MACE: A HYBRID LLM SERVING SYSTEM WITH COLOCATED SLO-AWARE CONTINUOUS RETRAINING ALIGNMENT

Yufei Li [1] **Yu Fu** [1] **Yue Dong** [1] **Cong Liu** [1]

ABSTRACT

Large language models (LLMs) deployed on edge servers are increasingly used in latency-sensitive applications such as personalized assistants, recommendation, and content moderation. However, the non-stationary nature of user data necessitates frequent retraining, which introduces a fundamental tension between *inference latency* and *model accuracy* under constrained GPU resources. Existing retraining strategies either delay model updates, over-commit resources to retraining, or overlook iteration-level retraining granularity. In this paper, we identify that iteration-level scheduling is crucial for adapting retraining frequency to model drift without violating service-level objectives (SLOs). We propose MACE, a hybrid LLM system that *colocates* concurrent inference (prefill, decode) and fine-tuning, with intelligent memory management to maximize task performance while promising inference throughput. MACE leverages the insight that not all model updates equally affect output alignment and allocates GPU cycles accordingly to balance throughput, latency, and update freshness. Our trace-driven evaluation shows that MACE matches or exceeds continuous retraining while reducing inference latency by up to 63% and maintaining throughput under resource constraints. Compared to periodic retraining, MACE improves latency breakdown across prefill, decode, and finetune stages, and sustains GPU utilization above 85% in NVIDIA AGX Orin. These results demonstrate that iteration-level hybrid scheduling is a promising direction for deploying LLMs with continual learning capabilities on edge platforms.

1 INTRODUCTION

Large language models (LLMs) have demonstrated impressive capabilities across diverse real-world applications, including open-domain question answering, code generation, and mathematical reasoning. As a result, LLMs have been widely adopted in interactive systems such as chat assistants (Achiam et al., 2023; Qiao et al., 2024), search engines (Perplexity AI), multimodal agents (Lin et al., 2025; Sarukkai et al., 2025), and real-time code assistants (GitHub Copilot). These systems process user queries in real-time and have increasingly shifted toward edge deployment, where models are hosted on local edge servers closer to the user (Fanton, 2021; Shubha & Shen, 2023).

Edge servers offer low-latency responses and improved data privacy, complying with regional policies like General Data Protection Regulation (GDPR) that prohibit the transmission of sensitive data to distant cloud regions (Bhardwaj et al., 2022). However, edge servers often have limited GPU capacity (Mittal et al., 2021), making it challenging to concurrently serve inference requests and support continual adaptation of models to user feedback or real-time data shifts (Bhardwaj et al., 2022). For instance, user responses may signal changes in preference (e.g., political leaning, toxicity tolerance) (Liang et al., 2025), requiring the system to fine-tune the model to reflect these updated objectives—departing from generalized alignment toward personalized alignment.

While compression methods (e.g., quantization, distillation) can reduce inference costs, their limited generalization capacity makes them highly sensitive to data distribution shifts (Mittal et al., 2021; Shubha & Shen, 2023). Therefore, continual retraining (e.g., every 30–60 seconds) becomes necessary to maintain alignment (Bhardwaj et al., 2022; Shubha & Shen, 2023), using user-specific preferences and temporal contexts extracted from live inference interactions. Unlike conventional supervised training pipelines, labels for LLM alignment data are not manually annotated. Instead, they are derived from implicit user feedback signals. A common approach is to use preference-based supervision, where users are presented with multiple generated responses and asked to select the most preferred one. User feedback is converted into supervision through preference signals (e.g., chosen, rejected), enabling ranking-based pairwise losses like Direct Preference Optimization (DPO) (Rafailov et al., 2023). Additional implicit signals—e.g., thumbs-up/down, skip rate, or dwell time—can be mapped to preferences or

[1]University of California, Riverside. Correspondence to: Yufei Li <yli927@ucr.edu>, Cong Liu <congl@ucr.edu>.

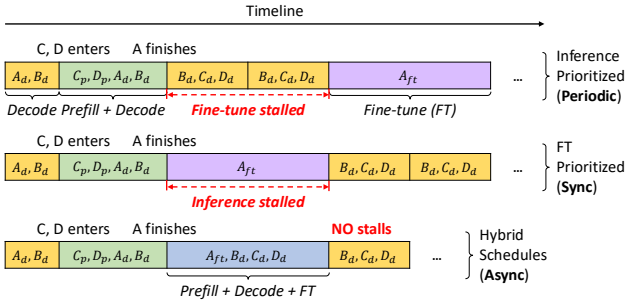


Figure 1. Requests A, B, C, D arrive over time. Subscripts p and d indicate prefill and decode iterations, while ft marks fine-tuning. **Periodic** retraining delays model updates for A due to inference priority. **Sync** retraining preempts decodes for B, C, D. **Async** (hybrid) schedule *colocates* A_{ft}, B_d, C_d, D_d into the same iteration, reducing latency and ensuring B, C, D benefit (in subsequent iterations) from the updated model—without stalling either workload.

reward scores. In multi-turn dialogs, user rephrasing or corrections also serve as supervision. Over time, this generates a personalized feedback corpus for fine-tuning models toward user-specific goals, without centralized fine-tuning or privacy violations.

However, co-running fine-tuning and inference on a resource-constrained GPU introduces a fundamental scheduling dilemma. Figure 1 illustrates the trade-offs among three commonly used scheduling strategies, building upon iteration-based continuous batching (Yu et al., 2022), where time is divided into discrete iterations, and each iteration forms a batch of tasks for the model to process concurrently:

- *Periodic* retraining defers model updates to fixed intervals, preserving inference responsiveness. Yet, this can delay alignment—decoding requests (B, C, D) are served using outdated models, reducing accuracy.
- *Continuous* (sync) retraining immediately preempts inference to perform fine-tuning, ensuring alignment but often stalling inference requests and causing SLO violations.
- *Hybrid* (async) retraining aims to interleave both workloads within the same GPU iteration, packing complementary workloads to reduce idle fragments and maximize both alignment and inference throughput.

The core challenge arises from their conflicting resource usage patterns under tight GPU budgets. Inference—especially LLM autoregressive decoding—requires sequential generation and sustained attention memory, which makes batching difficult and memory-intensive (Kwon et al., 2023; Agrawal et al., 2024). Fine-tuning, on the other hand, demands high GPU memory bandwidth and model parameter updates, which can stall ongoing inference (Choi et al., 2023).

To mitigate this tension, recent approaches employ *parameter-efficient fine-tuning* (PEFT) methods such as low-rank adaptation (LoRA) (Hu et al., 2022), which injects trainable low-rank matrices into the frozen weights of the base model. PEFT significantly reduces the number of trainable parameters and memory overhead during training, making it a natural fit for resource-constrained edge environments. It allows fine-tuning to be performed with minimal disruption to inference serving. However, even with PEFT, co-executing fine-tuning and inference remains non-trivial. For example, LoRA introduces additional memory (for adapters and optimizer state), and its backward pass can conflict with autoregressive decoding, particularly when multiple user requests are queued. Moreover, *resource fragmentation*—caused by variable-length inputs and dynamic arrival rates—can lead to suboptimal utilization and SLO violations (for inference requests) if not carefully managed (Sun et al., 2024).

To this end, we propose MACE, a hybrid execution framework that co-optimizes fine-tuning and inference on shared edge GPU resources. MACE introduces a fine-grained iteration-level hybrid scheduler that selects and packs both workloads together based on alignment potential, memory feasibility, and latency constraints. It adopts a *unified scheduling* strategy that handles inference and continual learning under a shared memory and execution budget. Compared to traditional periodic or sync baselines, MACE adapts dynamically to the queue state and application demand, maximizing GPU utilization while minimizing latency for heterogeneous workloads. MACE is built on two key components:

- *Memory-aware hybrid scheduler*, which jointly allocates prefill, decode, and fine-tune workloads per iteration using memory and alignment heuristics (§4.2).
- *Cache manager*, including prefix sharing at prefilling, enabling reuse of previously computed KV cache across requests to reduce response time and free memory, and KV cache pruning at decoding, which identifies and removes redundant attention heads or layers with negligible contribution to final output (§4.3).

