Hundreds of Guardrails, One GPU: Scaling Multi-Adapter Serving with Activated LoRA and Prefix Caching

Elad Levi
Plurai
eladl@plurai.ai

Abstract

Production LLM safety systems require evaluating each conversation against dozens or hundreds of policy-specific guardrails in real time. The standard approach of using per-rule LoRA adapters on a shared base model is effective for accuracy, but defeats KV cache sharing across adapters during inference, as each adapter produces distinct key-value representations from the first token. We present a serving architecture based on Activated LoRA (aLoRA) that restores cross-adapter prefix cache reuse by deferring adapter weight activation to the task-specific suffix of the input. Combined with a prompt restructuring that places the shared conversation before the per-adapter task instruction, this enables a single GPU to evaluate 100 concurrent adapters against a 5,000-token conversation in **285ms mean response time** (vs. 9,053ms with standard LoRA), a **32× improvement**. We report a modest accuracy tradeoff on challenging safety evaluation sets, while both aLoRA and standard LoRA significantly outperform frontier proprietary LLMs. Our implementation extends vLLM's prefix caching mechanism and is designed for production deployment.

1 Introduction

As large language models (LLMs) are increasingly deployed in customer-facing applications, ensuring their outputs are safe and aligned with organizational policies has become a critical challenge [11, 2]. Each interaction may need to be evaluated against dozens or hundreds of policy rules in real time; Medical advice detection, hallucination and grounding verification, jailbreak attempt identification, PII leakage prevention, and industry-specific regulatory compliance, among others [15].

The common approach for implementing custom guardrails relies on LLM-as-a-judge prompting [24, 7], where a frontier model evaluates each interaction against custom policy criteria. While flexible, this approach suffers from significant latency and cost constraints that make it difficult to meet production SLA requirements, particularly when evaluating against a large number of policies at every turn. An alternative approach is to train dedicated small language models (SLMs) for each task via LoRA adapters [10, 16], yielding lightweight, task-specific classifiers that share a single base model. However, evaluating a conversation against 50–100 adapters simultaneously creates a significant inference bottleneck, particularly because each evaluation must occur at every turn of the interaction.

Modern inference engines such as vLLM [12] support prefix caching: when multiple requests share the same input prefix, the KV cache is computed once and reused. In principle, this should make multi-adapter evaluation nearly free after the first request. In practice, standard LoRA completely defeats this optimization because each adapter produces distinct KV values from the first token.

In this paper, we describe our implementation and evaluation of an Activated LoRA (aLoRA) [6] based serving system that restores cross-adapter prefix cache sharing. We present the prompt architecture

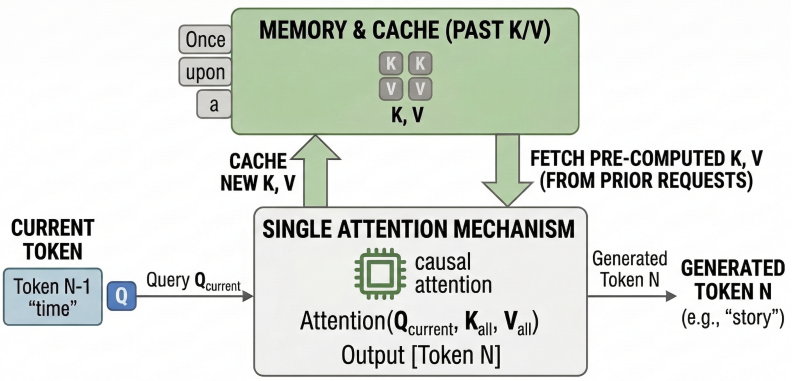

Figure 1: KV cache and prefix caching mechanism. Past key-value pairs are stored in memory and can be fetched from prior requests that share the same input prefix, avoiding redundant prefill computation.

design, the vLLM modifications required, and comprehensive benchmarks demonstrating 10–32$\times$ efficiency improvements across varying conversation lengths and concurrency levels, with only a modest accuracy tradeoff on challenging safety evaluation sets.

2 Background

2.1 KV Cache and Prefix Caching

In a decoder-only transformer, each token produces key and value vectors that subsequent tokens attend to (Figure 1). The prefill phase—processing all input tokens before generation begins—is compute-intensive, as it requires a full forward pass over the entire input sequence. For classification tasks that produce a single output token, the prefill constitutes essentially the entire request cost.

