MARCA: MULTI-AGENT REINFORCEMENT LEARNING FOR DYNAMIC COMPUTATION ALLOCATION IN LARGE-SCALE RECOMMENDER SYSTEMS

Wan Jiang[1†*] **Xinyi Zang**[1] **Yudong Zhao**[1] **Yusi Zou**[1] **Yunfei Lu**[1]
Junbo Tong[2] **Yang Liu**[1] **Ming Li**[1] **Jiani Shi**[1] **Xin Yang**[1]

[1]JD.com, Beijing, China [2]Tsinghua University, Beijing, China
{jiangwan1, zangxinyi3, zhaoyudong10, zouyusi1, luyunfei1}@jd.com,
{liuyang123, liming666, shijiani, yangxin81}@jd.com,
tjb21@mails.tsinghua.edu.cn

ABSTRACT

Modern recommender systems face significant computational challenges due to growing model complexity and traffic scale, making efficient computation allocation critical for maximizing business revenue. Existing approaches typically simplify multi-stage computation resource allocation, neglecting inter-stage dependencies, thus limiting global optimality. In this paper, we propose **MaRCA**, a **m**ulti-**a**gent **r**einforcement learning framework for end-to-end **c**omputation resource **a**llocation in large-scale recommender systems. MaRCA models the stages of a recommender system as cooperative agents, using Centralized Training with Decentralized Execution (CTDE) to optimize revenue under computation resource constraints. We introduce an AutoBucket TestBench for accurate computation cost estimation, and a Model Predictive Control (MPC)-based Revenue-Cost Balancer to proactively forecast traffic loads and adjust the revenue-cost trade-off accordingly. Since its end-to-end deployment in the advertising pipeline of a leading global e-commerce platform in November 2024, MaRCA has consistently handled hundreds of billions of ad requests per day and has delivered a 16.67% revenue uplift using existing computation resources.

1 INTRODUCTION

Modern recommender systems analyze user behavior and contextual information to filter relevant items from large candidate pools and generate a ranked list of recommendations through multi-stage processing pipelines (Cheng et al., 2016; Zhou et al., 2018). These systems have become crucial infrastructure for e-commerce platforms (Tawfiq et al., 2021).

Contemporary industrial recommender systems typically adopt a cascaded architecture comprising three stages: retrieval, pre-ranking, and ranking (Zhao et al., 2023; Liu et al., 2017). Each stage of recommender systems involves models of different complexity and is subject to specific computational constraints. However, most early academic research on recommender systems aims to maximize profit under the assumption of abundant computation resources (Zhao et al., 2021; 2020; Deng et al., 2021), failing to address resource allocation constraints.

With the continued advancement of deep learning-driven recommender systems, the tension between computational demand and available machine resources has become increasingly significant. In industrial recommender systems, traffic volume varies significantly across different periods (Koren, 2009), and the value of requests differs across media platforms and user demographics (Neophytou et al., 2022). An effective computation allocation strategy should dynamically adapt to fluctuating traffic conditions while maintaining system stability and high recommendation quality within limited machine resources. (An illustrative overview is provided in Appendix A).

*†Corresponding author.

To address these issues, recent efforts in Dynamic Computation Allocation (DCA) have explored two major categories of approaches: optimization-based methods and reinforcement learning (RL)-based methods. Optimization-based methods, such as linear programming and heuristic scheduling, allocate computation resources based on predefined constraints and business rules. These methods offer fast and interpretable decision-making but often struggle to adapt to dynamic traffic fluctuations and evolving system conditions. In contrast, RL-based methods frame resource allocation as a sequential decision-making problem, enabling flexible strategies by learning from historical data and real-time interactions. However, many reinforcement learning methods have adopted either a single-agent paradigm or centralized multi-agent architectures, which tend to overlook the localized operational constraints inherent in distributed recommender systems.