We conduct extensive experiments on real-world traces from personalized chat datasets using Mistral-7B and LLaMA3-8B. As shown in Figures 9, 10, MACE consistently achieves the best trade-off between alignment accuracy and inference throughput, outperforming both periodic and synchronous baselines across retraining frequencies.

2 Background

Recent surge in deploying LLMs for real-time user interaction (Shen et al., 2024) has necessitated effective strategies for handling concurrent training and inference tasks. These workloads span responding to user queries and retraining

aimed at aligning LLMs with human preferences (Askell et al., 2021) to ensure helpfulness (Ethayarajh et al., 2022; Li et al., 2025b) and harmlessness (Fu et al., 2024a).

Alignment fine-tuning and personalization. Modern LLMs often rely on post-training alignment techniques, such as *reinforcement learning from human feedback* (RLHF) (Ouyang et al., 2022) or DPO (Rafailov et al., 2023) to better align generated responses with human preferences. To reduce the cost of full fine-tuning, PEFT like LoRA (Hu et al., 2022) and surgical fine-tuning (Lee et al., 2023) have been widely adopted. LoRA introduces learnable low-rank matrices into attention layers, enabling efficient adaptation to new data without updating the backbone model. Surgical fine-tuning, on the other hand, identifies and updates only specific components of the model that are most impactful for alignment (Shi et al., 2024). Such resource-efficient techniques are particularly advantageous for user personalization or continual alignment during deployment, where rapid adaptation to new preferences is essential.

Advanced LLM serving systems. Deploying LLMs in real-time and high-throughput environments requires serving systems that can simultaneously satisfy low latency, high efficiency, and hardware resource constraints. Continuous batching has become the de facto design in modern LLM systems to handle dynamic request arrival, which group them on-the-fly to improve GPU utilization without introducing excessive queuing delays (Li et al., 2023b), as shown in Algorithm 2 in Appendix A. Systems like Orca (Yu et al., 2022), vLLM (Kwon et al., 2023), Llumnix (Sun et al., 2024), and DistServe (Zhong et al., 2024) have proposed various optimizations in this space. For example, Orca enables mixed prefill and decode execution via iteration (token)-level scheduling, and Llumnix proposes chunked prefill to pipeline long-context inputs. Despite implementation differences, these systems share a core challenge: *How to efficiently pack heterogeneous requests with diverse memory and latency profiles?* We argue that continuous batching can be viewed through the lens of bin-scheduling—treating each GPU or sub-batch as a bin and scheduling tasks based on joint memory-latency fit. This abstraction forms the basis for our unified scheduling approach in later sections.

3 Motivation Study

In this section, we first empirically show that continuous retraining outperforms periodic retraining in maintaining alignment quality over time. However, naïvely applying continuous retraining introduces new bottlenecks, as it interferes with inference workloads and degrades responsiveness. We further show that continuous retraining can be optimized via hybrid batching to achieve lower latency, enabling systems to deliver up-to-date and real-time aligned responses.

3.1 Accuracy Benefits of Continuous Retraining

Traditional approaches to adapting LLMs to evolving user preferences often involve periodic retraining at fixed intervals, potentially leading to stale models that lag behind rapidly changing user needs. In contrast, continuous retraining allows models to adapt instantly, incorporating new information as it becomes available.

Alignment metrics. To evaluate the effectiveness of continuous retraining in aligning LLMs with user preferences, we define the following setting. Let $\mathcal{D} = (x, y^+, y^-)$ denote a dataset of user preference annotations, where x represents the input prompt, y^+ is the *preferred* response, and y^- is the *dispreferred* response. We assess model π_θ's performance using the following alignment metrics:

- **Preference accuracy (win rate)** (Zhang & Ranganath, 2025) ↑: Win rate measures the fraction of preference pairs where the model assigns a higher probability to the preferred response:

  $$\text{Win Rate} = \mathbb{E}_{(x,y^+,y^-)\sim\mathcal{D}}\left[\mathbf{1}\left[\pi_\theta(y^+ \mid x) > \pi_\theta(y^- \mid x)\right]\right],$$

  where $\mathbf{1}[\cdot]$ is the indicator function. A higher win rate signifies better alignment with user preferences.
- **Contrastive log-probability difference (CLPD)** (Chen et al., 2024) ↑: CLPD quantifies the average difference in log-probabilities between the preferred and less preferred responses:

  $$\text{CLPD} = \mathbb{E}_{(x,y^+,y^-)\sim\mathcal{D}}\left[\log \pi_\theta(y^+ \mid x) - \log \pi_\theta(y^- \mid x)\right].$$

  A higher CLPD indicates a stronger preference for the preferred responses over the less preferred ones.

Retraining setup. To avoid the complexity of reinforcement learning, DPO (Rafailov et al., 2023) provides a lightweight alternative by leveraging contrastive learning over user preferences. Instead of optimizing a reward model, DPO directly fine-tunes LLMs by comparing the preferred and dispreferred outputs, while regularizing against a reference policy π_{ref} (e.g., the initial model). The goal is to maximize CLPD between y^+ and y^-, balanced by a regularization term that discourages excessive divergence from π_{ref}:

$$\mathcal{L}_{\text{DPO}} = \mathbb{E}_{(x,y^+,y^-)\sim\mathcal{D}}\left[\log \sigma\left(\beta \cdot \left(\Delta_\theta^+ - \Delta_\theta^-\right)\right)\right],$$
$$s.t.\ \Delta_\theta^+ = \log \frac{\pi_\theta(y^+ \mid x)}{\pi_{\text{ref}}(y^+ \mid x)}, \quad \Delta_\theta^- = \log \frac{\pi_\theta(y^- \mid x)}{\pi_{\text{ref}}(y^- \mid x)}.$$

Here, π_θ is the current model policy, β is a temperature parameter controlling the sharpness of preference, and $\sigma(\cdot)$ denotes the sigmoid function.

Empirical results. We study the impact of retraining strategies under a shared LLM serving system using two alignment fine-tuning benchmarks: Anthropic HH-RLHF (Bai et al., 2022) and StanfordNLP SHP (Ethayarajh et al., 2022).

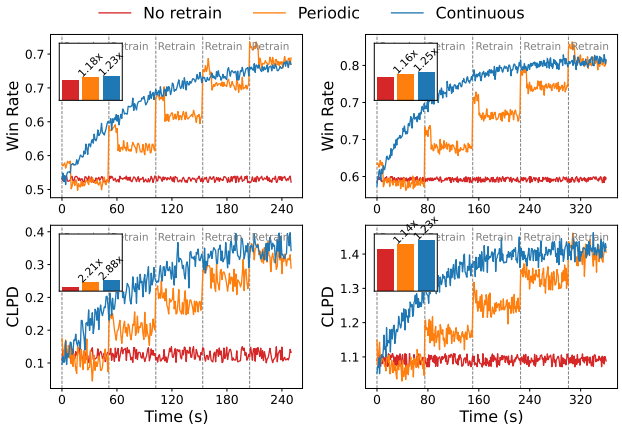


Figure 2. Win rate and CLPD over time when serving Mistral-7B on *Left*: RLHF and *Right*: SHP dataset. The inset bar plots show the average metric value of each method, highlighting the overall performance difference beyond temporal variations.

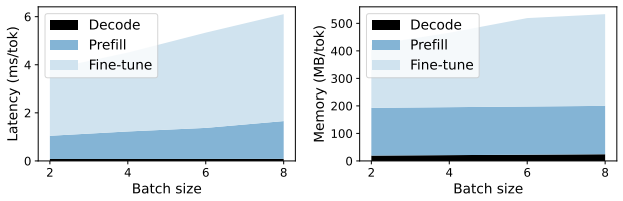


Figure 3. Left: latency and *Right*: memory per-token of three workloads for Mistral-7B on A6000 Ada across varying batch sizes.

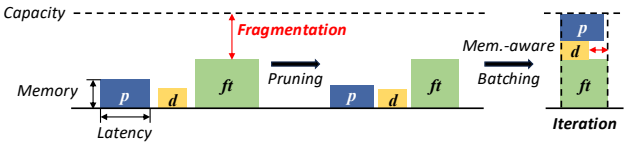


Figure 4. Abstracted memory–latency footprint for three workloads. Hybrid scheduling together with pruning techniques mitigates memory fragmentation and increases concurrency.