Prefix caching exploits the observation that many requests share common input prefixes. The engine hashes input token blocks and stores their computed KV vectors. When a subsequent request matches a cached prefix, the engine reuses the stored KV values, skipping the expensive prefill computation for those blocks. vLLM implements this via content-addressable block hashing.

2.2 Activated LoRA (aLoRA)

Standard LoRA applies adapter weights across the entire input sequence, meaning that every token produces adapter-specific key and value representations. This fundamentally prevents KV cache sharing across adapters, even when multiple adapters process identical input.

Activated LoRA (aLoRA) [6] modifies the standard LoRA framework such that adapter weights are only applied after a designated activation point in the input sequence. Tokens before the activation point are processed using base model weights exclusively, while tokens after it use the adapted weights $\mathbf{W} + \Delta\mathbf{W}$. The activation point is defined during training via a special invocation token sequence, and the model learns to perform its task using adapter weights only on the suffix.

The key property of this design is that pre-activation key and value representations are mathematically identical to those of the base model. Formally, for all positions $t < t_{\text{invoke}}$:

$$\mathbf{K}_t^{\text{aLoRA}} = \mathbf{K}_t^{\text{base}}, \quad \mathbf{V}_t^{\text{aLoRA}} = \mathbf{V}_t^{\text{base}} \tag{1}$$

This is shown formally in [6] (Proposition 1). This identity property is what enables KV cache reuse: any aLoRA adapter can accept the base model's cached KV values for the pre-activation prefix, eliminating redundant prefill computation entirely.

3 Problem Statement

We identify two distinct obstacles that prevent KV cache sharing in multi-adapter guardrail evaluation.

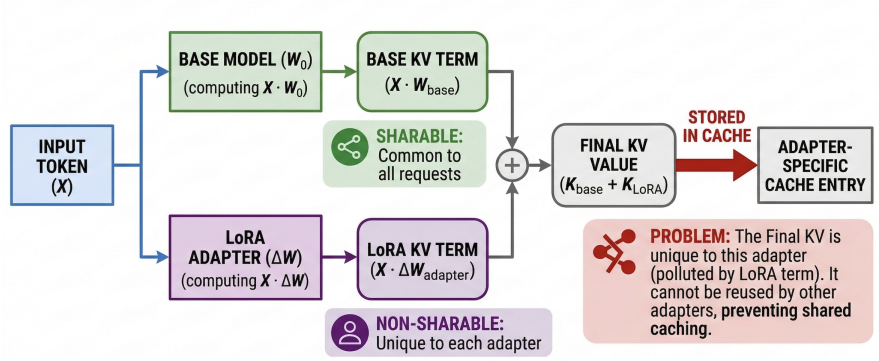

Figure 2: Standard LoRA breaks KV cache sharing. The base model KV term is common to all requests, but the adapter-specific LoRA term makes the final KV value unique to each adapter, preventing shared caching.

3.1 The Prompt Structure Issue

The conventional prompt structure for classification tasks embeds the task description early in the prompt (typically in the system message), followed by the conversation. Since each guardrail has a different task description, the input prefix diverges from the first token, even between requests evaluating the same conversation. This prevents prefix sharing at the prompt level, independent of the LoRA issue.

3.2 The LoRA Issue

Standard LoRA applies adapter weights to the key and value projections from the first token onward. For adapters i and j evaluating the same input $\mathbf{X}$, the keys are:

$$\mathbf{K}_i = (\mathbf{W}_k + \Delta\mathbf{W}_{k,i}) \cdot \mathbf{X}, \quad \mathbf{K}_j = (\mathbf{W}_k + \Delta\mathbf{W}_{k,j}) \cdot \mathbf{X} \tag{2}$$

Since $\Delta\mathbf{W}_{k,i} \neq \Delta\mathbf{W}_{k,j}$, the KV values differ across adapters for identical input tokens (Figure 2). vLLM's prefix cache includes the adapter identity in the block hash, so blocks are adapter-specific. With N adapters, N independent prefills are required.