In industrial settings, key stages of the recommendation pipeline are deployed across different data centers, each operating under distinct constraints and relying on localized observations. Centralized approaches combine all decisions into a single prediction, which may overlook localized factors. To overcome these constraints, we propose our framework, a cooperative multi-agent framework that models each stage as an autonomous agent. Using Centralized Training with Decentralized Execution (CTDE), each agent makes decisions using local information while benefiting from coordinated global learning. This approach leverages cooperative agent collaboration to maximize business revenue under limited computation resources. Extensive offline experiments of our framework [1] and a four-week online A/B test on a major e-commerce platform demonstrate that our framework achieves a **16.67% increase in advertising revenue** without additional computation cost (see more details in Section 4).

Our main contributions are summarized as follows:

1. We propose a novel collaborative multi-agent framework that leverages the **Adaptive Weighting Recurrent Q-Mixer (AWRQ-Mixer)**, which integrates an adaptive weighting recurrent Q for sequential decision-making with a mixing network to foster cross-stage coordination. By employing CTDE, our framework achieves globally optimal resource allocation.
2. We develop an **MPC-based Revenue-Cost Balancer** that optimally allocates resources by forecasting traffic fluctuations. This approach enables real-time, proactive decision-making, and therefore enhances system stability and efficiency under dynamic resource constraints.
3. We design an **AutoBucket TestBench** for computation cost estimation (see more details in Appendix C). Our framework employs automated testing and intelligent bucketing to address the absence of explicit cost labels.

2 Related Work

Dynamic computation allocation (DCA) has evolved from deterministic optimization to learning-based methods to address the scale and complexity of industrial recommender systems.

Deterministic Optimizers for Computation Allocation Early solutions relied on heuristic-based elastic degradation or static rules (e.g., disabling heavy models) (Beyer et al., 2016; Somarapu, 2024), which are robust but value-agnostic. To address this, DCAF (Jiang et al., 2020) formulated resource allocation as a 0–1 knapsack problem maximizing global revenue. CRAS (Yang et al., 2021) augmented this with a PID controller (Ang et al., 2005) to improve stability during bursts. Subsequent works like SACA (Shunhui et al., 2021) and GreenFlow (Lu et al., 2023) introduced elastic models and dynamic primal–dual solvers to handle multi-stage actions. However, these methods rely on a key simplification that recommendation stages are independent with static costs (Lu et al., 2023). Furthermore, their purely reactive control often leads to oscillations, limiting their ability to capture non-stationary inter-stage dependencies.

Learning-Based Approaches for Computation Allocation To handle dynamic fluctuations, researchers applied Reinforcement Learning (RL). Classic variants like DQN (Mnih et al., 2015),

[1] The publicly accessible code at: https://github.com/jd-opensource/MaRCA

DRQN (Hausknecht & Stone, 2015), Averaged-DQN (Anschel et al., 2017), and REM (Agarwal et al., 2020) address partial observability and value estimation but typically operate under a single-agent paradigm. A representative baseline, RL-MPCA (Zhou et al., 2023), coordinates the pipeline using a centralized agent. While effective, centralized approaches conflict with the decentralized reality of industrial systems where stages run on separate service clusters (Zhao et al., 2023; Covington et al., 2016).

This necessitates Multi-Agent RL (MARL) formulations (Huh & Mohapatra, 2024). We adopt the Centralized Training with Decentralized Execution (CTDE) paradigm (Qin et al., 2024a), which balances global coordination with local execution. Within CTDE, we favor value-decomposition methods (e.g., VDN (Sunehag et al., 2018), QMIX (Rashid et al., 2020b)) over centralized-critic algorithms (Lowe et al., 2017; Yu et al., 2022), as the latter struggle with the exponential joint-action spaces of recommender systems. While MARL has been applied to ad bidding (Jin et al., 2018) and ranking (Qin et al., 2024b), our framework is the first to apply cooperative MARL to end-to-end resource orchestration.