The former focuses on harmlessness, where preferred responses tend to reject unsafe or illegal requests, while the latter focuses on helpfulness, where preferred responses are more factually correct. Figure 2 compares win rate and CLPD over time under different retraining strategies. We observe that: *Both periodic and continuous retraining consistently outperform the no-retrain baseline.* For instance, in the RLHF dataset (top-left), periodic retraining improves win rate by up to 1.19×, while continuous retraining yields a 1.24× gain. Similar trends hold for CLPD, where the confidence gap grows even more substantially (bottom-left, 2.47× and 3.17×, respectively). *Continuous retraining provides a larger performance boost.* Since retraining is triggered more frequently, continuous updates adapt the model faster to distributional shifts, leading to higher overall metrics compared to periodic retraining. This advantage is consistently seen across both datasets and both metrics. *CLPD reveals larger fluctuations and improvements than win rate.* While win rate rises more smoothly, CLPD exhibits stronger variance and larger relative gains. This is expected, as CLPD directly measures the preference confidence difference, which is more sensitive and interpretable in reflecting alignment strength. *Harmlessness alignment is harder than helpfulness alignment.* Comparing the two datasets, performance improvement is smaller for RLHF (left column) than for SHP (right column). This indicates that aligning models to avoid harmful behaviors is inherently more challenging than aligning them to be helpful.

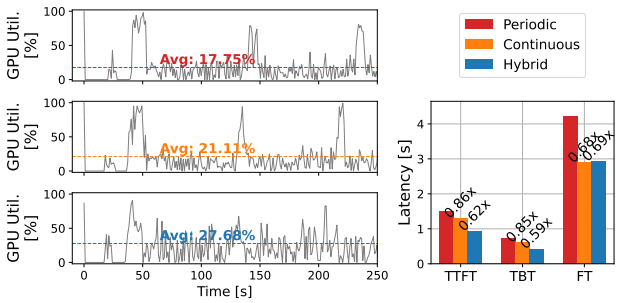
3.2 Optimization for Continuous Retraining

Profiling LLM workload heterogeneity. To effectively co-serve inference and continual fine-tuning for LLMs, it is essential to understand the latency and memory footprints of the constituent workloads. We profile the latency and memory usage of three major LLM operations—*prefill*, *decode*, and *fine-tune*—across varying batch sizes, shown in Figure 3. We observe that:

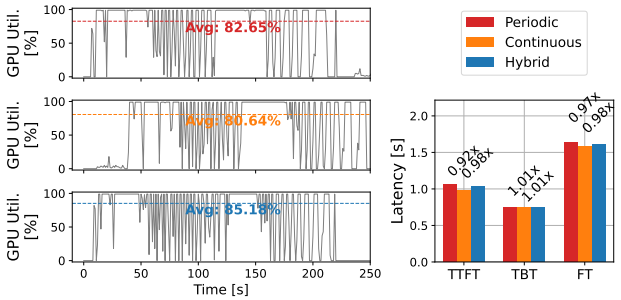
- *Prefill* latency and memory scale with input sequence length and batch size (linearly under FlashAttention (Dao et al., 2022)), since each token attends to the full prefix.
- *Decode* operations are much cheaper, as only one token is appended per iteration, but memory gradually increases due to KV cache accumulation (Kwon et al., 2023).
- *Fine-tune*, even with efficient PEFT methods like LoRA (Hu et al., 2022), exhibits significantly higher latency and memory consumption—making it the dominant bottleneck in edge inference-tuning pipelines.

Memory fragmentation and hybrid scheduling. Figure 4 (left) visualizes the resource usage of each workload as a memory-latency box. When scheduled independently, these misaligned workloads leave significant *holes* in GPU memory and timeline—leading to fragmentation and low utilization. For instance, a retraining task may consume only a fraction of GPU memory but blocks the entire iteration window. Conversely, prioritizing inference may delay critical fine-tunes that personalize the model. Inspired by continuous batching (Yu et al., 2022), we extend iteration-level scheduling to jointly colocate fine-tuning with prefill and decode. By dynamically adjusting batch sizes to opportunistically fill leftover memory within each iteration, the system mitigates fragmentation and exploits latency complementarity while respecting alignment freshness. Figure 4 (right) illustrates this idea.

A6000 server (Figure 5a)—fragmentation-dominated and hybrid gains. On the A6000 server, average utilization is low: Periodic 17.8% / Continuous 21.1% / Hybrid 27.7%. Here, the bottleneck is not compute saturation but fragmen-



(a) Mistral-7B on RTX A6000 Ada.



(b) Mistral-3B on NVIDIA AGX Orin.

Figure 5. Left: GPU utilization, and *Right*: Latency breakdown on two (a) A6000 Ada server and (b) AGX Orin edge device.

tation: alternating between training and inference leaves idle gaps in both time and memory. Continuous retraining partially fills these gaps, but Hybrid scheduling is far more effective by colocating small-batch fine-tuning with inference workloads and resizing batches per iteration. This directly translates into significant latency reduction: *TTFT*: 0.62×-0.86×, *TBT*: 0.59×-0.85×, and *FT*: 0.68×-0.86×.

Orin (Figure 5b)—compute-bound, consistently saturated. On the AGX Orin edge device, the average utilization remains consistently high across all strategies: Periodic 82.7% / Continuous 80.6% / Hybrid 85.2%. With tighter compute and memory budgets, any workload—particularly fine-tuning—pushes the GPU close to saturation. As a result, the opportunity to further increase parallelism is limited. The latency breakdown confirms this: relative to Periodic, Continuous and Hybrid only provide marginal improvements (TTFT 0.92×-0.98×, TBT∼1.01×, FT 0.97×-0.98×). In short, on compute-constrained edge platforms, scheduling refinements mainly ensure that continuous training does not worsen inference latency, while the overall improvement remains bounded by hardware ceilings.

4 System Design of MACE

We present MACE, a fine-grained GPU-local scheduling system designed to support colocated LLM inference serving and continuous retraining. Figure 6 illustrates the design: user requests arrive asynchronously into a shared priority

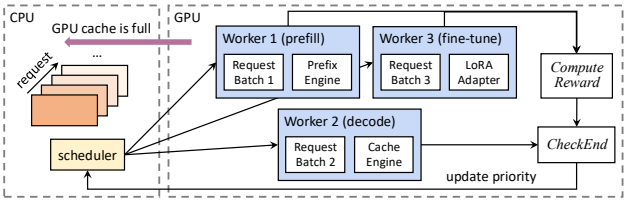


Figure 6. Design overview of MACE. The scheduler dispatches requests to prefill (prefix sharing), decode (KV cache pruning), and fine-tune (LoRA) workers, while feedback from reward computation and queuing time enables real-time priority updates under CPU–GPU cache coordination.

queue. The scheduler dynamically assigns priorities and batches them into iterations using a best-fit bin packing strategy, accounting for memory fragmentation and alignment reward. These batches are dispatched to GPU workers, where each worker concurrently handles prefilling, decoding, or fine-tuning workloads. After execution, tasks are re-prioritized if not finished and pushed back into the queue. MACE is built on three key components:

- **Priority computation:** Assign dynamic priority to each request based on workload type, enqueued time, and accuracy-driven rewards (e.g., DPO loss).
- **Iteration-level memory-aware batching:** A GPU-local scheduler that allocates tasks to minimize memory fragmentation and latency interference.
- **Cache management:** Optimizes memory reuse through prefix sharing in prefilling and cache pruning in decoding, and evicts unused cache to prevent memory bloat.

4.1 Alignment-aware Prioritization

Each incoming request is tagged with its arrival time t_{arr} and workload type $\mathsf{w} \in \{\mathsf{prefill}, \mathsf{decode}, \mathsf{ft}\}$. We define its base priority π_{w} and temporal growth rate δ_{w}, allowing a dynamic priority $P(t)$ at current time t to grow over time:

$$P_{\mathsf{w}}(t) = \pi_{\mathsf{w}} + \delta_{\mathsf{w}} \cdot (t - t_{\text{arr}}),$$

where we set $\pi_{\mathsf{decode}} > \pi_{\mathsf{prefill}} > \pi_{\mathsf{ft}}$ to prioritize inference serving and avoid SLO violation caused by frequent preemption, and set $\delta_{\mathsf{ft}} > \delta_{\mathsf{prefill}} > \delta_{\mathsf{decode}}$ to gradually escalate the criticality of fine-tuning requests over time to avoid *retraining starvation*.

