Quality-of-Service Aware LLM Serving

Rithvik Chuppala and Siddharth Jha

Introduction

As applications continue to integrate LLM services at the frontend of user experience, **serving LLM queries with low-latency** guarantees is imperative. However, unpredictable workloads with bursty arrival rates requires system modifications and tradeoffs to meet user application requirements. We propose a two-pronged approach via **scheduling at the network level**, as well as **dynamic model selection**.

- We leverage the QUIC network protocol and its ability to multiplex several streams of data over a single point-to-point connection.
- We implement QUIC stream schedulers to provide QoS to client requests and model responses to meet the criteria of respective traffic classes
- We perform model selection by exploring tradeoffs in model complexity through a hybrid serving architecture.
- We employ an RL router agent to send client queries to appropriate models in order to maximize response quality while meeting user defined latency guarantees.
- We evaluate our system, analyzing current LLM application requirements, and running request loads to benchmark against existing model-serving techniques.

Background

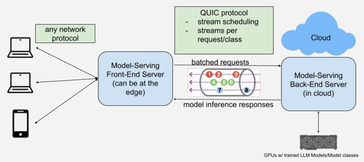
- Typical model serving systems do not dynamically adjust the quality of service in order to attend to periods of high request load. As such, they need to overprovision expensive GPU resources to handle such events or sacrifice service availability. This leads to either prohibitively high costs on the application developer or unacceptable latency for the user. We propose dynamically adjusting response quality to maximize client utility while meeting deadlines.
- DQN is a reinforcement learning algorithm that learns a Q-function which can be used to pick the best action given the state of the environment.
- As opposed to on-policy algorithms such as REINFORCE, TRPO, PPO, and A2C, off-policy algorithm such as DQN achieve better sample complexity.

- State-of-the-art model serving systems utilize REST API/gRPC endpoints over base HTTP/2 (TCP/TLS) network protocols. Not much research has gone into optimizing serving applications for network and traffic patterns observed in their use. As such, model-serving architectures are agnostic to application-specific network patterns and do not cater to QoS differentiation between requests.
- Previous attempts to multiplex a connection in TLS/TCP suffer from head-of-line blocking. QUIC resolves this by providing stream-level retransmission and congestion semantics
- However, baseline QUIC implementations do not provide rich scheduling abstractions for stream multiplexing. Most implementations only use a default Round Robin or FCFS scheduler between streams.

Metrics of Success

- Meet user-defined **latency deadlines and QoS requirements,** achieving high request completion rates.
- **Serve highest possible model quality** given resource contention
- **Handle both stable and unpredictable client requests patterns** across a wide range of client request rates.

Design and Implementation



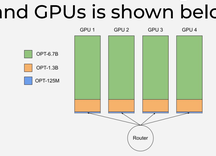
- Clients requests reach front-end server over any network protocol and medium.
- Front-end server can be at the edge or in cloud.
  - Batches requests and initiates request-response streams to model server.
- Each request assigned to a QUIC stream and scheduled according to traffic class.
- The RL agent analyzes the query and the load at each of the models and GPUs and determines the best model to route the query in order to maximize quality and meet latency deadlines.

Network Scheduling

- The frontend server establishes a QUIC connection with the model server.
- Each client request creates a QUIC stream which can be scheduled.
- We augment the open-source Quic-Go implementation with 4 scheduling interfaces, in addition to the default Round Robin.
  - EDF (deadline for each traffic class)
  - FCFS
  - Absolute Priority (priority level for each traffic class)
  - Multi-Level (higher level served via EDF, lower level served via Round Robin)
- We allow each traffic class to assume different requirements, utilizing the scheduler to best meet QoS.

Model Serving

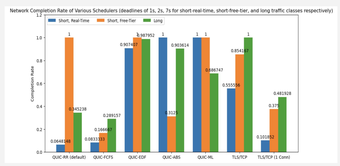
- Receives a request and determines which model and GPU to send it to.
- As state, it keeps track of is the task corresponding to the model, the current batch size at each of the models, and an estimate of the arrival rate.
- Each request has a corresponding deadline associated with it and the request must be served by that deadline or the reward for its service is zero. If it is served then the reward is the accuracy of the model.
- We train a RL agent with DQN to maximize the sum of rewards. An image of the various models and GPUs is shown below.

