DNN-Adapt: Reinforcement Learning-based Hybrid Batching for Efficient DNN Serving

Milind Varma
Department of Computer Science
University of California, Santa Cruz
Santa Cruz, USA

Sai Venkat Malreddy
Department of Computer Science
University of California, Santa Cruz
Santa Cruz, USA

Liting Hu
Department of Computer Science
University of California, Santa Cruz
Santa Cruz, USA

***Abstract*—Deep Neural Network (DNN) inference serving presents significant challenges due to variable workloads, heterogeneous models, and strict Service Level Objectives (SLOs). While GPU acceleration enables high-throughput inference, the cost-effectiveness of these specialized resources depends on maintaining high utilization. Current DNN serving systems employ static or heuristic-based batching strategies that cannot adapt effectively to dynamic cloud workloads. We present DNN-Adapt, a novel system that employs reinforcement learning (RL) to dynamically optimize batching decisions in DNN serving environments. DNN-Adapt introduces a hybrid batching framework that combines traditional batching techniques with RL-based decision making, a multi-timescale architecture that operates at different levels of granularity, and a hybrid decision system that integrates rule-based safety constraints with learned policies.**

***Index Terms*—DNN inference scheduling, reinforcement learning, batching, GPU utilization**

I. Introduction

Deep Neural Network (DNN) inference serving has become a critical workload for applications ranging from computer vision to natural language processing. As these services grow in scale and importance, optimizing the efficiency of inference execution becomes essential for both cost-effectiveness and service quality. GPU accelerators provide the computational capacity needed for DNN inference, but their high cost demands sustained utilization to realize their economic benefits.

A key technique for improving GPU utilization is batching—grouping multiple inference requests together for simultaneous processing. Batching amortizes fixed costs and leverages the GPU's parallel processing capabilities, significantly increasing throughput. However, current batching approaches suffer from several limitations:

1) **Workload Variability**: Inference serving systems experience dynamic request patterns with diurnal variations, sudden spikes, and unpredictable bursts that static batching strategies cannot effectively handle.
2) **Multi-tenancy Challenges**: Inference services typically run multiple models concurrently, creating complex resource contention patterns that heuristic-based approaches struggle to navigate.
3) **Heterogeneous Model Landscape**: Models with different computational profiles require tailored batching strategies, making one-size-fits-all approaches suboptimal.
4) **Complex Optimization Objectives**: Inference serving systems must balance throughput, latency, fairness, and resource efficiency, often with dynamically changing priorities.

In this Work-in-Progress (WIP) paper, we introduce DNN-Adapt, a novel batching system for DNN inference that builds upon the predictability principles of Clockwork [1] while adding adaptive intelligence through reinforcement learning (RL). By framing batching as a sequential decision-making problem, DNN-Adapt learns optimal strategies from experience, continuously adapting to changing conditions without requiring explicit programming for every possible scenario.

Our key contributions include:

- A hybrid batching framework that combines traditional batching techniques with reinforcement learning to optimize multiple objectives.
- A hybrid decision system that integrates rule-based safety guarantees with learned policies to ensure both reliability and optimality.

II. Background and Motivation

A. Batching to achieve throughput

Batching is a critical technique that improves GPU utilization through two mechanisms:

1) **Fixed cost amortization**: The overhead of launching GPU kernels and data transfer is distributed across multiple requests.
2) **Parallelism exploitation**: GPU architectures can process multiple inputs simultaneously, often achieving higher throughput with batched execution.

The relationship between batch size and throughput typically follows a saturating curve, while the relationship between batch size and latency is generally linear for many models:

$$batch\_latency(b) = \alpha b + \beta \tag{1}$$

Where β is the fixed cost of model invocation and α is the marginal cost per request.

B. Prior work

Prior work, notably Nexus [2], Usher [3] and Clockwork [1], have made significant strides in optimizing DNN serving through advanced scheduling and resource allocation techniques. Nexus [2] formulates inference serving with batching as a MILP and proposes a greedy algorithm for generating round robin schedules on GPUs, Usher [3] proposes a similar technique but emphasises both GPU

2025 IEEE 18th International Conference on Cloud Computing (CLOUD) | 979-8-3315-5557-3/25/$31.00 ©2025 IEEE | DOI: 10.1109/CLOUD67622.2025.00054

2159-6190/25/$31.00 ©2025 IEEE
DOI 10.1109/CLOUD67622.2025.00054

memory and compute utilization with models colocated on the same GPU, while Clockwork [1] emphasized predictable executions through "consolidating choice" at the system level. However, these systems rely on fixed strategies or simple reactive policies that cannot fully leverage the inherent predictability of DNN inference while adapting to the dynamic nature of inference workloads.