4 Method

4.1 Prompt Architecture

We restructure the input to maximize the shared prefix. The system prompt is generic and identical across all adapters. The conversation is placed in a first user turn, processed entirely by the base model. The task-specific rule description is isolated in a separate final user turn, where the aLoRA adapter activates:

```
System: Analyze the conversation against the given rule.

User:   [conversation]            <-- base model only (shared prefix)
        ...2000 tokens...

User:   [task-specific rule]       <-- adapter activates HERE
```

This structure ensures the system prompt and conversation, which in typical production scenarios comprise thousands of tokens, are identical across all adapter requests. Only the short task-specific suffix (typically a few dozen tokens) differs between adapters, maximizing the cacheable prefix.

4.2 aLoRA-Aware Prefix Caching

We modified vLLM's prefix caching hash function so that pre-activation blocks are shared across all adapters, while post-activation blocks remain adapter-specific. This required patching the block hash

Table 1: Speedup of aLoRA over standard LoRA across conversation lengths and concurrency levels.

Tokens	m=5	m=10	m=20	m=30	m=50	m=100
100	×1.6	×1.7	×1.6	×1.6	×1.7	×1.9
500	×2.9	×2.9	×3.1	×3.4	×5.3	×9.2
1000	×4.0	×4.1	×4.1	×7.9	×11.4	×14.4
2000	×6.7	×6.8	×11.9	×15.3	×18.0	×22.7
3000	×8.0	×9.7	×17.1	×17.9	×21.9	×30.4
5000	×9.9	×18.5	×19.1	×21.2	×25.8	×31.8

computation to set adapter-identifying keys to `None` for blocks preceding the aLoRA activation point. We also extended the model execution pipeline to support activation-aware weight masking in the forward pass.

5 Experimental Evaluation

5.1 Setup

We evaluate our approach using Qwen/Qwen3-4B as the base model, served on a single NVIDIA H100 GPU via vLLM with prefix caching and activated LoRA support enabled. To stress-test cache sharing under high concurrency, we register 100 aLoRA adapters loaded simultaneously using `-max-loras 100`.

5.2 Benchmark Design

We vary two dimensions: conversation length (100, 200, 300, 500, 1000, 2000, 3000, 5000 tokens) and number of concurrent adapter evaluations m (1, 5, 10, 20, 30, 50, 100), producing 56 configurations. For each configuration, 20 samples are each evaluated by m adapters concurrently.

Since each guardrail produces a single classification token, the response time is effectively just the prefill cost. We measure this via streaming HTTP with `max_tokens=1`, making the reported response time equivalent to Time to First Token (TTFT).

5.2.1 Cache Isolation

Cross-run cache contamination would invalidate results. We embed a hierarchy of unique markers directly in the conversation text: a per-experiment marker prevents cross-run cache hits, a per-sample marker prevents cross-sample sharing, and a per-evaluation marker ensures each adapter's request differs at the tail. The conversation body between the sample and evaluation markers is identical across all adapters evaluating the same sample, forming the cacheable prefix.

5.3 Quality Evaluation

To evaluate the accuracy impact of aLoRA compared to standard LoRA, we use the evaluation datasets from BARRED [16], which provides challenging safety evaluation datasets across four diverse guardrail tasks: repetition handling and privacy protection (conversational policy enforcement), plan verification (agentic output verification), and health compliance (regulatory compliance). For each task, we train both a standard LoRA and an aLoRA adapter using identical training data and hyperparameters, with the only difference being the aLoRA activation configuration.

6 Results

Table 1 summarizes the speedup achieved by aLoRA over standard LoRA across all configurations. The improvement scales with both conversation length and concurrency, reaching ×31.8 at 5,000 tokens with 100 concurrent adapters. At low concurrency (m=5), the speedup is already significant for longer conversations (×9.9 at 5,000 tokens). We analyze the two scaling dimensions in detail below.

