [1] **Harry Xu** [2]

Abstract

Large language model (LLM) serving demands low latency and high throughput, but high load variability often leads to significant GPU under-utilization. In this paper, we identify a synergistic but overlooked opportunity to co-serve latency-critical online requests alongside *latency-tolerant offline* tasks, which existing systems fail to exploit because their coarse-grained resource management introduces interference. We present ConServe, a co-serving system that enables fine-grained resource sharing through latency-aware token-level scheduling, sub-iteration layer-wise preemption, and incremental KV-cache management. These mechanisms allow offline execution to fill *millisecond-scale* GPU idle time while preserving strict online latency guarantees. Across real-world workloads with Llama-3.1 and Qwen-2.5 models, ConServe improves throughput by 2.2× on average and reduces online tail latency by 2.9× over state-of-the-art systems.

1. Introduction

Large language models (LLMs) power chatbots (OpenAI, 2025; The Claude Team, 2024; Chiang et al., 2024), coding assistants (GitHub, 2025; Rozière et al., 2024), and document processing (Narayan et al., 2018; Google Labs, 2024), driving massive GPU deployments (OpenAI, 2024b; Anthropic, 2024b). Yet inference clusters remain underutilized: bursty online traffic (load can triple within seconds (Wang et al., 2024)) forces providers to provision for peak demand, leaving substantial capacity idle.

[1]University of California, Berkeley, CA, USA [2]University of California, Los Angeles, CA, USA [3]Intel, USA [4]University of California, Davis, CA, USA [5]Rice University, Houston, TX, USA. Correspondence to: Yifan Qiao <yifanqiao@berkeley.edu>.

Proceedings of the 43^{rd} International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.

A natural approach to improving utilization is to backfill idle resources with *best-effort* workloads. However, GPU resource harvesting is challenging because online inference workloads must promptly reclaim resources to meet millisecond-level SLOs. Prior approaches primarily co-locate inference with training or fine-tuning jobs (Xiao et al., 2020; Miao et al., 2024; Oliaro et al., 2026), typically as independent processes running in separate containers or virtual machines (Fu et al., 2024; Zhang et al., 2024), incurring task-switching overheads that take seconds to minutes.

To make GPU harvesting practical, it is critical to find best-effort jobs suitable for co-locating. In this work, we argue that a more synergistic yet overlooked opportunity lies in co-serving online, latency-sensitive requests with *offline*, latency-tolerant batch inference. This option is increasingly viable as providers like OpenAI (OpenAI, 2024a) and Anthropic (Anthropic, 2024a) now offer batch APIs with a 24-hour turnaround at a lower price for tasks such as data analytics (Liu et al., 2024) and model testing/evaluation (Liang et al., 2023). Offline batch inference offers two key advantages for GPU harvesting: (1) offline tasks share the same model and can be co-scheduled with online ones in the same inference engine, enabling seamless reuse of the loaded model without costly process switching; and (2) offline requests can be freely preempted to quickly accommodate sudden online traffic spikes.

However, simply co-locating online and offline requests is insufficient, as existing serving systems manage requests and their states at a coarse granularity, leading to severe interference between co-located workloads and causing violations of the tight SLOs of online workloads. Specifically, our profiling reveals three critical problems spanning the entire serving lifecycle: scheduling, execution, and KV cache state management.

- *Static scheduling leads to unpredictable latency*. Chunked prefill with fixed compute budgets works for online-only serving but fails under co-serving where GPUs are kept near saturation, making latency highly sensitive to context lengths and KV cache states. As a result, static, chunk-level scheduling cannot control latency accurately, which causes frequent SLO violations.

- *Iteration-level preemption causes slow responsiveness*. Current engines execute batches as uninterruptible iterations for efficiency, allowing preemption only after completion. However, our profiling shows that this coarse-grained approach can delay online requests by up to 970 ms.
- *Request-level state management incurs prohibitive overhead*. KV caches can grow to gigabytes per request. Preempting a request either discards its cache (expensive recomputation) or swaps to host memory (long stalls), both degrading throughput.

This paper addresses a central question: *Can idle GPU cycles and memory be harvested during online LLM serving while achieving both high throughput and strict SLO guarantees?* Our key insight is that resolving the tension between low latency and high efficiency requires managing resources and orchestrating execution at a granularity *finer than requests or iterations*, which enables millisecond-scale scheduling and preemption to preserve SLOs while improving throughput.

We introduce ConServe, an LLM co-serving system that harvests idle GPU resources while adhering to strong online SLOs. ConServe operates at sub-request and sub-iteration granularity: (1) SLO-aware scheduling controls offline token admission, (2) layer-wise preemption enables fast reaction to traffic spikes, and (3) incremental KV cache management avoids expensive recomputation or swapping. Together, these mechanisms allow ConServe to saturate GPU resources with offline work while retaining millisecond-scale responsiveness for online requests.

Across four real-world datasets and diverse synthetic workloads, ConServe provides strong performance isolation for online inference, reducing P99 latency by an average of $2.9\times$ while improving offline throughput by $2.2\times$ compared to state-of-the-art preemptive serving systems.

In summary, this paper makes the following contributions:

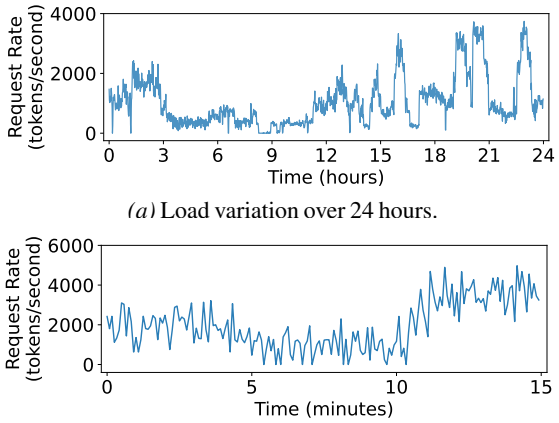
- We identify fundamental limitations of existing co-serving systems and design an SLO-aware, token-level scheduling framework with a precise cost model that captures both compute and memory pressure for latency prediction.
- We introduce layer-wise preemption and incremental KV cache management, enabling millisecond-scale preemption with near-zero overhead.
- We implement ConServe with $2.9\times$ lower P99 latency and $2.2\times$ higher offline throughput on real-world workloads.

Conflict of Interest Disclosure. The authors declare no financial conflicts of interest in connection with this work.

2. Background

2.1. Online and Offline LLM Serving

Broadly, LLM serving can be categorized into two types: online serving, which generates real-time responses to user inputs, and offline serving, which processes inputs in batches and responds asynchronously.



(a) Load variation over 24 hours.

(b) Load variation over 15 minutes.

Figure 1: User traffic to ChatGPT within a campus exposes high load variability at various time scales.

Online serving is latency-critical and is the default mode offered by existing API providers (OpenAI, 2024b; Anthropic, 2024b). Because LLMs generate variable-length token sequences, serving latency is measured at the token level, using *time to first token (TTFT)* for the prefill phase and *time between tokens (TBT)* for per-token latency during decoding. To ensure responsive user experiences, LLM serving systems must tightly control both TTFT and TBT under strict SLOs. For instance, interactive chatbots may require P99 TTFT of 1.5s and P99 TBT of 100ms (Proxet., 2023; Mosaic AI Research., 2024).

In contrast to online serving, which prioritizes low latency, offline serving targets latency-tolerant, best-effort workloads. Requests are submitted in batches and processed to maximize hardware utilization; as a result, *token throughput* becomes the primary performance metric. Offline serving is widely used for large-scale tasks such as document summarization (Jin et al., 2024), data wrangling (Narayan et al., 2022), data analytics (Liu et al., 2024), and LLM benchmarking and evaluation (Liang et al., 2023).