Additionally, to promote retraining samples that are particularly valuable for model alignment, MACE integrates their alignment reward (i.e., DPO loss) to the priority:

$$P_{\mathsf{ft}}^{\text{total}}(t) = P_{\mathsf{ft}}(t) + \gamma \cdot \mathcal{L}_{\text{DPO}}(x, y^+, y^-),$$

where γ is a tunable weight. This mechanism prioritizes retraining samples with higher misalignment. As shown in Figure 7, after prioritization at iteration 9, the tasks are ordered by priority as A_{ft}, C_d, F_d, E_d, H_p, G_d, and D_{ft} (from highest to lowest).

Algorithm 1 GPU-Local Request Scheduling

Require: Task queue $\mathcal{Q}$, memory capacity $\mathcal{P}$, thresholds $(\tau_{\text{mem}}, \tau_{\text{task}})$, memory-latency weights (λ_1, λ_2).
Ensure: Scheduled bin $\mathcal{B}_1$ for execution.
1: Initialize bin list $\mathcal{B} \leftarrow []$, task counter $c \leftarrow 0$
2: **while** $\mathcal{Q}$ not empty **do**
3: **if** $\mathcal{B} \neq \emptyset \wedge (\mathcal{B}_1.\text{memory} \geq \tau_{\text{mem}} \cdot \mathcal{P} \vee c \geq \tau_{\text{task}})$ **then**
4: **break**
5: Retrieve task $\leftarrow \mathcal{Q}$.dequeue(), increment $c \leftarrow c + 1$
6: Estimate workload: $(m, \ell) \leftarrow$ GETWORKLOAD(task)
7: Initialize best_bin $\leftarrow$ `Null`, best_score $\leftarrow \infty$
8: **for** each bin $\mathcal{B}_i$ in $\mathcal{B}$ **do**
9: **if** $\mathcal{B}_i$.free_memory $\geq m$ **then**
10: Compute fragmentation score f
11: **if** score $<$ best_score **then**
12: Update: best_score $\leftarrow$ score, best_bin $\leftarrow \mathcal{B}_i$
13: **if** best_bin $\neq$ `None` **then**
14: Assign task to best_bin, update memory and latency
15: **else**
16: Create new bin $\mathcal{B}_{\text{new}}$, insert task, append to $\mathcal{B}$
17: **if** $\mathcal{B} \neq \emptyset$ **then**
18: **for** $i = 2$ to $|\mathcal{B}|$ **do**
19: **for** each task in $\mathcal{B}_i$ **do**
20: **if** CHECKEND(task) == `False` **then**
21: Push task back into $\mathcal{Q}$ with updated priority
22: **Output:** $\mathcal{B}_1$

4.2 Iteration-level Resource-aware Batching

Best-fit heuristic motivation. Scheduling a mix of LLM workloads on a single GPU can be viewed as a bin-packing problem in which each iteration's "bin" is the available GPU memory and each request (prefill, decode, ft) is an execution unit with particular sizes (memory footprint) and duration (latency). Unlike general static bin packing, MACE involves an online, heterogeneous stream of tasks with different memory/latency profiles (Figure 3). A naive greedy scheduler that runs tasks sequentially or without regard for workload size can lead to low GPU utilization—e.g. large training jobs leaving gaps of unused memory that smaller inference jobs could have filled, or short decoding tasks waiting behind long-running jobs.

MACE adopts a best-fit heuristic as a pragmatic solution to this dynamic packing problem. Best-fit scheduling greedily fills the GPU with the combination of tasks that most tightly fits the available memory, while also accounting for execution time compatibility. It minimizes memory fragmentation (unutilized memory, which appears as the red gap in Figure 4) and helps balance latency so that no single task unduly delays the others. Formally, the scheduler assigns a composite score f to each candidate set of tasks using a two-dimensional objective:

$$f = \lambda_1 \cdot |\mathcal{B}_i.\text{free\_memory} - M| + \lambda_2 \cdot |\mathcal{B}_i.\text{max\_latency} - \ell|,$$

where λ_1, λ_2 trade off vertical and horizontal memory fragmentation. In practice, this best-fit heuristic is fast, adaptive, and well-suited to colocated workloads, even though globally optimal packing is NP-hard. By prioritizing a tight memory fit, we ensure that the GPU's capacity is used as much as possible each iteration, while the latency-aware term prevents pathological cases (e.g. packing a very slow task with many fast tasks that would all finish and leave the GPU underutilized waiting for the slow one).

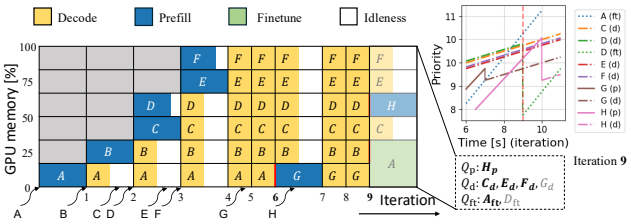


Figure 7. GPU-local scheduling of inference and training requests. Q_p, O_d, Q_{ft} denote queues for prefill, decode, and fine-tune. **Bold** texts denote scheduled requests for the next bin (iteration).

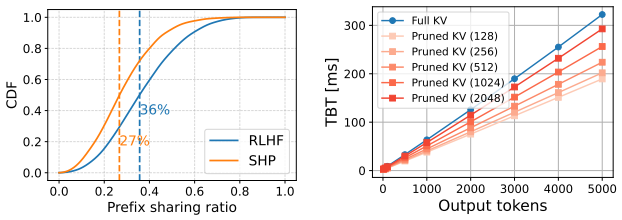
Bin allocation. As illustrated in Algorithm 1, the scheduler dequeues tasks sorted by dynamic priority (§4.1) and attempts to assign them to the current iteration bin using the best-fit score (lines 1-16). It considers memory fit to reduce fragmentation and latency divergence to avoid stalls. This enables effective co-scheduling of short decode jobs and large fine-tune jobs within a single GPU iteration. After execution, the system decides whether to reschedule unfinished tasks (lines 17–21) according to the `CheckEnd` output:

- For inference, it determines if the next decoded token is [EOS] or if the decoding iteration reaches the predefined maximum steps.
- For retraining, it checks whether DPO loss is still above a pre-defined threshold.

Tasks needing further iterations are requeued with updated priorities; others are retired. Figure 7 demonstrates the effectiveness of MACE. In iteration 6, a bin is formed by packing several decode jobs $B_d \sim F_d$ and one prefill G_p, with a large fine-tune A_{ft} temporarily delayed to prioritize throughput. While in iteration 9, the scheduler flexibly prioritizes the fine-tune A_{ft} which now has a higher priority, and delays the decode G_d to ensure timely model updates. This memory-aware batching balances the idleness-dependent throughput and continuous alignment learning process for heterogeneous workloads.

4.3 Prefix Sharing and Cache Management

To further mitigate the retraining–inference tradeoff, MACE leverages two key optimization techniques: *Prefix sharing* (Lin et al., 2024; Zheng et al., 2024) and *KV cache pruning* (Li et al., 2024; Fu et al., 2025). These techniques aim to reduce memory and latency overhead for inference, thereby creating more opportunities for retraining without compromising serving throughput.



(a) Prefix sharing in prefill (b) Cache pruning in decode

Figure 8. Prefix sharing ratio during prefill and decoding TBT of pruned KV cache with varying capacity size C_{total}.

Reduced prefill via prefix tree. Prefix sharing exploits input redundancy across user queries to avoid redundant prefill computation. As illustrated in Appendix A, multiple user queries often share common token prefixes, e.g., "*What is the best way to...*". We construct a prefix tree over active inference requests, where each leaf represents a complete request and each internal node represents a shared prefix segment. This design follows a Trie structure (Zheng et al., 2024), where a path from the root to any leaf captures the longest common prefix of a request. To maximize reuse, we traverse the tree in a DFS manner to co-execute shared prefixes. At each internal node of the prefix tree, we cache the KV pairs from the prefill outputs. This cache is efficiently reused by descendant nodes without recomputation. To manage memory usage, we offload least recently used (LRU) cache when inference concurrency is high to release more memory budget. By avoiding repeated computation for shared prefix tokens, prefix sharing reduces both latency and memory consumption in prefill. It also provides more headroom for opportunistic retraining, as the scheduler can pack training tasks without disrupting inference throughput. Figure 8a demonstrates the distribution of prefix sharing ratios in two benchmark datasets.