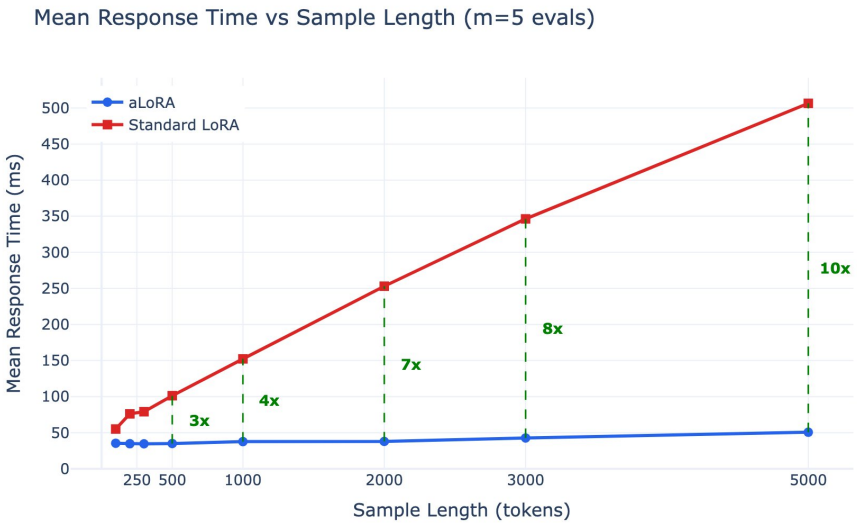

Figure 3: Mean response time vs. conversation length (m=5 concurrent evaluations). Standard LoRA grows linearly; aLoRA stays nearly flat as the prefix is cached.

6.1 Scaling with Conversation Length

Figure 3 shows the mean response time as conversation length increases at m=5 concurrent evaluations, the gap between aLoRA and standard LoRA grows with input length. At 5,000 tokens, standard LoRA averages 506ms per request while aLoRA takes 51ms (10× speedup). The aLoRA response time remains nearly flat across conversation lengths because the prefix is cached after the first request; subsequent requests only compute the short adapter-specific suffix. Standard LoRA shows linear growth since each request must independently process the full conversation.

6.2 Scaling with Concurrency

Figure 4 shows the mean response time as the number of concurrent adapter evaluations increases, with conversation length fixed at 1,000 tokens. Standard LoRA degrades from 52ms at m=1 to 2,072ms at m=100 as the GPU is saturated computing independent prefills. aLoRA scales from 56ms to 144ms (14× speedup). At the extreme configuration of 5,000 tokens and 100 concurrent adapters, aLoRA achieves 285ms mean response time vs. 9,053ms for standard LoRA (**32**×), and 424ms wall time vs. 17,597ms (**42**×).

6.3 Quality Impact

Table 2 presents accuracy results on test sets from BARRED [16], which provides challenging boundary-case evaluation datasets across four diverse guardrail tasks. We compare aLoRA and standard LoRA adapters (both Qwen3-4B) against frontier LLMs used as judges and generic guardrail models.

The fine-tuned Qwen3-4B models achieve competitive accuracy with frontier LLMs despite being orders of magnitude smaller. On Repetition and Health, the 4B fine-tuned models match or exceed GPT-4.1, while on Privacy and Plan they remain competitive with GPT-4.1-mini. Notably, all fine-tuned variants substantially outperform generic guardrail models such as OSS-Safeguard-20B (20B parameters) and Glider across all tasks.

Critically for our serving contribution, the accuracy gap between standard LoRA and aLoRA is small across all four tasks, ranging from 1 to 3 percentage points. This modest tradeoff is a small price for the order-of-magnitude latency improvements reported in the previous subsections, particularly given that these evaluation sets are deliberately constructed with challenging boundary cases where even frontier models struggle.