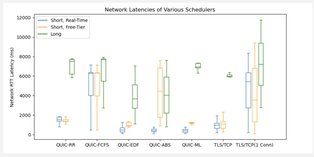


Design Benchmarking

Network Scheduling

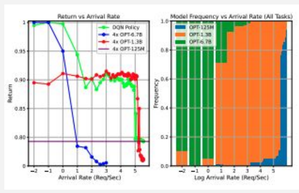
- We evaluate QUIC schedulers, along with QUIC-RR and TLS/TCP as a baseline.
- 3 traffic classes: real-time short (8 KB, 1 sec deadline), low-prio short (8 KB, 2 sec deadline), long input (102.4 KB, 7 sec deadline).
- Simulated 3G network - 5 Mbps, 50 ms addt'l latency, 0.5% packet drop.
- Client sends 80 requests, randomly selecting a traffic class; server echoes back to client
- Evaluate completion rate and average latency of each traffic class for all schedulers.

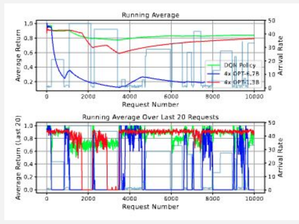




Model Serving

- We train the RL policy for 1.2 million iterations and evaluate both on a stable (top) and unpredictable workload (bottom). As seen, the policy outperforms static model serving baselines in both cases and dynamically switches model usage.





System Evaluation

- We evaluate with three classes of request traffic
  - High priority traffic with small input sizes (common LLM usage)
  - Low priority traffic with small input sizes (common LLM usage)
  - Low priority traffic with large input sizes (large text synthesis)
- We assign an end-to-end per-token latency requirement of 1800 ms for the small input streams and a latency requirement of 2300 ms for the large input stream.
- We implement a variety of network schedulers as mentioned.
- We train a DQN policy for 1.2 million iterations to act as the router agent.
- We double the RL reward for high priority traffic and calculate the reward as the accuracy if the deadline is met, otherwise zero.
- Below we show the performance on a stable workload where the arrival rate is fixed for a fixed time before transitioning to the next arrival rate. The policy and EDF outperforms all baselines.

Performance Scores Across LLM Models and Network Schedulers

	QUIC-RR (default)	QUIC-FCFS	QUIC-EDF	QUIC-ABS	QUIC-ML	TLS/TCP	TLS/TCP (1 conn)
Large Model	0.20	0.19	0.21	0.20	0.21	0.20	0.18
Medium Model	0.25	0.24	0.37	0.35	0.29	0.24	0.22
Dynamic Routing	0.42	0.36	0.52	0.49	0.45	0.41	0.34

- Below we show the performance on an unpredictable workload where the arrival rate and its duration vary randomly. The policy outperforms static serving with the large model but does not outperform static serving with the medium model.

Performance Scores Across LLM Models and Network Schedulers

	QUIC-RR (default)	QUIC-FCFS	QUIC-EDF	QUIC-ABS	QUIC-ML	TLS/TCP	TLS/TCP (1 conn)
Large Model	0.19	0.17	0.19	0.18	0.19	0.18	0.16
Medium Model	0.54	0.53	0.64	0.62	0.57	0.54	0.51
Dynamic Routing	0.42	0.36	0.48	0.46	0.44	0.41	0.35

- In both workloads, we see that EDF performs the best regardless of the serving mechanism. Then, Absolute Priority scheduling and Multi-Level scheduling perform the best.
- In all cases QUIC outperforms TLS/TCP.

Future Work

- Understand why the policy does worse than static serving with the medium model on the unpredictable workload and close the gap.
- Implement dynamic scheduler which changes algorithms to request load.
- Train the policy to take request's network delay into account.

References

- Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).
- Kwon, Woosuk, et al. "Efficient memory management for large language model serving with pagedattention." Proceedings of the 29th Symposium on Operating Systems Principles. 2023.
- Langley, Adam, et al. 2017. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In Proceedings of SIGCOMM '17.
- J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport." RFC 9000, May 2021.
- L. Clemente and M. Seemann, "A QUIC implementation in pure Go," 2022. Accessed: November, 2023.