3. Motivation

3.1. GPU Underutilization in LLM Serving

Online Load Burstiness. LLM service providers and recent studies (Moonshot AI, 2024; Patel et al., 2025; Wang et al., 2024) have reported that real-world LLM workloads often expose diurnal patterns and high load variability. As shown in Figure 1(a), ChatGPT traffic varies significantly between peak afternoon hours and off-peak morning periods. Figure 1(b) further shows that load fluctuates sharply even at the minute timescale, with request rates increasing by up to $3\times$ within 10 minutes.

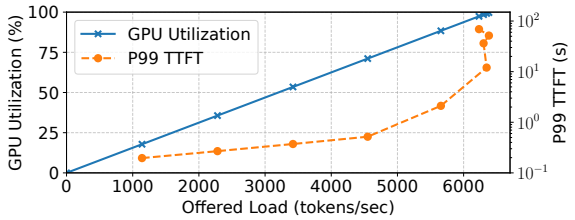


Figure 2: GPU utilization and the P99 TTFT across request rates measured with Llama-3.1 8B on an H100 GPU. Utilization is defined as the offered load relative to the GPU's maximum achievable load.

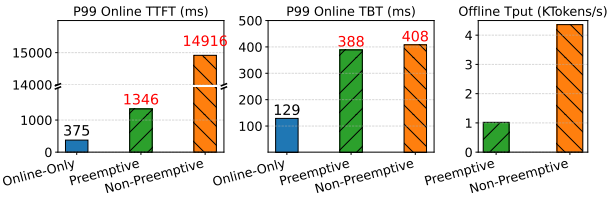


Figure 3: P99 online latencies and offline throughput when co-serving online and offline requests with preemptive or non-preemptive scheduling with Llama-3.1 8B on one H100 GPU.

Static Reservation Underutilizes GPUs. To meet stringent latency SLOs under such highly variable demand, LLM service providers typically reserve GPU capacity for peak load using long-term or on-premise GPU instances (Xia et al., 2025). Because peak demand can be several times higher than average load, this static reservation leads to substantial underutilization during off-peak periods. As shown in Figure 2, while the model achieves high utilization at peak load, under a moderate load of 2000 tokens/s (the average load in Figure 1b), more than 70% of GPU resources are left idle.

3.2. Challenges in LLM Co-Serving

While co-serving online and offline workloads on the same LLM instance can significantly improve GPU utilization, it poses a fundamental challenge: simultaneously satisfying the strict low-latency requirements of online inference and the high-throughput demands of offline workloads. To quantify this trade-off, we extend vLLM (vLLM team, 2025) with priority-based scheduling and preemption (details in §6.1), denoted as `Preemptive`. Building on chunked-prefill (Agrawal et al., 2024), the scheduler fills unused capacity with low-priority offline tokens and preempts offline requests when online requests arrive. We also introduce a non-preemptive variant, `Non-Preemptive`, which schedules offline requests at low priority but allows them to run to completion, providing an upper bound on achievable offline throughput.

We compared these two policies against `Online-Only`, which serves only online inference and represents optimal online latency. We used a synthetic online workload (Gamma arrivals, CV=0.5, 2 req/s) alongside sufficient offline requests. As shown in Figure 3, simple preemptive scheduling

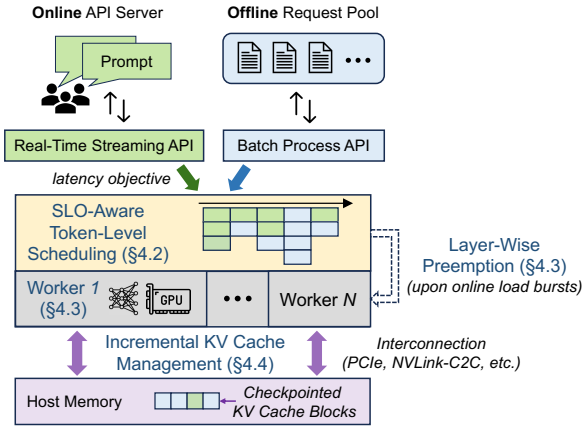


Figure 4: Design overview of ConServe.

achieves neither low online latency nor high offline throughput: compared to `Online-Only`, `Preemptive` increases P99 online TTFT and TBT by 3.59× and 3.01×, respectively, and reduces offline throughput by 4.27× relative to `Non-Preemptive`. Our analysis revealed three root causes:

- **Coarse-grained scheduling leads to contention**: The scheduler operates at chunk granularity and greedily fills each chunk with offline tokens without accounting for the resulting compute and memory contention, which slows concurrent online decoding and inflates TBT.
- **Iteration-level preemption causes queuing delays**: Preemption is only possible after a full iteration finishes, forcing incoming online requests to wait for the entire in-flight chunk to complete. Such iteration-level preemption creates head-of-line blocking that can last hundreds of milliseconds, severely degrading TTFT.
- **Request-level KV cache management incurs high overhead**: KV cache state is managed at the entire sequence's granularity, forcing a costly dilemma upon preemption: either recompute the entire KV cache later or stall the GPU to swap it to host memory. Our profiling shows that under memory pressure, up to 69% of GPU time is wasted on recomputation or swapping (Kwon et al., 2023).

Takeaway. These challenges necessitate a system featuring fine-grained resource management: (1) Adaptive scheduling that bounds the latency impact of co-served offline work within online SLOs; (2) sub-iteration preemption to mitigate head-of-line blocking; and (3) sub-request KV cache management for efficient context switching.

4. Design

4.1. ConServe Overview

As shown in Figure 4, ConServe is an LLM co-serving system that harvests idle GPU compute and memory by jointly serving online and offline requests within a single

engine, while preserving strict online SLOs. ConServe achieves this by operating at a much finer granularity than existing systems across the entire serving stack. At the scheduling layer, an SLO-aware scheduler admits offline work at the token level using a context-aware performance model to dynamically determine a safe token budget each iteration. At the execution layer, ConServe enables fast, layer-wise preemption, allowing offline requests to be interrupted within a forward pass to promptly accommodate online traffic bursts. At the state management layer, ConServe employs incremental, token-level KV cache checkpointing to preserve offline progress at near-zero cost. Together, these mechanisms allow ConServe to safely saturate GPU resources with offline work while maintaining millisecond-scale responsiveness for online inference.

4.2. SLO-Aware Token-Level Scheduling

Limitations of static chunk-level scheduling. Modern serving engines (Zheng et al., 2024; Kwon et al., 2023) adopt chunked prefill (Agrawal et al., 2024) by default for online serving. Chunked prefill splits a long prefill into fixed-size chunks to reduce head-of-line blocking and stabilize TBT, assuming that a chunk of a given token size incurs roughly constant latency. While this assumption holds in online-only serving with lightly loaded GPUs, it breaks down under co-serving at high GPU utilization, where chunk latency becomes highly sensitive to request context length and accumulated KV cache state. For example, prefilling a 2048-token chunk for Llama-3.1 8B on an H100 takes 51ms for a new request but increases to 124ms when the request already has a 40K-token context. Given a 100ms TBT SLO, a static chunking policy treats both cases identically, leading to an SLO violation in the latter case. This demonstrates that fixed, static chunk-level scheduling yields inaccurate latency estimates under varying request contexts and cannot simultaneously provide high efficiency and strict latency guarantees under co-serving.