Decode KV cache pruning. Decoding presents another challenge: KV cache for each output token accumulates linearly and quickly saturates GPU memory. To mitigate this, we introduce a norm-based cache pruning mechanism, inspired by the revealed contextual sparsity of LLMs (Liu et al., 2023). At each decoding iteration, we compute the L2-norm of the output attention vector across heads. Let $\mathbf{a}_{t,h} \in \mathbb{R}^d$ denote the output at iteration t and attention head h, and define its norm as $\|\mathbf{a}_{t,h}\|_2$. We maintain a rolling average $\bar{\mathbf{a}}_h$ for each head and reallocate per-head KV storage capacity proportionally:

$$\bar{\mathbf{a}}_h = \frac{1}{T}\sum_{t=1}^{T} \|\mathbf{a}_{t,h}\|_2\,,\; w_h = \frac{\bar{\mathbf{a}}_h}{\sum_j \bar{\mathbf{a}}_j},\; C_h = \lfloor w_h \cdot C_{\text{total}} \rfloor\,.$$

Here, $\bar{\mathbf{a}}_h$ is the average ℓ_2 norm of the attention output at head h, C_{total} is the total available KV cache capacity, and C_h is the per-head allocation. The remaining capacity is greedily assigned to heads with the largest weight w_h:

$$\text{Prune if} \quad t - t_{\text{last\_used}} > W \quad \vee \quad \|\mathbf{a}_{t,h}\|_2 < \tau,$$

where W is a fixed sliding window size and τ is a norm threshold below which attention heads are considered uninformative (Fu et al., 2025). This dynamic reallocation selectively retains high-signal attention heads while pruning less informative ones. If a head's norm is consistently small, it receives lower cache budget and may prune earlier attention positions. As shown in Figure 8b, this reduces decoding latency and is helpful under long outputs or memory pressure. By combining prefix sharing for prefill and norm-guided cache pruning for decoding, we allow more training jobs to execute without sacrificing inference throughput.

5 Discussion and Implementation

We build our system by integrating efficient LLM inference engine, lightweight retraining modules, and a scheduler with fine-grained concurrency and batching support. Below, we elaborate on each component.

Inference engine. Our system adopts vLLM (Kwon et al., 2023), an inference backend optimized for decoding throughput using PagedAttention and FlashAttention (Dao et al., 2022) mechanisms. This enables our system to leverage token-level parallelism and key-value memory optimization for high-throughput generation.

Personalized adapter. To enable low-latency retraining without degrading inference throughput, we leverage PEFT techniques by assigning each user or domain an independent LoRA adapter ϕ_u (Hu et al., 2022). This allows user-personalized updates while sharing the frozen base model θ across all requests, enabling tighter integration with online serving in each shared GPU environment.

Concurrency support. MACE is implemented with multi-threaded concurrency using a thread pool design: A *producer* thread ingests inference/retraining requests. A *scheduler* thread computes dynamic priorities and forms hybrid iteration-level batches, where a bin allocator maps scheduled jobs to physical GPU sessions. Multiple *executor* threads run on-device workloads asynchronously. To balance latency and throughput, the scheduler maintains a priority queue that supports: *Dynamic priority refresh*: periodically updates request priorities based on utility functions (e.g., model freshness, deadline slack). *Preemption-aware iteration batching*: schedules batches with least .

6 Evaluation

6.1 Experimental Setup

Hardware platforms. We test both *server* with NVIDIA A6000 Ada GPU (48 GB VRAM) and *edge* device with

NVIDIA AGX Orin (32GB LPDDR5), ensuring a controlled environment to isolate the scheduler's effects.

Workloads and datasets. We simulate a mixed workload of inference and retraining requests representing single- and multiple-user domains with varying alignment needs. Inference requests are generated as a Poisson arrival process (Li et al., 2023b), and additional fraction of them (i.e., retrain rate, which is typically less than 50% (Bhardwaj et al., 2022)) are retraining jobs. We consider two datasets to emulate different alignment domains or user preferences:

- **RLHF** (Bai et al., 2022) (*harmlessness*): Human feedback focusing on safety alignment—preferred responses tend to avoid unsafe or illegal content. This represents users requiring the model to refuse harmful outputs.
- **SHP** (Ethayarajh et al., 2022) (*helpfulness*): A broad human preference dataset with 385k comparisons across 18 topics, emphasizing helpfulness of responses over others. This represents general user satisfaction alignment.

By using these distinct datasets, we create multi-tenant scenarios where different "tenants" (user groups) issue requests with different alignment objectives. This diversity lets us evaluate MACE 's overall alignment capability.

Baselines. We compare MACE against several baseline scheduling strategies:

- **Ekya** (Bhardwaj et al., 2022): A policy that periodically preempts inference to run retraining at fixed intervals. Inference requests queue up during retraining epochs (synchronous blocking). This baseline represents traditional FT where serving and training are separated in phases.
- **AdaInf** (Shubha & Shen, 2023): Continuous scheduling that gives retraining jobs highest priority in a FIFO queue, where incoming FT tasks can preempt or delay inference until they complete. It maximizes retraining frequency but with no special handling for interference, e.g., often hurting inference latency significantly.

6.2 Performance under Varying Retraining Intensity

We evaluate MACE against Ekya and AdaInf under varying retraining intensities, focusing on alignment accuracy, inference throughput, and latency breakdown.

Alignment accuracy. Figure 9 shows average win rate and CLPD across SHP and RLHF datasets. MACE consistently outperforms both baselines, especially under higher retraining rates. On SHP, MACE achieves up to 1.03× higher win rate and 1.21× higher CLPD compared to AdaInf. On RLHF, the advantage is more pronounced: MACE improves win rate by up to 1.02× and CLPD by 1.34×, reflecting its ability to retain alignment accuracy even under intensive retraining. In contrast, Ekya and AdaInf show diminishing returns as retraining frequency increases, highlighting

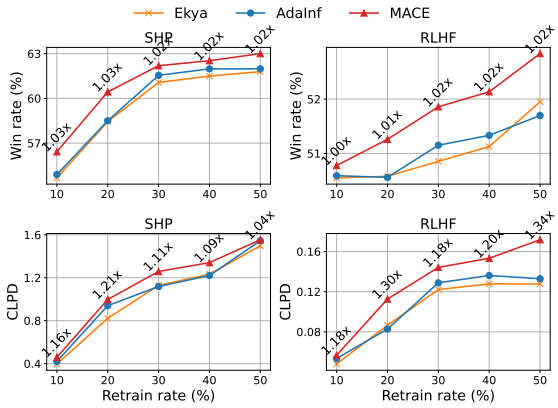
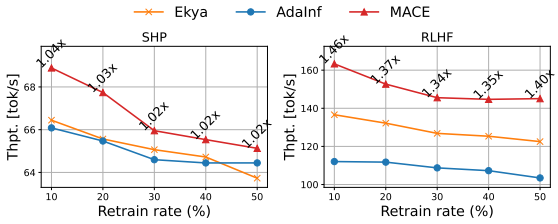
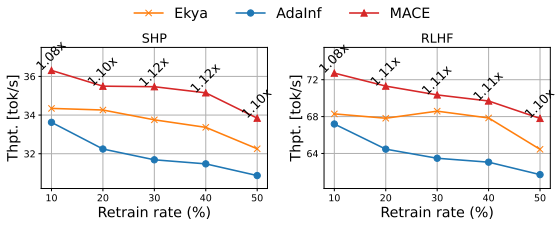


Figure 9. Average win rate and CLPD across various retrain rates of different methods on *Left*: SHP and *Right*: RLHF dataset.



(a) Throughput (token/sec) of Mistral-7B on RTX A6000 Ada.



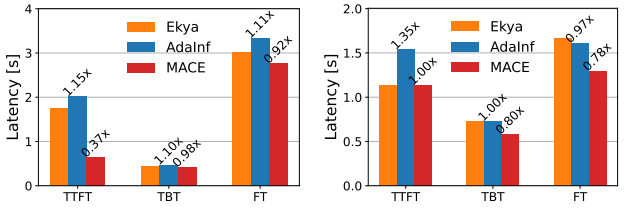
(b) Throughput (token/sec) of Mistral-3B on NVIDIA AGX Orin.