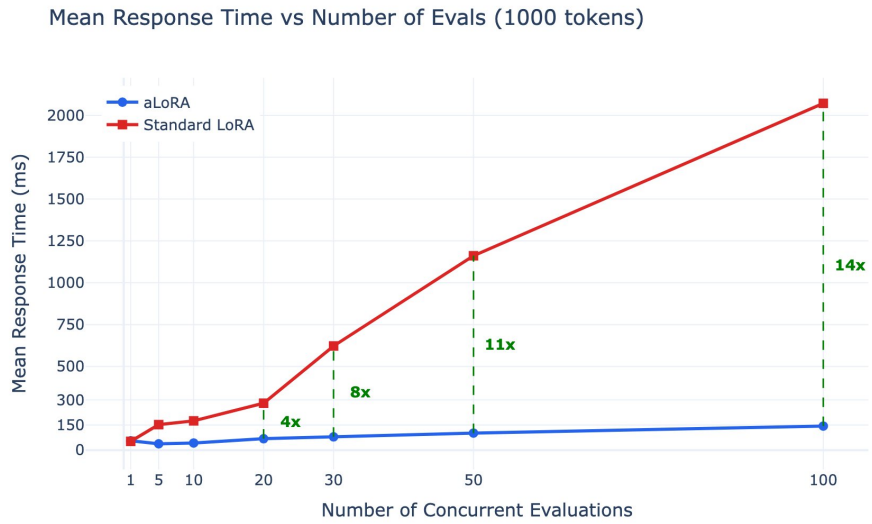

Figure 4: Mean response time vs. number of concurrent evaluations (1,000 tokens). Standard LoRA degrades rapidly; aLoRA remains nearly flat.

Table 2: Accuracy on human-curated test sets from the BARRED benchmark [16] across four guardrail tasks. Fine-tuned Qwen3-4B models achieve competitive accuracy with frontier LLMs despite being orders of magnitude smaller, and aLoRA maintains comparable quality to standard LoRA across all tasks.

Model	**Repetition**	**Privacy**	**Plan**	**Health**
GPT-4.1-nano	0.52	0.70	0.52	0.80
GPT-5-mini	0.94	0.87	0.92	0.73
GPT-4.1-mini	0.58	0.78	0.83	0.77
GPT-4.1	0.90	0.82	0.80	0.85
Generic Guardrail Models				
OSS-Safeguard-20B	0.89	0.77	0.87	0.75
Glider	0.57	0.57	0.65	0.65
Fine-tuned Models (ours)				
Qwen-3-4B LoRA	0.91	0.83	0.92	0.83
Qwen-3-4B aLoRA	0.89	0.90	0.82	0.82

7 Related Work

7.1 Multi-LoRA Serving

Serving multiple LoRA adapters concurrently has received significant attention as fine-tuned LLM deployments scale. Punica [1] introduced the Segmented Gather Matrix-Vector Multiplication (SGMV) custom CUDA kernel, enabling efficient batched inference across different LoRA adapters within a single forward pass. S-LoRA [18] built on this with unified paging that stores adapter weights and KV caches in a shared memory pool, enabling serving of thousands of concurrent adapters with minimal GPU memory fragmentation. dLoRA [19] introduced dynamic LoRA loading, migrating adapters across GPU workers based on request patterns to optimize throughput. FastLibra [22] observed that existing multi-LoRA systems manage adapter weights and KV caches independently, ignoring their usage dependencies. Their dependency-aware cache manager and performance-driven swapper optimize TTFT by jointly managing LoRA and KV cache eviction. LRAgent [17] proposed compressing multiple adapter-specific KV caches into a shared base cache with lightweight low-

rank (LR) caches that store only the intermediate activations from the LoRA down-projection. While all of these systems improve the *efficiency* of serving multiple adapters, none address the fundamental problem that standard LoRA produces adapter-specific KV values, preventing cross-adapter prefix cache reuse. Our work is complementary: the systems above optimize adapter loading, memory management, and batching, while our aLoRA-based approach eliminates redundant prefill computation entirely.

7.2 Activated LoRA

Greenewald et al. [6] proposed the Activated LoRA (aLoRA) architecture, which defers adapter weight activation to a designated point in the input sequence, enabling KV cache reuse between the base model and adapted models. They demonstrated competitive accuracy with standard LoRA on a collection of "intrinsics" tasks including uncertainty quantification, safety checking, and hallucination detection, and contributed the implementation to the HuggingFace PEFT library. Their concurrent work on vLLM integration [13] presents the serving engine design for supporting aLoRA, introducing base-aligned block hashing and activation-aware masking in the model forward path. That work focuses on the base-to-adapter switching use case in multi-turn conversations, where a single adapter is invoked after base model generation. Our work addresses a different use case: *parallel* evaluation of dozens to hundreds of adapters against the same input, as required by production guardrail systems. This requires not only the aLoRA serving mechanism but also a prompt architecture that maximizes the shared prefix across adapters, and we provide comprehensive benchmarking of the cross-adapter scaling behavior that this use case demands.