Context-aware performance model. To accurately predict the latency of any potential batch with varying contexts, we build a context-aware performance model for LLMs. We model iteration latency as a function of two key variables: the number of compute tokens (P) being actively processed, and the number of pre-existing context tokens (C) held in the KV cache.

$$\textbf{Latency}(P,C) = \underbrace{k_1 P}_{\text{Linear Compute}} + \underbrace{k_2 P(P+C)}_{\text{Attention}} + \underbrace{k_3 P}_{\text{Communication}} + \underbrace{k_4(P+C)}_{\text{Memory Access}} + \underbrace{k_5}_{\text{Constant}}$$

Each term captures a distinct component of transformer execution. The linear compute term ($k_1 P$) accounts for per-token operations such as QKV projections and FFN layers. The attention term ($k_2 P(P+C)$) explicitly models the quadratic cost of attention, reflecting interactions between newly generated tokens and all existing context tokens, and dominates latency for long-context requests. The communication term ($k_3 P$) captures collective communication overheads (e.g., `all_reduce`) in tensor-parallel settings. The memory-access term ($k_4(P+C)$) models HBM traffic for loading the KV cache, while the constant term (k_5) captures fixed per-iteration overheads.

ConServe learns the coefficients (k_1–k_5) using a *one-time, offline* profiler. For a given model and hardware configuration, the profiler measures iteration latency across a grid of (P,C) values and fits the model via polynomial regression. Profiling completes within 20 minutes even for 70B models and achieves under 4% relative prediction error.

At runtime, the profiler exposes a `can_schedule(batch, req, TBT)` interface that determines whether and how many tokens from `req` can be admitted into the current batch while satisfying the TBT SLO; returning zero indicates that admission would violate the latency budget.

Latency-aware token-level scheduling. Guided by the latency prediction model, ConServe performs dynamic, token-level scheduling at each iteration to compose the optimal batch that respects online SLOs while maximizing utilization. As shown in Algorithm 1, the scheduler always prioritizes online requests to avoid queuing delay, then incrementally admits offline tokens by querying the profiler to ensure the resulting batch satisfies the online TBT SLO.

Beyond latency control, the scheduler must also manage the limited GPU KV cache capacity. In co-serving, paused offline requests continue to occupy memory and can block incoming online requests; to address this, ConServe releases their KV caches on demand and restores them efficiently when offline execution resumes (§4.4). When no online traffic is present, the scheduler switches to an offline-optimized mode, using the performance model to select batch sizes that maximize throughput until online requests arrive.

4.3. Layer-Wise Preemption

As discussed in §3, iteration-level preemption in co-serving forces new online requests to wait for long-running offline iterations, causing TTFT violations under traffic spikes. Avoiding this head-of-line blocking requires sub-iteration preemption. However, it is challenging because inference engines execute each iteration uninterruptibly as a single executable or a CUDA Graph. Intra-iteration interruption can leave inconsistent states and potentially crash the engine.

ConServe breaks this iteration-level barrier by instrumenting the model at the Transformer layer granularity, which brings two key advantages. First, modern LLMs typically comprise dozens of layers where each layer runs for a few milliseconds, ensuring high responsiveness. Second, layers

Algorithm 1 SLO-aware token-level scheduling.

Input: Online queue Q_{on}, offline queue Q_{off}, TBT target t_{TBT}.
▷ *Invoked every iteration to compose the next batch.*
$B \leftarrow \emptyset$
// Phase 1: Schedule online requests with preemption if needed.
for request $r \in Q_{\text{on}}$ **do**
  if GPU memory insufficient for r **then**
    Preempt paused offline requests to free KV cache
  end if
  $n \leftarrow \text{Profiler.can\_schedule}(B, r, t_{\text{TBT}})$
  break when $n = 0$ *// SLO budget exhausted*
  Add n tokens from r to B
end for
// Phase 2: Fill remaining capacity with offline tokens.
for request $r \in Q_{\text{off}}$ **do**
  if GPU memory insufficient for r **then**
    break *// stop offline scheduling*
  end if
  $n \leftarrow \text{Profiler.can\_schedule}(B, r, t_{\text{TBT}})$
  break when $n = 0$ *// SLO budget exhausted*
  Add n tokens from r to B
end for
return Batch B

represent natural computation boundaries with clear input and output dependencies, which allows offline requests to pause and exit early with simple cleanup.

ConServe codesigns the scheduler and workers to realize layer-wise preemption. The scheduler is responsible for detecting potential latency violations. It spawns a dedicated monitor thread, which is triggered upon receiving a new online request. It uses the performance model and profiler (§4.2) to estimate if waiting for the current batch to finish would cause the new request to miss its TTFT SLO. If a violation is predicted, the scheduler immediately signals for preemption by writing to a designated flag in shared host memory.

ConServe workers instrument the model with lightweight synchronization points (safepoints) between layers, inspired by language runtime designs (OpenJDK, 2024). At each safepoint, the worker checks the preemption flag with a single device-to-host memory load. If the flag is set, the worker will terminate computation for all offline requests in the batch, clean up their hidden states and attention metadata, and then resume execution for the remainder of the iteration with only the online requests.

Tensor parallelism. One additional challenge in implementing the safepoint with tensor parallelism (TP) is that collective communication (*e.g.*, all-reduce) in TP requires strict synchronization across workers. Arbitrary preemption of a subset of workers can cause communication deadlocks (Pan et al., 2023). To address this, ConServe will synchronize all workers with `ncclBroadcast` at the beginning of the safepoint before checking for preemption signals.

CUDA graph compatibility. Our safepoint mechanism is fully compatible with CUDA Graphs. First, modern

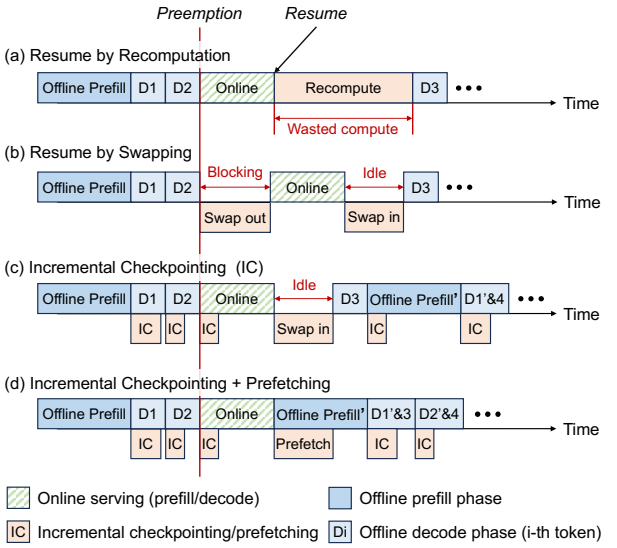


Figure 5: Different preemption and resumption strategies. (a) Resuming by recomputation achieves low preemption delay at the cost of additional computation. (b) Resuming by swapping avoids recomputation but blocks the schedule of incoming online requests. (c) Incremental checkpointing (IC) minimizes both preemption delay and resumption cost. (d) Incremental checkpointing + prefetching overlaps swap-in with computation of the next batch.

engines such as vLLM capture each model layer as a separate graph, where we can insert the safepoint between graph launches. For systems that capture the model as a single graph, safepoint can be implemented as a conditional CUDA graph node (NVIDIA, 2025) within the graph.

Overhead analysis. We have carefully implemented safepoint to minimize its cost. Each safepoint check consists of a single read from pinned host memory (7µs) and a single broadcast/barrier operation across GPUs (13µs on NVLink). This raw cost of ∼20µs is negligible compared to the several milliseconds required to execute a transformer layer. More details are presented in §6.5.

4.4. Incremental KV Cache Management

Effective preemption requires not only interrupting computation but also rapidly reclaiming GPU memory. Modern LLM serving engines, such as SGLang (Zheng et al., 2024) and vLLM (Kwon et al., 2023), support preemption by evicting the KV cache of a request as a whole. This coarse-grained design forces a fundamental tradeoff between reclamation latency and restoration cost.

Figure 5 illustrates two strawman eviction strategies. (a) Recomputation immediately discards the KV cache, enabling fast memory reclamation but wasting GPU cycles by requiring the request to recompute its entire prefix upon resumption. (b) Swapping preserves the KV cache by spilling it to host memory, but incurs blocking I/O overhead propor-

tional to the request's token length, delaying online request admission and potentially causing TTFT SLO violations.

