QLM: Queue Management for Large Language Model Serving

Archit Patke[1] Dhemath Reddy[1] Saurabh Jha[2] Christian Pinto[3] Haoran Qiu[1]
Shengkun Cui[1] Chandra Narayanaswami[2] Zbigniew Kalbarczyk[1] Ravishankar Iyer[1]

[1]*University of Illinois at Urbana-Champaign* [2]*IBM Research* [3]*IBM Research Europe*

Background. The emergence of language models such as OpenAI GPT4 and Google Gemini has enabled a wide range of new applications, including chatbots, log summarization tools, and coding assistants. Consequently, serving large language models (LLMs) has become an increasingly important workload for cloud providers, catering to both enterprise and consumer applications. LLM serving systems need to be scalable, ensure high throughput, reduce SLO violations all while maximizing resource utilization across compute units. However, achieving these goals simultaneously is challenging due to: (i) *High variation in number of output tokens*: LLMs fundamentally differ from other ML models due to their *auto-regressive* nature. This auto-regressive nature causes LLMs to generate their output sequentially from the input prompt, while updating their internal state (also known as the key–value (KV) cache) via the attention mechanism. The number of output tokens cannot be directly deduced from the input prompt. (ii) *Continuous batching*: State-of-the-art LLM serving systems such as vLLM [1], Orca, tensorRT-LLM, and TGI use the notion of continuous batching to dynamically fit requests in memory. The size of the GPU memory and length of the KV cache decides the continuous batch size. If the GPU memory is fully utilized, requests have to wait in the queue causing Head-Of-Line (HOL) blocking and adding variability in serving latency. (iii) *Variable model size and request arrival rates*: An LLM serving system needs to serve different models of variable size and arrival rates. These can lead to high cold start time and low compute utilization.

To address these challenges, LLM serving systems are often combined with a decision-making framework for resource provisioning mechanisms (RPMs) such as batch sizing, scheduling, load balancing, request preemption, operator placement, etc. *The objective of this work is to develop a decision-making framework that is able to satisfy SLOs while ensuring high resource utilization by dynamically deciding the RPMs at runtime*.

Limitations of Related Work. The auto-regressive nature of LLMs along with continuous batching impacts decision-making frameworks for resource provisioning mechanisms (RPMs). These frameworks rely on profiling the cost of neural network inference ahead-of-time and solving resource allocation as an optimization problem using techniques such as linear programming [3] and black-box ML optimization [2] . However, these frameworks fail to apply to LLMs because output time-per-query is variable and cannot be directly deduced at query time.

QLM Design. QLM is a queue management system that enables decision-making for RPMs targetting LLM serving. QLM consists of the LLM request waiting queue, optimizer loop, and RPMs enabled by the downstream LLM serving system. Incoming requests join the request queue, are assigned priorities by the optimizer, and sent to the LLM serving system that actuates the RPMs. The QLM optimizer enables performance service-level objectives (SLOs) by assigning higher priorities to requests with longer wait times and stricter SLO requirements. The optimizer is based on Bayesian statistics and hierarchical optimization strategies that estimate request completion times and assigns a priority to each request. QLM currently supports the following four RPMs and is being actively extended to support additional mechanisms:

- *Priority Scheduling:* (shown by ❶) The waiting queue is reordered based on assigned request priority levels. High-priority requests would be placed ahead in the waiting queue while others would be moved back, leading to a re-balancing in waiting times for requests in the queue.
- *Request Eviction:* (shown by ❷) High-priority requests in the waiting queue evict the low-priority requests in the continuous batch currently running on the GPU. The cost of eviction is the time required to recompute the KV cache for the evicted requests.
- *GPU-CPU State Swapping:* (shown by ❸) The KV cache is proactively swapped between GPU and CPU memory to make space for high-priority requests on demand. Swapping time is determined by the GPU-CPU memory bandwidth barrier.
- *Model Warm Start:* (shown by ❹) Models with high-priority requests can have their weights loaded into CPU memory ahead of time such that incoming requests do not have to wait for expensive loading times. This involves using CPU memory even during idling.

Evaluating QLM QLM can be paired together with any existing LLM inference server that supports these RPMs. However, we use vLLM to evaluate QLM. vLLM is a state-of-the-art serving system that uses PagedAttention [1] and continuous batching to serve requests at high throughput. We evaluate QLM on two models of varying sizes: Mistral-7B and Vicuna-13B and two GPUs of varying compute power: Nvidia A10 and A100. For the workload, we sample requests from ShareGPT [**?**] dataset. Our experiments with QLM demon-

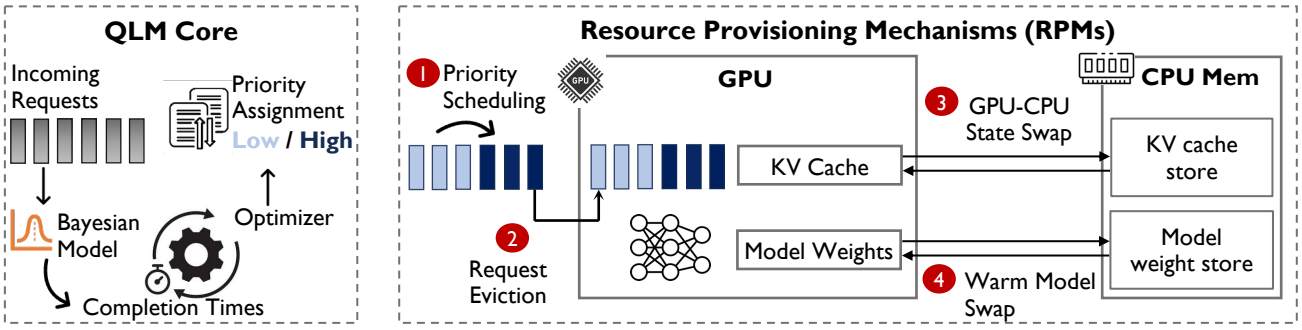

Figure 1: Overview of QLM.