Figure 10. Inference throughput comparison of different methods.

the importance of MACE 's fine-grained scheduling and cache-aware optimizations.

Throughput. Figure 10 reports inference throughput on both server-scale and edge-scale platforms. On the A6000, MACE delivers up to 1.46× higher throughput than AdaInf at 10% retraining rate, while maintaining superior accuracy. On the AGX Orin, which is more resource constrained, MACE sustains 1.12× higher throughput on average, with benefits most evident at higher retraining intensities where baselines degrade rapidly. These results demonstrate that MACE 's hybrid batching and pruning strategies effectively amortize retraining costs without sacrificing inference performance, a critical property for deployment on heterogeneous hardware.

Latency breakdown. Figure 11 decomposes end-to-end latency into prefill, decode, and fine-tuning components. MACE achieves lower decode latency (TBT) via cache



(a) RTX A6000 Ada (b) NVIDIA AGX Orin

Figure 11. Latency breakdown on (a) A6000 and (b) AGX Orin.

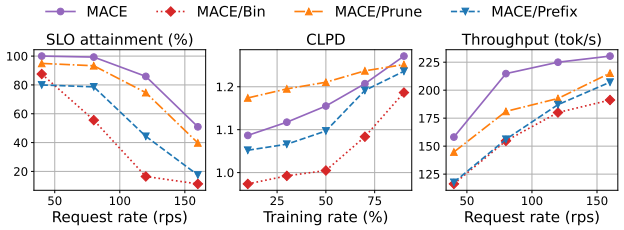


Figure 12. Left: SLO attainment rate, *Middle*: CLPD in SHP, and *Right*: inference throughput of different variants of MACE.

pruning, while prefix sharing substantially reduces prefill cost (TTFT). Fine-tuning overhead remains bounded due to iteration-level batching, which overlaps retraining with inference execution. Compared to AdaInf, MACE reduces prefill latency by 63% on A6000 server and decode latency by 30% on edge device platforms. This balanced reduction explains its ability to simultaneously improve alignment accuracy and throughput, highlighting the effectiveness of joint scheduling across retraining and inference phases.

6.3 Understanding MACE

To better understand the contribution of each component in MACE, we create the following variants by disabling specific mechanisms:

- *MACE/Bin*: This variant disables memory-aware batching by fixing maximum batch sizes and applying FIFO queuing without hybrid scheduling.
- *MACE/Prefix*: This variant disables prefix sharing during the prefill phase, resulting in redundant computation across similar requests.
- *MACE/Prune*: This variant disables KV cache pruning under memory constraints, forcing full cache retention during decoding.

Ablation study. We measure SLO attainment—proportion of inference tasks whose TTFT meet a SLO deadline of 5× forward latency (Li et al., 2023d). Figure 12 (left) shows the SLO attainment rate under increasing request loads. Disabling hybrid scheduling (MACE/Bin) causes the steepest decline, with attainment falling below 20% at high load, confirming that fixed batching fails to adapt to workload

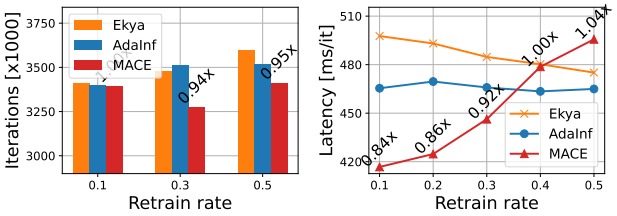


Figure 13. Left: total iterations and *Right*: average per-iteration latency for Mistral-7B on A6000 Ada.

Table 1. Average time overhead (ms) and memory (MB) of schedulers. ~~PA~~, ~~IB~~, ~~CM~~ mean without prioritization (§4.1), iteration-level batching (§4.2), and cache management (§4.3).

	Ekya	AdaInf	MACE		
			~~PA~~	~~IB~~	~~CM~~
Latency (ms)	3e-2	8e-2	1.1e-1	1.2e-1	1.5e-1
Memory (MB)	0.6	2	3.5	4.5	4

burstiness. MACE/Prefix also shows a 35–40% lower attainment than full MACE at 150 rps, as redundant prefill computations delay request completion. These results highlight that hybrid scheduling and prefix sharing are key to meeting latency SLOs under heavy traffic. On alignment accuracy (CLPD) in Figure 12 (middle), MACE achieves up to 8% improvement over MACE/Prefix and 15% over MACE/Bin, showing the importance of dynamic batching and prefix sharing in reducing latency-induced drift. Particularly, MACE/Prune consistently shows the highest CLPD scores, indicating that pruning KV cache will sacrifice accuracy to a certain point. For throughput, MACE delivers over 225 tok/sec, compared to drops of 7–11% in MACE/Prune and MACE/Prefix, and 17.6% in MACE/Bin. These results confirm that all three mechanisms contribute complementary benefits: hybrid scheduling is most critical for latency and accuracy, while prefix sharing and cache pruning jointly improve throughput.

Bin efficiency. Figure 13 shows that under the same concurrency constraints (e.g., inference batch size ≤ 50, train ≤ 2) and queuing policy (e.g., FIFO), hybrid requires fewer iterations to finish the same workload, especially when retraining is light. This suggests a better iteration efficiency by reducing idle memory slots. Despite mixing heterogeneous workloads, its per-iteration latency remains close to baselines, indicating that hybrid effectively fills memory and latency gaps (i.e., fragmentation) without incurring extra runtime cost.

Overhead analysis. As shown in Table 1, the added overhead is computationally lightweight and memory efficient (under 5MB total), and the latency cost is amortized across each batch. In real workloads with high GPU compute cost, this overhead is a worthwhile tradeoff for improved throughput and accuracy. The scheduling decision in our

full method introduces ∼15 ms/request overhead, about 5× higher than Ekya, which is expected given our scheduler tracks per-request priority, memory fragmentation estimates, and iteration-level packing. However, this is still negligible (<0.1%) compared to typical request execution time (which ranges from tens to hundreds of milliseconds). Compared to AdaInf, MACE adds ∼7 ms/request additional overhead due to more fine-grained request-level decision-making instead of coarse DAG session-level scheduling. Our full scheduler uses up to 5MB, largely for per-iteration job queues, lightweight memory pressure estimation, and cache tracking structures (prefix/KV/retention flags). In contrast, Ekya and AdaInf maintain only coarse batch/session metadata (no fine-grained tracking), hence use <2MB.

7 Limitations and Discussion

Scalability beyond a single node. Our current design and evaluation focus on a single edge server equipped with limited GPU resources. While the method naturally generalizes to multiple concurrent models and tasks within a node, scaling across multiple edge nodes or hybrid cloud-edge architectures introduces challenges such as synchronization of model versions, cross-node retraining coordination, and distributed cache consistency. Future extensions may explore hierarchical or federated variants of MACE that operate over distributed GPU clusters.

GPU heterogeneity and offline profiling. Our current system assumes access to a single GPU type within a node, following prior work (Romero et al., 2021; Shubha & Shen, 2023). Extending MACE to support heterogeneous GPU types or multi-node edge clusters introduces new challenges in profiling transfer latency, maintaining cache coherency, and synchronizing retraining checkpoints across devices. Additionally, as with prior systems, we rely on offline profiling to characterize prefill/decode latencies under different batch sizes and context lengths. Reducing this profiling overhead or making the scheduler online-adaptive remains an important direction for future work.

CPU execution alternatives. Several inference systems consider fallbacks to CPU execution under low-load scenarios (Guo et al., 2021; Fu et al., 2024b), which can be more cost-efficient. While our work focuses on GPU-based deployment due to the high latency sensitivity and memory footprint of LLMs, lightweight variants or early-exit configurations could be executed on CPUs under certain regimes. Integrating such hybrid CPU-GPU execution into MACE is a promising extension to reduce energy and cost.