This inefficiency stems from request-level KV cache management, which tightly couples preemption cost to a request's sequence length. To enable fine-grained preemption, ConServe manages the KV cache at token granularity with incremental checkpointing. Leveraging the insight that the cache is append-only, ConServe asynchronously saves only the state for the single, newly-generated token after each iteration. This decouples the cost of preemption from a request's history, making the overhead constant and negligible, and eliminating the swap-vs-recompute dilemma of monolithic eviction.

Unlike traditional swapping, incremental checkpointing keeps KV caches in GPU memory during normal execution and discards them only on demand at token granularity when preemption occurs. Symmetrically, when GPU memory becomes available, ConServe incrementally prefetches KV caches for evicted tokens in the background. As shown in Figure 5(d), prefetching fully overlaps with computation and remains off the critical path.

Overhead analysis. The overhead of incremental KV checkpointing is negligible. On an H100 with a 64 GB/s device-to-host interconnect, checkpointing Qwen-2.5-14B (192 KB KV cache per token) supports over 349K tokens/s, far exceeding typical generation throughput. Even checkpointing a full 2048-token chunk takes only 10 ms and can be fully overlapped with computation (detailed evaluation in §6.5).

5. Implementation

We have implemented ConServe atop vLLM (Kwon et al., 2023). To unify the scheduling logic, ConServe internally assigns priorities to online and offline requests upon their arrival via the real-time streaming API or the batch API. The ConServe scheduler manages all requests using a priority queue and coordinates the scheduling of online and offline requests with a unified control loop.

To support incremental KV cache checkpointing, ConServe manages KV cache at a granularity of 16 tokens to align with vLLM page size, and enhances vLLM's KV cache manager to track the mapping between each GPU KV page and its corresponding CPU KV page for checkpointing. This mapping is recorded in an extended field of the virtual page table.

To support efficient KV cache transfer for tokens scattered in GPU memory, we implement an efficient kernel that gathers their KV cache into a contiguous buffer and transfers it once.

To ensure overlap between computation and asynchronous swap I/O, ConServe creates a separate CUDA stream for KV cache checkpointing operations. Original model executable or CUDA Graphs are still launched in their own streams. Since the checkpointed or prefetched KV cache blocks have no data dependencies with the ongoing model computation, no synchronization is required between CUDA streams.

6. Evaluation

Our evaluation aims to answer the following questions:

1. Can ConServe efficiently harvest GPU resources while maintaining low online latency? (§6.2)
2. Can ConServe achieve both low online latency and high offline throughput under real-world workloads? (§6.3)
3. Can ConServe maintain efficiency for workloads with different degrees of burstiness and request lengths? (§6.4)

6.1. Setup

Environment and models. We conducted experiments on a DGX server with two 48-core CPUs, 2TB of memory, and eight NVIDIA H100 GPUs connected with 900GB/s fully-meshed NVLink.

We evaluated ConServe on three models of varying sizes, including Llama-3.1 8B (Grattafiori et al., 2024), Qwen-2.5 14B (Yang et al., 2024), and Llama-3.1 70B (Grattafiori et al., 2024). All models were served in FP16 using the minimum required GPUs: 1 H100 for Llama-3.1 8B and Qwen-2.5 14B, and 4 H100s for Llama-3.1 70B with tensor parallelism. Among them, Llama-3.1 8B represents a memory-rich setting, with small model weights (16 GB) leaving ample GPU memory for KV caches (60 GB), while Qwen-2.5 14B reflects a memory-constrained setting, where larger weights (28 GB) reduce KV cache capacity (48 GB). Llama-3.1 70B demonstrates a multi-GPU tensor-parallel setting.

Baselines. We compared ConServe with four baselines:

- *Online-Only*: Original vLLM serving only online inference (optimal online latency, zero offline throughput).
- *Non-Preemptive*: Schedules offline requests at low priority without preemption (upper bound on throughput).
- *Sarathi-Preemptive (Sarathi-P)*: Chunked prefill (Agrawal et al., 2024) extended with priority scheduling and preemption under memory pressure.
- *DistServe-Preemptive (DistServe-P)*: Prefill-decode disaggregated serving (Zhong et al., 2024) with similar preemption policies; we report the best prefill/decode GPU ratio.

We implemented or ported all baselines to the same engine with chunked-prefill enabled by default for fair comparisons.

Real-world workloads. For the online workload, we evaluated BurstGPT (Wang et al., 2024), a representative trace of GPT-4 (OpenAI, 2023) user requests collected from a university campus. For offline workloads, we evaluated three tasks: DuReader (He et al., 2018) for

Table 1: Real-world online and offline workloads for evaluation.

Task	Dataset	Input Tokens		Output Tokens	
		Mean	P99	Mean	P99
Online chatting	BurstGPT (Wang et al., 2024)	2747	3446	267	481
Benchmarking	DuReader (He et al., 2018)	21776	29306	166	754
Question answer	MultiNews (Fabbri et al., 2019)	6916	12084	394	788
Summarization	VCSUM (Wu et al., 2023)	18803	32768	337	510

model benchmarking, MultiNews (Fabbri et al., 2019) for multi-document summarization, and VCSUM (Wu et al., 2023) for meeting summarization. Table 1 summarizes all workloads and their input/output length distributions.

Synthetic workloads. We generate synthetic request arrivals using a Gamma process, following methodologies established in prior trace studies (Wang et al., 2024; Sheng et al., 2024; Li et al., 2023). We vary key parameters, including request rate, SLO tightness, request lengths, and traffic burstiness for overall performance testing (§6.2) and sensitivity analysis (§6.4).

Metrics. We use different metrics for online and offline serving. For online requests, we report P99 TTFT and TBT, following prior work (Agrawal et al., 2024). For offline requests, we measure throughput as total tokens processed per second, including both prefill and decode tokens.

6.2. GPU Efficiency and Latency

This subsection evaluates whether ConServe can harvest idle GPU resources to improve offline throughput while preserving online SLOs.

Setup. To enable a comprehensive evaluation, we generate synthetic online loads with varying request rates and SLO scales. Following BurstGPT (Wang et al., 2024), we set the burstiness (coefficient of variation, CV) to 0.5, with each request containing 4096 input tokens and 256 output tokens. A sensitivity analysis on request lengths and burstiness is provided in §6.4. We configured ConServe's target SLOs based on the P99 latencies achieved by `Online-Only`.

Online load vs. performance trade-offs. Figure 6(a) shows P99 latencies and offline throughput as request rate increases. ConServe is the only system achieving both low online latency and high offline throughput.

ConServe maintains P99 TTFT and TBT within 25% and 19% of the optimal baseline across all rates. Other baselines suffer from coarse-grained designs: `Non-Preemptive` has high latency from head-of-line blocking; preemptive baselines cannot bound interference from offline tokens; `Sarathi-P` sometimes yields worse TBT due to costly recomputation. `DistServe-P` shows poor latencies due to load imbalance between prefill/decode instances, causing frequent swapping and back-pressure.

For offline throughput, ConServe consistently outperforms all other preemptive baselines and achieves an average of 82.3% of the optimal throughput. At low online load, both `Sarathi-P` and ConServe achieve over 70% throughput due to sufficient GPU memory. As load increases, ConServe 's advantage widens: by eliminating recomputation overhead, it achieves 3.0× higher offline throughput on average (up to 7.48×) compared to `Sarathi-P`, while `DistServe-P` yields the lowest throughput due to frequent decode-side preemptions.

Online SLO scale vs. performance trade-offs. Figure 6(b) illustrates how system performance changes as we relax the online SLO scale s, where $\text{SLO}_{\text{TTFT/TBT}}(s) = s \cdot P99_{\text{TTFT/TBT}}$, dictating the allowed latency increase over `Online-Only`.