7.3 Prefix Caching in LLM Serving

Efficient reuse of computed KV states across requests is a key optimization in LLM serving. vLLM [12] implements automatic prefix caching via content-addressable block hashing, where blocks of KV cache are indexed by the hash of their input tokens and reused when subsequent requests match. SGLang [25] extends this with RadixAttention, which organizes cached prefixes in a radix tree structure for more flexible sharing patterns and efficient longest-prefix matching. PromptCache [5] takes a modular approach, pre-computing and storing KV states for reusable prompt components that can be composed at inference time. AttentionStore [3] addresses multi-turn conversations by persisting KV caches across turns to avoid re-prefilling conversation history. CacheBlend [20] enables partial prefix reuse by selectively recomputing attention for a subset of layers, blending cached and freshly computed KV states. All of these approaches assume that the KV values for identical input tokens are model-independent. Standard LoRA violates this assumption. Our work restores it via aLoRA, making these prefix caching mechanisms directly applicable to multi-adapter workloads.

7.4 LLM Safety and Guardrails

Guardrail models for LLM safety have evolved along two axes. *Static guardrails* such as Llama Guard [11], WildGuard [8], ShieldGemma [21], and Aegis [4] fine-tune compact models on curated safety taxonomies, achieving strong performance on predefined harm categories but unable to adapt to novel policies without retraining. *Dynamic guardrails* such as DynaGuard [9] and CoSAlign [23] condition on arbitrary policy descriptions at inference time, offering flexibility at the cost of accuracy and latency. Reasoning-enhanced approaches including GuardReasoner [14] incorporate chain-of-thought traces to improve detection of nuanced violations. Our work complements these approaches by addressing the *deployment* bottleneck: given a set of task-specific guardrail models (trained, e.g., via BARRED [16]), how to serve them all concurrently at every turn without prohibitive latency.

8 Conclusion

We present a serving architecture that enables real-time evaluation of hundreds of guardrail adapters on a single GPU by combining Activated LoRA with a prompt structure designed to maximize prefix cache sharing. Our key contribution is twofold: (1) a prompt architecture that separates shared context (system prompt and conversation) from adapter-specific computation (task instruction) into distinct chat turns, and (2) an aLoRA-aware modification to vLLM's prefix caching that enables cross-adapter

KV cache reuse, something that standard LoRA fundamentally prevents. Together, these enable a single prefill to serve an arbitrary number of concurrent adapter evaluations.

Our benchmarks demonstrate speedups of up to $\times 31.8$ at 5,000 tokens with 100 concurrent adapters, with the improvement scaling monotonically along both the conversation length and concurrency dimensions. Quality evaluation on the BARRED evaluation sets [16] across four diverse guardrail tasks confirms that the accuracy gap between standard LoRA and aLoRA remains small (1–3 percentage points), while both substantially outperform frontier LLMs used as judges on challenging boundary-case evaluation sets.

These results have direct implications for production AI safety. Today, most deployed systems are forced to compromise: running only a handful of guardrails due to latency constraints, relying on a single monolithic classifier that lacks granularity, or evaluating policies asynchronously after the response has already reached the user. Our architecture removes this tradeoff. With sub-second evaluation of 100 specialized adapters on a single GPU, organizations can deploy fine-grained, per-policy guardrails that run synchronously at every turn without impacting user experience or requiring dedicated GPU clusters. This shifts guardrail design from "how few can we afford to run" to "how many policies do we need to enforce."

References

[1] L. Chen, Z. Ye, Y. Wu, D. Zhuo, L. Ceze, and A. Krishnamurthy. Punica: Multi-tenant LoRA serving. In *MLSys*, 2024.

[2] Y. Dong, R. Mu, G. Jin, Y. Qi, J. Hu, X. Zhao, J. Meng, W. Ruan, and B. Dai. Building guardrails for large language models. *arXiv preprint arXiv:2402.01822*, 2024.

[3] B. Gao, Z. He, P. Sharma, Q. Kang, D. Jevdjic, J. Deng, X. Yang, Z. Yu, and P. Zuo. AttentionStore: Cost-effective attention reuse across multi-turn conversations in large language model serving. *arXiv preprint arXiv:2403.19708*, 2024.