C. Challenges in Cloud Environments

DNN Inference Serving in Cloud Environments Cloud-based DNN inference services execute models on input data, returning results to clients with strict latency requirements. The computational intensity of modern DNNs makes GPUs the preferred accelerator for these workloads. In cloud environments, these services must handle multiple concurrent requests from many users while maintaining SLOs. Cloud-based DNN serving presents several unique challenges:

Workload volatility: Cloud traffic exhibits significant variability, including diurnal patterns, sudden spikes, and unpredictable bursts. Static batching strategies either under-utilize resources during low traffic or violate SLOs during high traffic.

Multi-tenant resource sharing: Cloud servers typically execute multiple models concurrently, creating complex resource contention patterns. Optimal batching decisions must account for this interference.

Model heterogeneity: Different models have distinct computational characteristics, including optimal batch sizes, memory requirements, and execution patterns.

D. Reinforcement Learning for Adaptive Batching

Reinforcement learning provides an ideal framework for addressing these challenges. The RL paradigm consists of:

- **State**: The current condition of the system (queue lengths, GPU utilization, etc.) - **Action**: Decisions about batch execution (timing, size, etc.) - **Reward**: A scalar value quantifying the desirability of the outcome - **Policy**: A mapping from states to actions that maximizes cumulative rewards

The key advantage of RL for this problem is its ability to learn optimal strategies through experience rather than requiring explicit programming for every possible scenario.

III. System Design

A. Architecture Overview

DNN-Adapt is designed as a cloud-native system for optimizing DNN inference serving. Fig. 1 shows its high-level architecture, which consists of:

1) **Request Processor**: Receives and validates inference requests, routing them to appropriate model queues.
2) **Model Queues**: Maintain pending requests with metadata including arrival time and priority.
3) **State Observer**: Monitors system conditions including queue states, GPU utilization, and performance metrics.
4) **RL Decision Engine**: Makes batching decisions based on current state and learned policy.

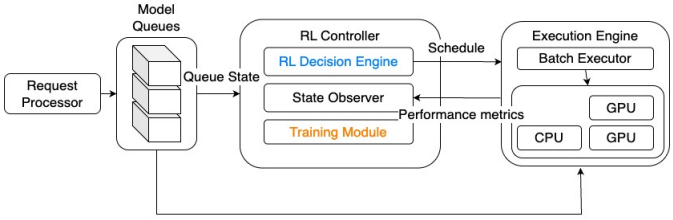

Fig. 1. DNN-Adapt System Design

5) **Execution Engine**: Executes the current schedule by batching requests and running inference on available GPUs.
6) **Training Module**: Updates the RL model based on the observed performance.

B. Reinforcement Learning Formulation

We have come up with the following simple formulation to validate our idea. Each GPU is divided into S number of slots. Initially, all slots are empty. The RL-agent can take an action to place a certain model in a certain slot with a certain batch size. For a given GPU, in one schedule, all slots are executed one after the other, starting with slot 1. There is an additional option for the RL-agent to specify whether to run a particular slot in parallel with others. This means that all consecutive slots with parallel flag set are run in parallel on that GPU. The rest are run sequentially.

1) State Representation: For each model, we capture the following as part of our state information:

- Request rate.
- Queue size.
- SLO latency.
- SLO satisfaction/violation rate.

We also capture the current schedule deployed. For each slot in the cluster:

- Model id deployed(can be empty)
- Batch size
- parallel flag

2) Action Space: The action space is designed to allow the RL agent to make incremental changes. At each step, the agent can decide to deploy a model in a given slot with given batch size or it can decide to change the batch size of an already deployed model. The action space is represented as follows:

- Slot ID(including no-op)
- Model ID(including empty)
- Batch Delta - [-4, 5]
- Parallel Flag

3) Reward Function: The reward function encodes the objectives of the system as a weighted sum of the following components:

- R_{SLO}, reward related to the SLO satisfaction rate.
- R_{GPU}, reward related to the number of GPUs being used by the schedule. The lower the better.