Uniquely, ConServe can trade off offline throughput for stricter online SLO adherence, enabling >99% TTFT and TBT attainment in all settings when $s > 1$. Even in the most challenging setting with no slack ($s = 1.0$), where minor system jitter can cause violations, ConServe still maintains at least 86% attainment, significantly exceeding all baselines. This SLO awareness also enables ConServe to harness latency slack for higher offline throughput until $s = 1.5$, where ConServe achieves an average 93% optimal throughput. Further increasing s brings a diminishing 2% throughput improvement.

In contrast, all other baselines are SLO-oblivious and show constant offline throughput. `DistServe-P` presents an interesting case. It can achieve high TBT attainment thanks to the disaggregation. But the attainment is still lower than ConServe in most cases because preemptions still cause TBT jitter. Its TTFT attainment remains poor due to back-pressure.

In summary, the results demonstrate that ConServe can effectively co-serve online and offline inference and adapt to different SLOs. It maintains near-optimal P99 online latency while achieving high GPU utilization for offline throughput, closely matching the optimal baseline. Compared to other baselines, on average, ConServe reduces P99 online TTFT by 2.37× and TBT by 2.72×, respectively, while simultaneously increasing offline throughput by 3.00×.

6.3. End-to-End Serving with Real-World Workloads

This subsection evaluates ConServe with real-world loads and compares its end-to-end performance against the baselines. We sampled a 10-minute trace from BurstGPT (Wang et al., 2024) as the online workload. For the offline workload, we use both a random mixture of three distinct offline datasets and evaluate co-serving with each dataset individually.

Figure 7 shows time-series results for Llama-3.1 70B, with red lines indicating SLOs. Only `Online-Only` and ConServe consistently meet SLOs; other baselines exhibit unstable latency with frequent violations.

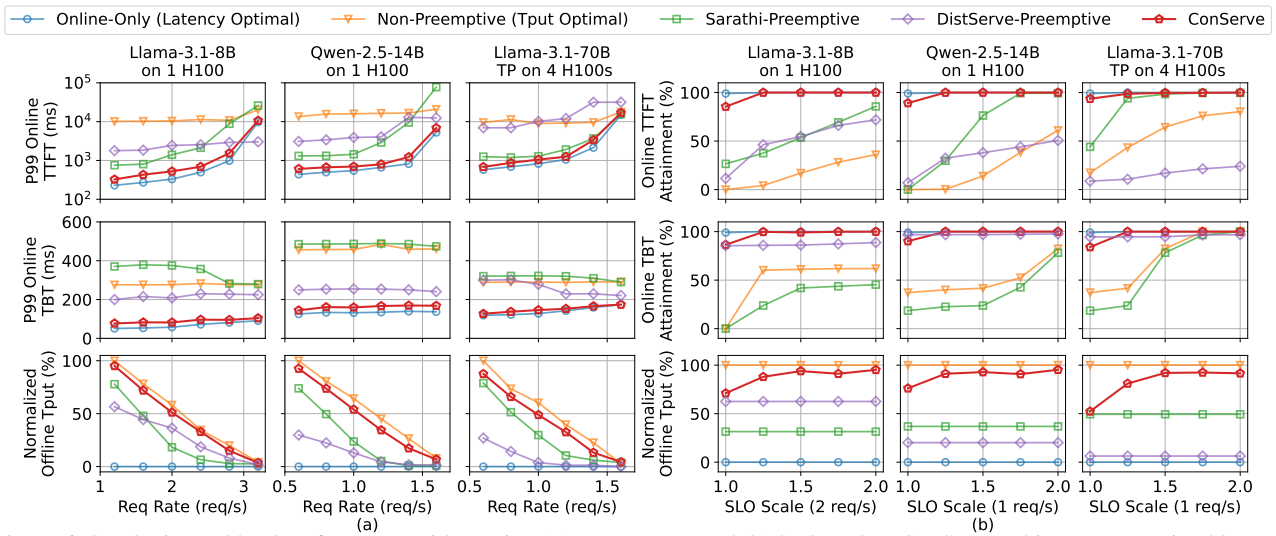


Figure 6: Synthetic workload performance with varying (a) request rates and (b) SLO scales. ConServe achieves near-optimal latency and throughput.

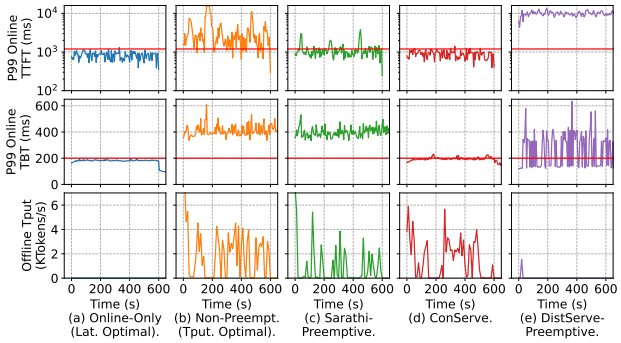


Figure 7: Llama-3.1 70B on real workloads. ConServe meets SLOs (red lines) with 78% of optimal throughput.

ConServe achieves 1553 tokens/s offline throughput (78% of optimal). `Sarathi-P` triggers costly recomputations on long sequences, yielding only 895 tokens/s. `DistServe-P` performs worst due to back-pressure (details in Appendix A.3).

In summary, under realistic and dynamic workload conditions, ConServe can maintain strict online SLOs while delivering near-optimal offline throughput. Detailed summarized results across all offline datasets are provided in Table 2 in Appendix A.2.

6.4. Performance Sensitivity Analysis

In this section, we evaluate ConServe's robustness to varying workload characteristics, specifically load burstiness and request lengths. We conduct these experiments based on the setup used for Figure 6(a), varying one parameter at a time (burstiness, input length, or output length) while keeping other conditions constant.

Load burstiness vs. performance. We use CV to represent load burstiness, where higher CV indicates greater load variability. As Figure 8(a) shows, ConServe demonstrates strong resilience to bursty loads with low online latency and high offline throughput that are close to ideal values. Compared to the best-performing baseline, ConServe reduces P99 TTFT and TBT by an average of 1.68× and 2.38×, respectively, and improves offline throughput by 3.62×.

Input/output lengths vs. performance. To maintain comparable load when varying sequence lengths, we adjust the request rate inversely proportional to length relative to the §6.2 setup (4096 input tokens, 256 output tokens, at 2 req/s).

Figure 8(b) and (c) show results for varying input and output lengths. ConServe delivers good online latency that is close to `Online-Only` across most settings. Regarding offline throughput, ConServe consistently outperforms two preemptive baselines. This advantage is most pronounced for typical requests with input lengths between 2K and 16K tokens, and output lengths longer than 128 tokens. We analyze the performance at the extremes. (1) Requests with short inputs (<2K tokens) cannot fully utilize KV cache capacity. This allows preemptive baselines to keep paused offline requests in GPU memory and avoid recomputation; (2) Requests with long inputs (>16K tokens) or short outputs (<128 tokens) are compute-intensive, leaving little idle capacity for offline serving. (3) Requests with long outputs (>512 tokens) are less compute-intensive but consume more KV cache because they stay on GPUs longer. This memory pressure limits the number of offline requests that can be served and slightly hinders offline throughput.

In summary, ConServe can achieve robust performance across varying degrees of burstiness and request lengths, and it delivers consistently higher offline throughput than baselines, especially for typical-length requests that closely align with real-world workloads.