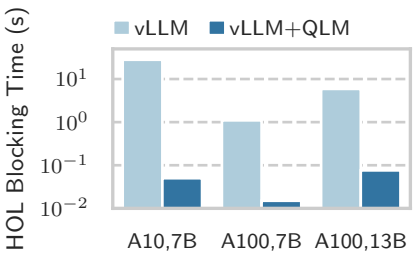

Figure 2: QLM reduces Head-Of-Line blocking time of requests.

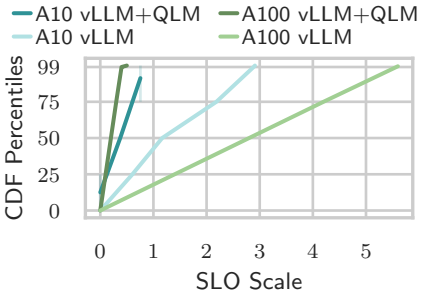

Figure 3: QLM reduces tail latency for high-priority requests.

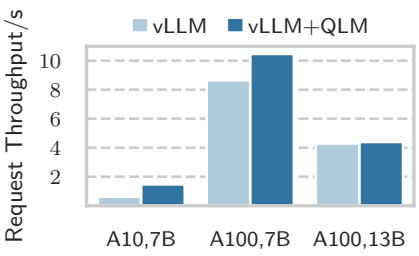

Figure 4: QLM improves overall request throughput.

strate the following characteristics of the LLM serving system:

QLM can eliminate head-of-the-line (HOL) blocking time for high-priority requests under bursty workloads. Under bursty workloads, priority scheduling reorders requests such that high-priority requests are placed ahead in the waiting queue. However, priority scheduling is not sufficient by itself because existing requests running in the GPU (continuous batch) prevent allocation of the high-priority request resulting in Head-Of-Line blocking. Therefore, in such scenarios, evicting requests from the GPU is required as supported by QLM. Figure 2 illustrates the HOL blocking time when run with a mixed workload comprising low-priority and high-priority requests. Request eviction can significantly reduce the waiting time because high-priority requests only need to wait for a single decoding iteration before they can be scheduled, resulting in a 100–1000× reduction in waiting time.

QLM selects between multiple RPMs to satisfy SLOs for high-priority requests. QLM is able to significantly lower tail latency by selecting between multiple RPMs. For example, when a high-priority request is blocked because of the already running requests in the GPU, two mechanisms can be used to allow immediate execution: request eviction and GPU-CPU state swapping. This choice primarily depends upon the memory bandwidth between GPU and CPU memory the compute power of the GPU, and the model architecture itself. As the GPU computation cost decreases, less time is required to recompute using the partially completed query and request eviction is preferable over swapping. If the size of the KV cache per output token increases relative to the time required to compute it, swapping would become preferable. QLM using posterior inference on the Bayesian model to decide which of these mechanisms to use. By deciding between tradeoffs such as swapping vs eviction, QLM better satisfies SLOs compared to the original LLM serving system. Figure 3 shows that QLM offers upto 3–6× reduction in SLO scale (i.e. ratio between tail latency and SLO latency) for high-priority requests compared to using vLLM only.

QLM can improve the throughput of vLLM even when SLOs are relaxed. While QLM prevents SLO violations for higher priority requests, it can also improve the throughput of vLLM even when no SLO constraints are present. For example, the benefit of QLM is noticeable during swapping vs preemption decision-making in vLLM. As an artifact of continuous batching, vLLM periodically preempts currently running requests out of the GPU, and later recomputes them or loads their saved KV cache from the CPU memory. By default, vLLM chooses the recompute option. However, QLM shows that this choice depends upon the GPU compute power and model size and the correct choice leads to improvement in request throughput. Figure 4 shows that request throughput for QLM can be 20% higher compared to vLLM.

Future Directions In this work, we showed the importance of using a decision engine to automatically manage RPMs through QLM. In future, we plan to enhance QLM in several ways: (i) *incorporate additional RPMs* such as auto-scaling, MIG partitioning, and configuration optimization of inferencing servers. (ii) *improve fault tolerance and scalability* by

adopting principles of distributed system design. (iii) *accommodate emerging models and deployment options* such as s-LoRA and RAG.

References

[1] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[2] Azalia Mirhoseini, Hieu Pham, Quoc V Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device placement optimization with reinforcement learning. In *International conference on machine learning*, pages 2430–2439. PMLR, 2017.

[3] Hong Zhang, Yupeng Tang, Anurag Khandelwal, and Ion Stoica. {SHEPHERD}: Serving {DNNs} in the wild. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 787–808, 2023.