- R_{BFR}, reward related to batch fill rate, it encourages the agent to select batch sizes that are consistently filled, discouraging over-sized batches that do not reach capacity.
- R_{SSR}, reward related to slot switch rate, designed to discourage frequent re-assignment of models to different slots.
- $SLO_{mask} = 1 - R_{SLO}$, Masked used to prioritize the SLO satisfaction rate over the other reward terms.

$$R = \alpha \cdot R_{SLO} + \beta \cdot R_{GPU} \cdot SLO_{mask} + \gamma \cdot R_{BFR} \cdot SLO_{mask} + \omega \cdot R_{SSR} \cdot SLO_{mask} \quad (2)$$

4) Learning Algorithm and Training: DNN-Adapt employs Proximal Policy Optimization (PPO) [4] as its core reinforcement learning algorithm, chosen for its sample efficiency, stability.

IV. Related Work

A. DNN Serving Systems

Several specialized systems have been developed for serving DNN models in production environments. TensorFlow Serving [5] and PyTorch Serve [6] provide frameworks for deploying trained models but offer limited support for advanced batching strategies. Clipper [7] introduced adaptive batching, which dynamically adjusts batch sizes based on observed latency, but uses simple heuristics rather than learning from experience.

Nexus [2] represents a significant advancement for DNN serving on GPU clusters, introducing sophisticated scheduling techniques including "squishy bin packing" and query optimization for complex workflows. Although Nexus achieves high throughput while meeting latency SLOs, its epoch-based scheduling limits adaptability to rapidly changing workloads.

B. Batching Optimization

Batching optimization has been explored through several approaches: (1) Static batching uses fixed batch sizes determined offline; (2) Time-based batching collects requests over fixed intervals but struggles with bursty traffic; (3) Size-based batching triggers execution once a threshold is met, often increasing latency under low load; and (4) Dynamic batching adjusts sizes based on observed performance, though typically relies on simple heuristics rather than learned policies.

DNN-Adapt differs fundamentally by framing batching decisions as a reinforcement learning problem, enabling more sophisticated and adaptive strategies.

C. Reinforcement Learning for Systems Optimization

Reinforcement learning has emerged as a powerful technique for optimizing complex systems. In cloud computing contexts, RL has been applied to resource allocation [8], query optimization [9], and adaptive video streaming [10].

The application of RL to DNN inference batching represents a novel contribution of our work. While previous systems have used RL for related problems, none have specifically addressed the unique challenges of batching optimization in cloud-based DNN serving environments.

V. Conclusion and Future Work

DNN-Adapt introduces a reinforcement learning-based hybrid batching approach to optimize DNN inference serving in cloud environments. While our initial design focuses on adapting to dynamic workload patterns and multi-tenant scenarios, several key areas remain for future exploration.

As this is a work in progress, we do not yet have comprehensive results to report. In future work, we intend to further develop the RL agent to achieve converged policies capable of consistently satisfying latency SLOs. We also plan to scale our system to operate on larger GPU clusters with more diverse model deployments and explore how DNN-Adapt can generalize across different production settings and workload types.

References

[1] A. Gujarati, R. Karimi, S. Alzayat, W. Hao, A. Kaufmann, Y. Vigfusson, and J. Mace, "Serving dnns like clockwork: Performance predictability from the bottom up," 2020. [Online]. Available: https://arxiv.org/abs/2006.02464

[2] H. Shen, L. Chen, Y. Jin, L. Zhao, B. Kong, M. Philipose, A. Krishnamurthy, and R. Sundaram, "Nexus: a gpu cluster engine for accelerating dnn-based video analysis," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, ser. SOSP '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 322–337. [Online]. Available: https://doi.org/10.1145/3341301.3359658

[3] S. S. Shubha, H. Shen, and A. Iyer, "USHER: Holistic interference avoidance for resource optimized ML inference," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. Santa Clara, CA: USENIX Association, Jul. 2024, pp. 947–964. [Online]. Available: https://www.usenix.org/conference/osdi24/presentation/shubha

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[5] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: a system for large-scale machine learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16. USA: USENIX Association, 2016, p. 265–283.

[6] PyTorch Team, "Torchserve," https://pytorch.org/serve/, 2020, accessed: 2025-06-05.

[7] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: a low-latency online prediction serving system," in *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'17. USA: USENIX Association, 2017, p. 613–627.

[8] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, ser. HotNets '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 50–56. [Online]. Available: https://doi.org/10.1145/3005745.3005750

[9] R. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil, and N. Tatbul, "Neo: a learned query optimizer," *Proc. VLDB Endow.*, vol. 12, no. 11, p. 1705–1718, Jul. 2019. [Online]. Available: https://doi.org/10.14778/3342263.3342644

[10] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 197–210. [Online]. Available: https://doi.org/10.1145/3098822.3098843