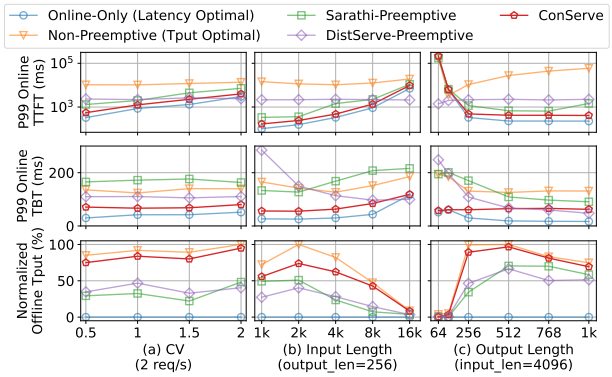


Figure 8: Llama-3.1 8B serving performance under workloads with varying CVs and input and output lengths.

6.5. Design Drill-Down

Ablation study. We quantified the individual contribution of each of our three core optimizations by incrementally enabling them atop the `Sarathi-P` baseline, using the Llama-3.1 8B setup from Figure 6(a). Compared to `Sarathi-P`, our SLO-aware scheduler reduces P99 online TBT from 375ms to 73ms. This shorter iteration time also makes the scheduler more responsive to new requests and reduces P99 online TTFT from 1399ms to 686ms. Enabling layer-wise preemption eliminates TTFT jitter and further reduces P99 TTFT from 686ms to 523ms and worst-case TTFT by 34% from 866ms to 573ms. Finally, adding incremental KV cache management further improves offline throughput by 1.29× from 8810 tokens/s to 11366 tokens/s.

Cost of incremental KV cache management. We measured the overhead and PCIe utilization of our incremental KV cache mechanism. For Llama-3.1 8B on a single H100 GPU, our kernel can transfer KV caches at 37GB/s, and the largest transfer finishes within 7ms and can be completely overlapped. The kernel introduces a $<$500µs cost to gather KV caches of scattered tokens, which is negligible compared to model execution. It also scales well in a distributed setting. For Llama-3.1 70B on 4 H100s, each TP worker checkpoints its local KV cache shard concurrently, achieving an aggregated bandwidth of 132GB/s.

Efficiency of layer-wise preemption. We measured the runtime cost of the preemption safepoint and compared it to the model execution time. On a single GPU, the instrumented safepoint incurs negligible cost ($<$10µs). Even on four GPUs with TP, each safepoint takes only 167.2µs on average. This is slightly higher than raw instruction cost due to the minor lag among TP workers, but still three orders of magnitude faster than model execution, which takes 262ms on average.

ConServe instruments the model every 4 layers and introduces a total latency increase of 3.0ms when no preemption occurs, accounting for a 1.1% overhead compared to the model execution time. Meanwhile, ConServe maintains responsiveness by detecting new requests and preempting the running batch within 13ms, ensuring timely responses to online load bursts.

7. Related Work

LLM Serving Systems. Prior systems optimize either throughput (continuous batching (Yu et al., 2022), PagedAttention (Kwon et al., 2023)) or latency (chunked-prefill (Agrawal et al., 2024; Holmes et al., 2024)) for single workload types. DistServe (Zhong et al., 2024) and Splitwise (Patel et al., 2025) disaggregate prefill/decode but suffer from load imbalance. Concurrent with our work, HyGen (Sun et al., 2026) also explores online-offline co-serving with SLO-aware scheduling. While HyGen focuses on latency prediction and prefix sharing at iteration granularity, ConServe uniquely enables sub-iteration preemption through layer-wise interruption and addresses KV cache overhead through incremental checkpointing.

Generic Model Serving Systems. Prior ML serving systems (NVIDIA, 2024; The PyTorch Foundation, 2024; Crankshaw et al., 2017; Gujarati et al., 2020; Han et al., 2022; Li et al., 2023; Bai et al., 2020) batch and schedule requests for general neural networks, with some supporting preemptive GPU multiplexing (Han et al., 2022; Zhang et al., 2023; Bai et al., 2020). However, they do not account for LLMs' large model size and autoregressive generation.

Offline LLM Serving. Systems like FlexGen (Sheng et al., 2023) and DeepSpeed (Aminabadi et al., 2022) offload to host memory for throughput but are ill-suited for online latency. BlendServe (Zhao et al., 2026) and PSM (Liu et al., 2024) optimize offline batch reordering; ConServe is compatible with these for further improvements.

8. Conclusion

We present ConServe, a co-serving system that harvests idle GPU cycles for offline inference while enforcing strict online SLOs. By orchestrating execution at fine-grained token and layer levels, ConServe resolves the tension between utilization and latency, significantly outperforming state-of-the-art systems in both responsiveness and throughput.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

Acknowledgment

We thank the anonymous reviewers for their comments. This work was supported in part by a Jane Street Fellowship awarded to Yifan Qiao at UCLA; Amazon AI Fellowships awarded to Yifan Qiao and Shan Yu at UCLA; and the U.S. National Science Foundation under grants 2106404, 2106838, 2330831, 2403254, 2426162, and 2546642. We also gratefully acknowledge the generous support from Amazon and Samsung.

References

Agrawal, A., Kedia, N., Panwar, A., Mohan, J., Kwatra, N., Gulavani, B., Tumanov, A., and Ramjee, R. 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, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/agrawal.

Aminabadi, R. Y., Rajbhandari, S., Awan, A. A., Li, C., Li, D., Zheng, E., Ruwase, O., Smith, S., Zhang, M., Rasley, J., and He, Y. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '22. IEEE Press, 2022. ISBN 9784665454445.

Anthropic. Message batches. https://docs.anthropic.com/en/docs/build-with-claude/message-batches, 2024a.

Anthropic. Streaming messages. https://docs.anthropic.com/en/api/messages-streaming, 2024b.

Bai, Z., Zhang, Z., Zhu, Y., and Jin, X. PipeSwitch: Fast pipelined context switching for deep learning applications. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 499–514. USENIX Association, November 2020. ISBN 978-1-939133-19-9. URL https://www.usenix.org/conference/osdi20/presentation/bai.

Chiang, W.-L., Zheng, L., Sheng, Y., Angelopoulos, A. N., Li, T., Li, D., Zhu, B., Zhang, H., Jordan, M. I., Gonzalez, J. E., and Stoica, I. Chatbot arena: an open platform for evaluating llms by human preference. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Crankshaw, D., Wang, X., Zhou, G., Franklin, M. J., Gonzalez, J. E., and Stoica, I. Clipper: A Low-Latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pp. 613–627, Boston, MA, March 2017. USENIX Association. ISBN 978-1-931971-37-9. URL https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw.

Fabbri, A., Li, I., She, T., Li, S., and Radev, D. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. In Korhonen, A., Traum, D., and Màrquez, L. (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 1074–1084, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1102. URL https://aclanthology.org/P19-1102/.

Fu, Y., Xue, L., Huang, Y., Brabete, A.-O., Ustiugov, D., Patel, Y., and Mai, L. ServerlessLLM: Low-Latency serverless inference for large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 135–153, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/fu.

GitHub. Github copilot - write code faster. https://copilot.github.com/, 2025.

Google Labs. NotebookLM — Note Taking and Research Assistant Powered by AI. https://notebooklm.google.com/, 2024. URL https://notebooklm.google.com/.

Grattafiori, A., Dubey, A., Jauhri, A., and others. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Gujarati, A., Karimi, R., Alzayat, S., Hao, W., Kaufmann, A., Vigfusson, Y., and Mace, J. Serving DNNs like clockwork: Performance predictability from the bottom up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 443–462. USENIX Association, November 2020. ISBN 978-1-939133-19-9. URL https://www.usenix.org/conference/osdi20/presentation/gujarati.

Han, M., Zhang, H., Chen, R., and Chen, H. Microsecond-scale preemption for concurrent GPU-accelerated DNN inferences. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 539–558, Carlsbad, CA, July 2022. USENIX Association. ISBN 978-1-939133-28-1. URL https://www.usenix.org/conference/osdi22/presentation/han.