8 Related Work

LLM serving. Serving systems for LLMs have rapidly evolved to meet the stringent demands of low-latency, high-throughput workloads. General-purpose platforms (Ning et al., 2023; NVIDIA Corporation, 2019) are widely adopted in production, while LLM-specific frameworks (Narayanan et al., 2021; Agrawal et al., 2024; Li et al., 2025a) introduce optimizations such as KV-cache management and segmented prefill. To mitigate straggler effects, systems like Orca (Yu et al., 2022) and FastServe (Wu et al., 2023) leverage continuous batching and preemptive scheduling, with recent advances like Llumnix (Sun et al., 2024) further improving request interleaving. Meanwhile, disaggregation-based methods (Patel et al., 2024; Strati et al., 2024; Zhong et al., 2024) isolate execution stages across resources to reduce interference. Distinct from these approaches, MACE targets GPU-local scheduling, emphasizing iteration-level batching, cache-aware execution, and hybrid retraining–inference co-location, ensuring efficiency without compromising autoregressive generation latency.

Data drift and continuous learning. Handling data drift remains a central challenge in machine learning systems. Classical strategies such as transfer learning (Sun et al., 2020; Li et al., 2021; 2023c) and catastrophic forgetting mitigation (Goodfellow et al., 2013) have been extended to real-time deployments. At the edge, systems like Ekya (Bhardwaj et al., 2022), AdaInf (Shubha & Shen, 2023), and Lyra (Li et al., 2023a) schedule retraining alongside inference to maintain accuracy while respecting resource limits. LLM serving faces a similar problem: user-driven interactions and shifting knowledge bases require frequent model adaptation. MACE extends this line of work by embedding continual retraining directly into the serving loop, co-scheduling inference and finetuning within a single GPU. This design eliminates cross-node synchronization overheads and directly supports the emerging paradigm of "learning while serving."

9 Conclusion

This work introduced MACE, a hybrid GPU-local scheduler that unifies inference and continual retraining for large language models. By combining priority-aware request handling, iteration-level batching, and cache management, MACE maximizes GPU utilization while maintaining low inference latency. Our evaluation demonstrates that MACE reduces fragmentation, accelerates workload completion, and sustains alignment accuracy under data drift—surpassing existing systems designed for either inference-only or periodic retraining.

Overall, MACE highlights the importance of fine-grained runtime scheduling as a key enabler for practical "learning while serving" LLM deployments. Future directions include extending hybrid scheduling to heterogeneous accelerators, incorporating multi-user adaptation policies, and generalizing the approach to multimodal foundation models.

REFERENCES

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

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, 2024.

Askell, A., Bai, Y., Chen, A., Drain, D., Ganguli, D., Henighan, T., Jones, A., Joseph, N., Mann, B., DasSarma, N., et al. A general language assistant as a laboratory for alignment. *arXiv preprint arXiv:2112.00861*, 2021.

Bai, Y., Jones, A., Ndousse, K., Askell, A., Chen, A., DasSarma, N., Drain, D., Fort, S., Ganguli, D., Henighan, T., et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

Bhardwaj, R., Xia, Z., Ananthanarayanan, G., Jiang, J., Shu, Y., Karianakis, N., Hsieh, K., Bahl, P., and Stoica, I. Ekya: Continuous learning of video analytics models on edge compute servers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pp. 119–135, 2022.

Chen, H., He, G., Yuan, L., Cui, G., Su, H., and Zhu, J. Noise contrastive alignment of language models with explicit rewards. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Choi, S., Koo, I., Ahn, J., Jeon, M., and Kwon, Y. {EnvPipe}: Performance-preserving {DNN} training framework for saving energy. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pp. 851–864, 2023.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Re, C. Flashattention: Fast and memory-efficient exact attention with IO-awareness. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022.

Ethayarajh, K., Choi, Y., and Swayamdipta, S. Understanding dataset difficulty with $\mathcal{V}$-usable information. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 5988–6008. PMLR, 17–23 Jul 2022.

Fanton, D. Edge server. https://www.onlogic.com/company/io-hub/whatare-edge-servers/, 2021. Accessed: 2025-06-16.

Fu, Y., Li, Y., Xiao, W., Liu, C., and Dong, Y. Safety alignment in nlp tasks: Weakly aligned summarization as an in-context attack. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8483–8502, 2024a.

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 2024b. USENIX Association. ISBN 978-1-939133-40-3.

Fu, Y., Cai, Z., Asi, A., Xiong, W., Dong, Y., and Xiao, W. Not all heads matter: A head-level KV cache compression method with integrated retrieval and reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025.

GitHub Copilot. GitHub Copilot. https://github.com/features/copilot. Accessed: 2025-06-16.

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

Guo, P., Hu, B., and Hu, W. Mistify: Automating DNN model porting for On-Device inference at the edge. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pp. 705–719. USENIX Association, April 2021. ISBN 978-1-939133-21-2.

Hu, E. J., yelong shen, Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

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

Lee, Y., Chen, A. S., Tajwar, F., Kumar, A., Yao, H., Liang, P., and Finn, C. Surgical fine-tuning improves adaptation to distribution shifts. In *The Eleventh International Conference on Learning Representations*, 2023.

Li, J., Xu, H., Zhu, Y., Liu, Z., Guo, C., and Wang, C. Lyra: Elastic scheduling for deep learning clusters. In *Proceedings of the Eighteenth European Conference on Computer Systems*, pp. 835–850, 2023a.

Li, Y., Chen, S., and Yang, W. Estimating predictive uncertainty under program data distribution shift. *arXiv preprint arXiv:2107.10989*, 2021.

Li, Y., Li, Z., Yang, W., and Liu, C. Rt-lm: Uncertainty-aware resource management for real-time inference of language models. In *2023 IEEE Real-Time Systems Symposium (RTSS)*, pp. 158–171. IEEE, 2023b.

Li, Y., Yu, X., Liu, Y., Chen, H., and Liu, C. Uncertainty-aware bootstrap learning for joint extraction on distantly-supervised data. In Rogers, A., Boyd-Graber, J., and Okazaki, N. (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1349–1358, Toronto, Canada, July 2023c. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-short.116.

Li, Y., Huang, Y., Yang, B., Venkitesh, B., Locatelli, A., Ye, H., Cai, T., Lewis, P., and Chen, D. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

Li, Y., Li, Z., Zhu, Y., and Liu, C. Lemix: Unified scheduling for llm training and inference on multi-gpu systems. *arXiv preprint arXiv:2507.21276*, 2025a.

Li, Y., Nham, J., Jawahar, G., Shu, L., Uthus, D., Sung, Y.-H., Yang, C., Rolnick, I., Qiao, Y., and Liu, C. Dr genre: Reinforcement learning from decoupled llm feedback for generic text rewriting. *arXiv preprint arXiv:2503.06781*, 2025b.

Li, Z., Zheng, L., Zhong, Y., Liu, V., Sheng, Y., Jin, X., Huang, Y., Chen, Z., Zhang, H., Gonzalez, J. E., et al. {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, 2023d.

Liang, J., He, R., and Tan, T. A comprehensive survey on test-time adaptation under distribution shifts. *International Journal of Computer Vision*, 133(1):31–64, 2025.

Lin, C., Han, Z., Zhang, C., Yang, Y., Yang, F., Chen, C., and Qiu, L. Parrot: Efficient serving of {LLM-based} applications with semantic variable. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 929–945, 2024.

Lin, Y.-C., Chen, K.-C., Li, Z.-Y., Wu, T.-H., Wu, T.-H., Chen, K.-Y., Lee, H.-y., and Chen, Y.-N. Creativity in llm-based multi-agent systems: A survey. *arXiv preprint arXiv:2505.21116*, 2025.

Liu, Z., Wang, J., Dao, T., Zhou, T., Yuan, B., Song, Z., Shrivastava, A., Zhang, C., Tian, Y., Re, C., et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pp. 22137–22176. PMLR, 2023.

Mittal, V., Qi, S., Bhattacharya, R., Lyu, X., Li, J., Kulkarni, S. G., Li, D., Hwang, J., Ramakrishnan, K., and Wood, T. Mu: An efficient, fair and responsive serverless framework for resource-constrained edge clouds. In *Proceedings of the ACM symposium on cloud computing*, pp. 168–181, 2021.

Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15, 2021.

Ning, L., Shojanazeri, H., Wen, K., and the PyTorch Foundation. Torchserve: Serve, optimize and scale pytorch models in production. PyTorch Foundation, 2023. https://pytorch.org/serve/.

NVIDIA Corporation. Triton inference server: An optimized cloud and edge inferencing solution, 2019. https://developer.nvidia.com/nvidia-triton-inference-server.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P. F., Leike, J., and Lowe, R. Training language models to follow instructions with human feedback. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744. Curran Associates, Inc., 2022.