[4] S. Ghosh et al. AEGIS 2.0: A diverse AI safety dataset and risks taxonomy for alignment of LLM guardrails. In *NAACL*, 2025.

[5] I. Gim, G. Chen, S.-s. Lee, N. Sarda, A. Khandelwal, and L. Zhong. Prompt cache: Modular attention reuse for low-latency inference. In *MLSys*, 2024.

[6] K. Greenewald, L. Lastras, T. Parnell, V. Shah, L. Popa, G. Zizzo, C. Gunasekara, A. Rawat, and D. Cox. Activated LoRA: Fine-tuned LLMs for intrinsics. *arXiv preprint arXiv:2504.12397*, 2025.

[7] J. Gu, X. Huang, L. Ye, S. He, W. Che, and T. Liu. A survey on LLM-as-a-judge. *arXiv preprint arXiv:2411.15594*, 2024.

[8] S. Han, K. Rao, A. Ettinger, L. Jiang, B. Y. Lin, N. Lambert, Y. Choi, and N. Dziri. WildGuard: Open one-stop moderation tools for safety risks, jailbreaks, and refusals of LLMs. *arXiv preprint arXiv:2406.18495*, 2024.

[9] M. Hoover, V. Baherwani, N. Jain, K. Saifullah, J. Vincent, C. Jain, M. K. Rad, C. B. Bruss, A. Panda, and T. Goldstein. DynaGuard: A dynamic guardian model with user-defined policies. *arXiv preprint arXiv:2509.02563*, 2025.

[10] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. LoRA: Low-rank adaptation of large language models. In *ICLR*, 2022.

[11] H. Inan et al. Llama Guard: LLM-based input-output safeguard for human-AI conversations. *arXiv preprint arXiv:2312.06674*, 2023.

[12] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with PagedAttention. In *SOSP*, 2023.

[13] A. Li, K. Greenewald, T. Parnell, and N. Azizan. Efficient multi-adapter LLM serving via cross-model KV-cache reuse with activated LoRA. *arXiv preprint arXiv:2512.17910*, 2025.

[14] Y. Liu et al. GuardReasoner: Towards reasoning-based LLM safeguards. *arXiv preprint arXiv:2501.18492*, 2025.

[15] T. Markov et al. A holistic approach to undesired content detection in the real world. In *AAAI*, 2023.

[16] A. Mazza and E. Levi. BARRED: Synthetic training of custom policy guardrails via asymmetric debate. Under review, 2025.

[17] J. Park et al. LRAgent: Efficient KV cache sharing for multi-LoRA LLM agents. *arXiv preprint arXiv:2602.01053*, 2026.

[18] Y. Sheng, S. Cao, D. Li, C. Hooper, N. Lee, S. Yang, C. Chou, B. Zhu, L. Zheng, K. Keutzer, J. E. Gonzalez, and I. Stoica. S-LoRA: Serving thousands of concurrent LoRA adapters. In *MLSys*, 2024.

[19] B. Wu, Y. Sheng, et al. dLoRA: Dynamically orchestrating requests and adapters for LoRA LLM serving. In *OSDI*, 2024.

[20] J. Yao, K. Li, et al. CacheBlend: Fast large language model serving with cached knowledge fusion. *arXiv preprint arXiv:2405.16444*, 2024.

[21] W. Zeng et al. ShieldGemma: Generative AI content moderation based on Gemma. *arXiv preprint arXiv:2407.21772*, 2024.

[22] H. Zhang, J. Shi, Y. Wang, Q. Chen, Y. Shan, and M. Guo. FastLibra: Improving the serving performance of multi-LoRA LLMs via efficient LoRA and KV cache management. *arXiv preprint arXiv:2505.03756*, 2025.

[23] J. Zhang, A. Elgohary, A. Magooda, D. Khashabi, and B. Van Durme. Controllable safety alignment: Inference-time adaptation to diverse safety requirements. In *ICLR*, 2025.

[24] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *NeurIPS*, 2023.

[25] L. Zheng, L. Yin, Z. Xie, J. Huang, C. H. Sun, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez, C. Barrett, and Y. Sheng. SGLang: Efficient execution of structured language model programs. *arXiv preprint arXiv:2312.07104*, 2023.