He, W., Liu, K., Liu, J., Lyu, Y., Zhao, S., Xiao, X., Liu, Y., Wang, Y., Wu, H., She, Q., Liu, X., Wu, T., and Wang,

H. Dureader: a chinese machine reading comprehension dataset from real-world applications, 2018. URL https://arxiv.org/abs/1711.05073.

Holmes, C., Tanaka, M., Wyatt, M., Awan, A. A., Rasley, J., Rajbhandari, S., Aminabadi, R. Y., Qin, H., Bakhtiari, A., Kurilenko, L., and He, Y. Deepspeed-fastgen: High-throughput text generation for llms via mii and deepspeed-inference, 2024.

Jin, H., Zhang, Y., Meng, D., Wang, J., and Tan, J. A comprehensive survey on process-oriented automatic text summarization with exploration of llm-based methods, 2024. URL https://arxiv.org/abs/2403.02901.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, pp. 611–626, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702297. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Li, Z., Zheng, L., Zhong, Y., Liu, V., Sheng, Y., Jin, X., Huang, Y., Chen, Z., Zhang, H., Gonzalez, J. E., and Stoica, I. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pp. 663–679, Boston, MA, July 2023. USENIX Association. ISBN 978-1-939133-34-2. URL https://www.usenix.org/conference/osdi23/presentation/li-zhouhan.

Liang, P., Bommasani, R., Lee, T., Tsipras, D., Soylu, D., Yasunaga, M., Zhang, Y., Narayanan, D., Wu, Y., Kumar, A., Newman, B., Yuan, B., Yan, B., Zhang, C., Cosgrove, C., Manning, C. D., Re, C., Acosta-Navas, D., Hudson, D. A., Zelikman, E., Durmus, E., Ladhak, F., Rong, F., Ren, H., Yao, H., WANG, J., Santhanam, K., Orr, L., Zheng, L., Yuksekgonul, M., Suzgun, M., Kim, N., Guha, N., Chatterji, N. S., Khattab, O., Henderson, P., Huang, Q., Chi, R. A., Xie, S. M., Santurkar, S., Ganguli, S., Hashimoto, T., Icard, T., Zhang, T., Chaudhary, V., Wang, W., Li, X., Mai, Y., Zhang, Y., and Koreeda, Y. Holistic evaluation of language models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=iO4LZibEqW. Featured Certification, Expert Certification, Outstanding Certification.

Liu, S., Biswal, A., Cheng, A., Mo, X., Cao, S., Gonzalez, J. E., Stoica, I., and Zaharia, M. Optimizing llm queries in relational workloads, 2024.

Miao, X., Shi, C., Duan, J., Xi, X., Lin, D., Cui, B., and Jia, Z. Spotserve: Serving generative large language models on preemptible instances. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, pp. 1112–1127, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703850. doi: 10.1145/3620665.3640411. URL https://doi.org/10.1145/3620665.3640411.

Moonshot AI. Mooncake KV Cache Transfer Engine. https://github.com/kvcache-ai/Mooncake/, 2024. URL https://github.com/kvcache-ai/Mooncake/.

Mosaic AI Research. Llm inference performance engineering: Best practices. https://www.databricks.com/blog/llm-inference-performance-engineering-best-practices, 2024.

Narayan, A., Chami, I., Orr, L., Arora, S., and Ré, C. Can foundation models wrangle your data?, 2022. URL https://arxiv.org/abs/2205.09911.

Narayan, S., Cohen, S. B., and Lapata, M. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *ArXiv*, abs/1808.08745, 2018. URL https://api.semanticscholar.org/CorpusID:215768182.

NVIDIA. Triton Inference Server. https://developer.nvidia.com/triton-inference-server, 2024.

NVIDIA. Conditional CUDA Graph Nodes. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#conditional-graph-nodes, 2025. URL https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#conditional-graph-nodes.

Oliaro, G., Miao, X., Cheng, X., Kada, V., Wu, M., Gao, R., Huang, Y., Delacourt, R., Yang, A., Wang, Y., Unger, C., and Jia, Z. FlexLLM: Token-Level Co-Serving of LLM inference and finetuning with SLO guarantees. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pp. 1359–1379, Renton, WA, May 2026. USENIX Association. ISBN 978-1-939133-54-0. URL https://www.usenix.org/conference/nsdi26/presentation/oliaro.

OpenAI. Gpt-4 technical report, 2023.

OpenAI. Batch api. https://platform.openai.com/docs/guides/batch/batch-api, 2024a.

OpenAI. Streaming api. https://platform.openai.com/docs/api-reference/streaming, 2024b.

OpenAI. Chatgpt: Conversational language model. https://chat.openai.com, 2025.

OpenJDK. HotSpot Glossary of Terms: Safepoint. https://openjdk.org/groups/hotspot/docs/HotSpotGlossary.html, 2024. URL https://openjdk.org/groups/hotspot/docs/HotSpotGlossary.html.

Pan, L., Liu, J., Yuan, J., Zhang, R., Li, P., and Xiao, Z. Occl: a deadlock-free library for gpu collective communication, 2023.

Patel, P., Choukse, E., Zhang, C., Shah, A., Goiri, I. n., Maleki, S., and Bianchini, R. Splitwise: Efficient generative llm inference using phase splitting. In *Proceedings of the 51st Annual International Symposium on Computer Architecture*, ISCA '24, pp. 118–132. IEEE Press, 2025. ISBN 9798350326581. doi: 10.1109/ISCA59077.2024.00019. URL https://doi.org/10.1109/ISCA59077.2024.00019.

Proxet. Llm has a performance problem inherent to its architecture: Latency. https://www.proxet.com/blog/llm-has-a-performance-problem-inherent-to-its-architecture-latency, 2023.

Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Sauvestre, R., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M., Ferrer, C. C., Grattafiori, A., Xiong, W., Défossez, A., Copet, J., Azhar, F., Touvron, H., Martin, L., Usunier, N., Scialom, T., and Synnaeve, G. Code llama: Open foundation models for code, 2024.

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Chen, B., Liang, P., Ré, C., Stoica, I., and Zhang, C. Flexgen: high-throughput generative inference of large language models with a single gpu. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Sheng, Y., Cao, S., Li, D., Zhu, B., Li, Z., Zhuo, D., Gonzalez, J. E., and Stoica, I. Fairness in serving large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 965–988, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/sheng.

Sun, T., Wang, P., and Lai, F. Hygen: Efficient LLM serving via elastic online-offline request co-location. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=cQxLCVa9u7.

The Claude Team. Introducing the next generation of claude. https://www.anthropic.com/news/claude-3-family, 2024.

The PyTorch Foundation. Torchserve is a performant, flexible, and easy to use tool for serving pytorch models in production. https://pytorch.org/serve/, 2024.

vLLM team. vLLM Piecewise CUDA Graph. https://docs.vllm.ai/en/stable/design/torch_compile.html?h=piecewise+cuda#cudagraph-capture, 2025. URL https://docs.vllm.ai/en/stable/design/torch_compile.html?h=piecewise+cuda#cudagraph-capture.

Wang, Y., Chen, Y., Li, Z., Tang, Z., Guo, R., Wang, X., Wang, Q., Zhou, A. C., and Chu, X. Towards efficient and reliable llm serving: A real-world workload study, 2024.

Wu, H., Zhan, M., Tan, H., Hou, Z., Liang, D., and Song, L. Vcsum: A versatile chinese meeting summarization dataset, 2023. URL https://arxiv.org/abs/2305.05280.

Xia, T., Mao, Z., Kerney, J., Jackson, E. J., Li, Z., Xing, J., Shenker, S., and Stoica, I. Skylb: A locality-aware cross-region load balancer for llm inference. *arXiv preprint arXiv:2505.24095*, 2025.