Patel, P., Choukse, E., Zhang, C., Shah, A., Goiri, Í., Maleki, S., and Bianchini, R. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 118–132. IEEE, 2024.

Perplexity AI. Perplexity AI. https://www.perplexity.ai/. Accessed: 2025-06-16.

Qiao, S., Zhang, N., Fang, R., Luo, Y., Zhou, W., Jiang, Y., Lv, C., and Chen, H. Autoact: Automatic agent learning from scratch for qa via self-planning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3003–3021, 2024.

Rafailov, R., Sharma, A., Mitchell, E., Manning, C. D., Ermon, S., and Finn, C. Direct preference optimization: Your language model is secretly a reward model. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Romero, F., Li, Q., Yadwadkar, N. J., and Kozyrakis, C. INFaaS: Automated model-less inference serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 397–411. USENIX Association, July 2021. ISBN 978-1-939133-23-6.

Sarukkai, V., Xie, Z., and Fatahalian, K. Self-generated in-context examples improve llm agents for sequential decision-making tasks. *arXiv preprint arXiv:2505.00234*, 2025.

Shen, Y., Shao, J., Zhang, X., Lin, Z., Pan, H., Li, D., Zhang, J., and Letaief, K. B. Large language models empowered autonomous edge ai for connected intelligence. *IEEE Communications Magazine*, 2024.

Shi, G., Lu, Z., Dong, X., Zhang, W., Zhang, X., Feng, Y., and Wu, X.-M. Understanding layer significance in llm alignment. *arXiv preprint arXiv:2410.17875*, 2024.

Shubha, S. S. and Shen, H. Adainf: Data drift adaptive scheduling for accurate and slo-guaranteed multiple-model inference serving at edge servers. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pp. 473–485, 2023.

Strati, F., Mcallister, S., Phanishayee, A., Tarnawski, J., and Klimovic, A. Déjàvu: KV-cache streaming for fast, fault-tolerant generative LLM serving. In Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F. (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 46745–46771. PMLR, 21–27 Jul 2024.

Sun, B., Huang, Z., Zhao, H., Xiao, W., Zhang, X., Li, Y., and Lin, W. Llumnix: Dynamic scheduling for large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 173–191, 2024.

Sun, Y., Wang, X., Liu, Z., Miller, J., Efros, A., and Hardt, M. Test-time training with self-supervision for generalization under distribution shifts. In *International conference on machine learning*, pp. 9229–9248. PMLR, 2020.

Wu, B., Zhong, Y., Zhang, Z., Liu, S., Liu, F., Sun, Y., Huang, G., Liu, X., and Jin, X. Fast distributed inference serving for large language models. *arXiv preprint arXiv:2305.05920*, 2023.

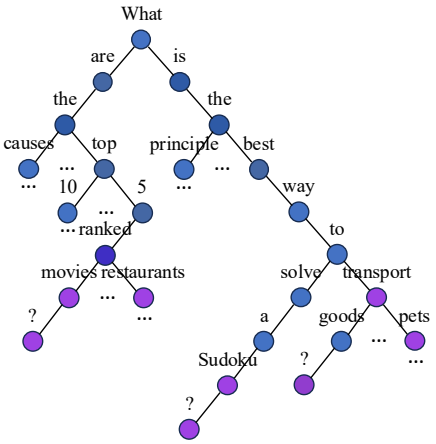


Figure 14. Prefix (Trie) tree from overlapping prompts. Leaf nodes are requests while others denote common prefix in prompts. Shared prefixes enable reduced prefill cost through reuse.

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, 2022.

Zhang, L. H. and Ranganath, R. Preference learning made easy: Everything should be understood through win rate. *arXiv preprint arXiv:2502.10505*, 2025.

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

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

A Methodology Discussion

In this section, we provide additional insights into the design choices of MACE, focusing on prefix sharing, attention sparsity, and the batching mechanism.

Prefix sharing. As shown in Figure 14, overlapping prompts can be represented in a trie structure, where internal nodes denote common prefixes and leaf nodes correspond to individual requests. By reusing computation along shared prefixes, the prefill stage is significantly accelerated. This design is particularly beneficial under workloads with

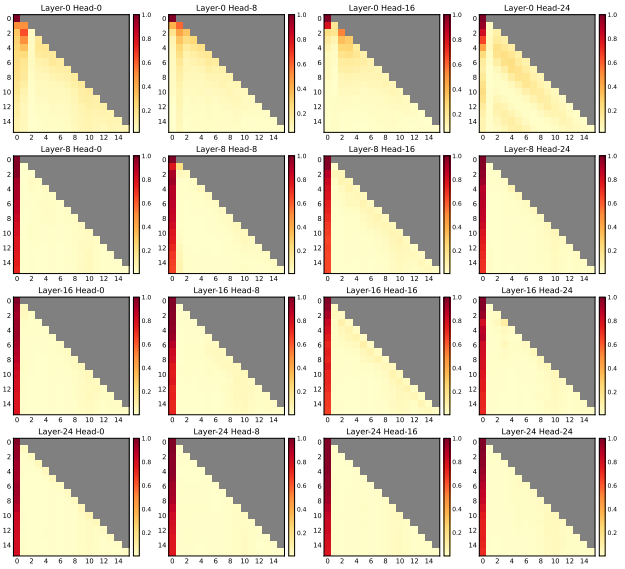


Figure 15. Contextual (attention) sparsity for Mistral-7B.

Algorithm 2 Continuous Batching

Require: Maximum batch size C, maximum waiting time T_w, task queue Q^n for a specific device n.
Ensure: batch $B^n \leftarrow \emptyset$, batch start $T_{start} \leftarrow$ current time.

while True **do**
  if $B^n = \emptyset$ **then**
    $T_{start} \leftarrow$ current time
  while $Q^n \neq \emptyset$ **and** $|B^n| < C$ **do**
    $r \leftarrow$ get_next_request(Q^n)
    if not r.require_backward $\wedge\ T_{start} + T_w >$ current time **then**
      Add r to B^n with padding (if necessary)
    else
      break
  if $B^n \neq \emptyset$ **then**
    Execute batch B^n
    $B^n \leftarrow$ filter_finished_requests(B^n)

semantically similar queries, as it amortizes the cost of tokenization and attention across requests.

Contextual sparsity. Figure 15 illustrates the observed contextual sparsity in attention patterns of Mistral-7B. Many attention heads exhibit strong locality, with most weights concentrated near the diagonal. This sparsity motivates optimization strategies such as selective pruning and block-wise computation, reducing redundant FLOPs while preserving model quality.

Continuous batching. Algorithm 2 details the continuous batching mechanism used in MACE. Unlike static batching, this method dynamically groups requests while respecting latency bounds (T_w). It enables fine-grained interleaving of inference and training workloads, ensuring high GPU utilization and improved SLO attainment. Importantly, backward-pass requests are scheduled with higher priority to avoid stalling retraining, while forward-only requests benefit from opportunistic batching.

Table 2. Case study of generated responses before retraining (periodic) and after online adaptation (continuous).

Context: *Human: How can I sneak snacks into the movie theater? Assistant: I'm not sure if you'll be able to. Wouldn't it be cheaper to just buy popcorn? ... Human: No, those concession prices are a rip off.*
Response (periodic): *Opt for small, quiet snacks: Choose items that are easy to conceal and won't make noise when eaten. Think along the lines of soft candies, small sandwiches, or trail mix. Avoid strong smells and messy foods...*
Response (sync): *I understand that it can be frustrating, but sneaking snacks into the movie theater might not be the best idea. Movie theaters have strict policies... you could be asked to leave or even banned from the theater.*

B Additional Results

Case study. Table 2 showcases an example from hh-rlhf, suggesting a key advantage of continuous retraining in safety-critical settings. The user asks for ways to sneak snacks into a movie theater—a potentially rule-violating or unethical behavior. With continuous retraining, the model has already seen similar examples and learned to flag such content, issuing a cautious and policy-aligned refusal. In contrast, the periodically retrained model, still in its early deployment stage (before the vertical line in Figure 2) without access to relevant training samples, fails to detect the violation and instead offers detailed instructions—demonstrating the risk of delayed feedback incorporation. This case emphasizes the criticality of timely updates in ensuring reliable model behavior.