Xiao, W., Ren, S., Li, Y., Zhang, Y., Hou, P., Li, Z., Feng, Y., Lin, W., and Jia, Y. AntMan: Dynamic scaling on GPU clusters for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 533–548. USENIX Association, November 2020. ISBN 978-1-939133-19-9. URL https://www.usenix.org/conference/osdi20/presentation/xiao.

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., Lin, H., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Lin, J., Dang, K., Lu, K., Bao, K., Yang, K., Yu, L., Li, M., Xue, M., Zhang, P., Zhu, Q., Men, R., Lin, R., Li, T., Xia, T., Ren, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Wan, Y., Liu, Y., Cui, Z., Zhang, Z., and Qiu, Z. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Yu, G.-I., Jeong, J. S., Kim, G.-W., Kim, S., and Chun, B.-G. 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, Carlsbad, CA, July 2022. USENIX Association. ISBN 978-1-939133-28-1. URL https://www.usenix.org/conference/osdi22/presentation/yu.

Zhang, D., Wang, H., Liu, Y., Wei, X., Shan, Y., Chen, R., and Chen, H. Fast and live model auto scaling with o(1) host caching, 2024. URL https://arxiv.org/abs/2412.17246.

Zhang, H., Tang, Y., Khandelwal, A., and Stoica, I. SHEPHERD: Serving DNNs in the wild. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pp. 787–808, Boston, MA, April 2023. USENIX Association. ISBN 978-1-939133-33-5. URL https://www.usenix.org/conference/nsdi23/presentation/zhang-hong.

Zhao, Y., Yang, S., Zhu, K., Zheng, L., Kasikci, B., Qiao, Y., Zhou, Y., Xing, J., and Stoica, I. Blendserve: Optimizing offline inference with resource-aware batching. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '26, pp. 255–273, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400723599. doi: 10.1145/3779212.3790133. URL https://doi.org/10.1145/3779212.3790133.

Zheng, L., Yin, L., Xie, Z., Sun, C., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., Barrett, C., and Sheng, Y. Sglang: efficient execution of structured language model programs. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Zhong, Y., Liu, S., Chen, J., Hu, J., Zhu, Y., Liu, X., Jin, X., and Zhang, H. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin.

A. Appendix

A.1. Layer-Wise Preemption Details

Algorithm 2 illustrates how exactly ConServe leverages layer-wise preemption to provide strict SLO guarantees. The procedure runs asynchronously in a dedicated monitor thread, so the check never blocks worker execution. On each new online request, the monitor uses the cost model (§4.2) to project whether the in-flight batch's remaining time plus the new request's prefill would exceed the TTFT budget t_{TTFT}; if so, it raises a shared-memory flag that worker safepoints (§4.3) detect to yield the GPU mid-iteration. When no offline work is in flight, the procedure returns immediately, so online-only execution incurs zero scheduling overhead.

Algorithm 2 Preventing online TTFT violations.

Input: Online TTFT objective t_{TTFT}.
Let Q_{on} denote incoming online requests, B denote the current batch, and t_{passed} denote elapsed time since the batch started.
▷ *Run in the monitor thread.*
Function ONRECVONLINEREQUEST(t_{TTFT}):
if no offline requests in the current batch **then**
  return
end if
$t_{on} \leftarrow \text{Profiler.estimate\_exec\_time}(Q_{on})$
$t_{remaining} \leftarrow \text{Profiler.estimate\_exec\_time}(B) - t_{passed}$
if $t_{remaining} + t_{on} > t_{\text{TTFT}}$ **then**
  Worker.signal_layerwise_preemption()
end if

A.2. Summarized Results for End-to-End Serving

Table 2 summarizes results across all offline datasets in complement to Figure 7. Compared to the best-performing baseline, ConServe reduces P99 online TTFT and TBT by an average of 2.04× and 2.86×, respectively, while improving the offline throughput by 2.23× on average.

In summary, under realistic and dynamic workload conditions, ConServe can maintain strict online SLOs while delivering close to optimal offline throughput.

Table 2: Overall latency and throughput of Llama-3.1 70B when co-serving real-world online and different offline workloads achieved by `Online-Only` / `Non-Preemptive` / `Sarathi-Preemptive` / `DistServe-Preemptive` / ConServe.

Offline Workload	P99 Online TTFT (s)					P99 Online TBT (ms)					Offline Tput (tok/s)				
	OO	NP	SP	DP	**Ours**	OO	NP	SP	DP	**Ours**	OO	NP	SP	DP	Ours
Mixed	1.08	15.2	2.91	12.4	**1.14**	184	450	441	1398	**201**	0	1995	895	26	1553
DuReader		38.7	2.98	14.9	**1.67**		659	400	1607	**204**		1276	715	15	1187
MultiNews		11.0	2.11	11.3	**1.04**		305	212	1384	**194**		2765	1698	39	2281
VCSUM		46.2	3.01	15.1	**1.68**		566	453	1774	**208**		1218	255	15	1064

A.3. Llama-3.1 8B with Real-World Workloads

Figure 9 shows the performance of all systems on the BurstGPT workload with Llama-3.1 8B. ConServe maintains low P99 online TTFT and TBT, close to the ideal latencies of the `Online-Only` baseline. It concurrently achieves an offline throughput of 2645 tokens/s, which is 88% of the optimal throughput measured with `Non-Preemptive`.

In contrast, both `Sarathi-P` and `DistServe-P` suffer from high and unstable online latencies. Their offline throughputs are also much lower, at 661 and 1034 tokens/s, respectively.

Note that while we report the best-performing configuration for `DistServe-P`, finding an optimal static setting is infeasible in practice due to the high variability in request rate, length, and burstiness of real-world workloads. This also highlights

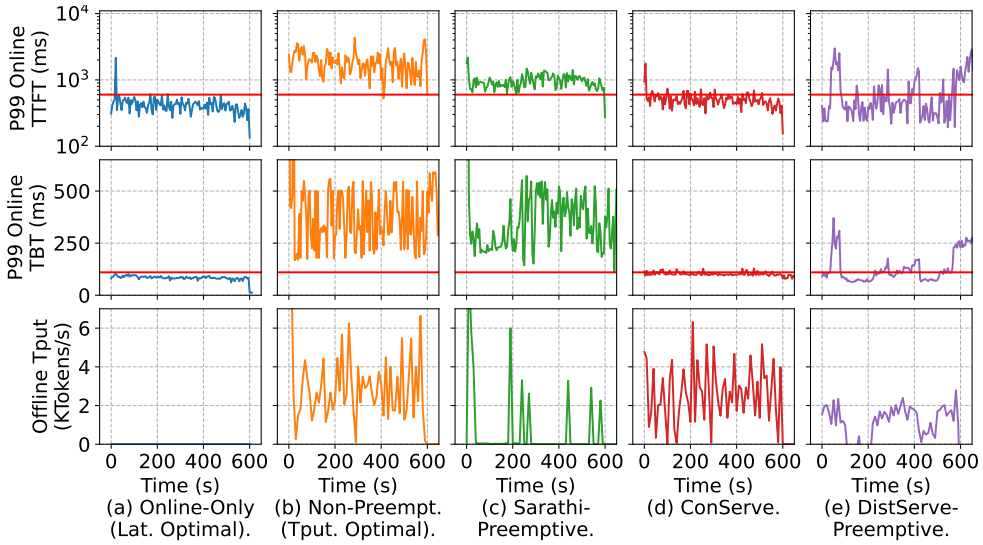


Figure 9: Serving performance for Llama-3.1 8B on real workloads. ConServe achieves consistently low P99 TTFT and TBT below online latency SLOs and 88% of the ideal offline serving throughput (measured with `Non-Preemptive`).

the key advantage of ConServe: its ability to dynamically adapt to maintain high performance and efficiency across real-world conditions without manual re-tuning.