3 METHODOLOGY

3.1 PROBLEM FORMULATION

To address the challenges outlined in the previous sections, we formulate the multi-stage recommendation process as a constrained sequential decision-making problem that aims to maximize overall business revenue while adhering to strict computation resource constraints. We formally define the following:

- **State Space $\mathcal{S}$.** At each step t, the system observes a state $s_t \in \mathcal{S}$, encapsulating user profile features, real-time traffic patterns, and resource utilization metrics.
- **Action Space $\mathcal{A}$.** At each step t, the joint action combination is $\mathbf{a} = (a_1, \ldots, a_n) \in \mathcal{A}$. The joint action space is $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$. The action combination $\mathbf{a}$ collectively specifies the decisions across multiple stages, including retrieval channels to activate, switch modules to enable, and queue truncation lengths.
- **Action Value $Q(s_t, a_t)$.** Given the current state s_t and action a_t, $Q(s_t, a_t)$ denotes the expected business revenue.
- **Computation Cost $C(s_t, a_t)$.** $C(s_t, a_t)$ represents the computation resources required to execute the action a_t given the current state s_t. We define: $C(s_t, a_t) = \hat{C}(s_t, a_t) + f(D_t)$ where $\hat{C}(\cdot)$ is the computation cost predicted by the AutoBucket TestBench (see more details in Appendix C) and D_t is the elastic degradation level mapped into equivalent computation cost through $f(\cdot)$ (calibrated in isolated load tests).
- **Reward $R(s_t, \mathbf{a})$.** The reward function is designed as business revenue. Unlike conventional step-wise rewards, our system can only observe business revenue after the entire action sequence for a request is completed.

Consider a batch of M user requests indexed by i. For a request i, the system assigns a binary decision variable $x_{i,\mathbf{a}} \in \{0, 1\}$ indicating whether a specific action combination $\mathbf{a}$ is selected ($x_{i,\mathbf{a}} = 1$) or not ($x_{i,\mathbf{a}} = 0$). We impose a computation resource budget C_m to limit the overall computation cost over all requests. Following (Jiang et al., 2020), we formulate the constrained optimization problem as stated in Eqs. (1)–(2).

$$\max_{x_{i,\mathbf{a}}} \quad \sum_{i=1}^{M} \sum_{\mathbf{a} \in \mathcal{A}} x_{i,\mathbf{a}} Q(s_t, \mathbf{a}) \tag{1}$$

$$\text{s.t.} \quad \sum_{i=1}^{M} \sum_{\mathbf{a} \in \mathcal{A}} x_{i,\mathbf{a}} C(s_t, \mathbf{a}) \leq C_m; \quad \sum_{\mathbf{a} \in \mathcal{A}} x_{i,\mathbf{a}} = 1, \ \forall i; \quad x_{i,\mathbf{a}} \in \{0, 1\} \tag{2}$$

We enforce a one-hot structure over the joint action space. For each request i, exactly one composite action is executed.

To satisfy the budget constraint in Eqs. (2) while maximizing total revenue in Eq. (1), we adopt a Lagrangian relaxation approach, as in (Jiang et al., 2020). The complete derivation is provided in Appendix B. This yields the following request-level decision rule:

$$\mathbf{a}^* = \arg\max_{\mathbf{a}\in\mathcal{A}} \left(Q(s_t, \mathbf{a}) - \lambda C(s_t, \mathbf{a})\right) \tag{3}$$

Here, $\mathbf{a}^*$ represents the chosen action combination that maximizes the net benefit, measured by the revenue $Q(s_t, \mathbf{a})$ minus the λ-weighted cost $C(s_t, \mathbf{a})$. Through our framework, we can learn appropriate policy parameters and dynamically adapt λ based on real-time load.

3.2 System Design

As illustrated in Figure 1, the system follows a collaborative multi-agent framework, where the AWRQ-Mixer and AutoBucket TestBench feed their computed metrics into the MPC-based Balancer, which then orchestrates the final action selection. The AWRQ-Mixer assesses the expected business revenue $Q(s_t, \mathbf{a})$ under various states and expected action combinations. Meanwhile, the AutoBucket TestBench (see more details in Appendix C) processes trace-log data and predicted action outcomes to estimate the computation cost $C(s_t, \mathbf{a})$ for each action combination. Subsequently, the MPC-based revenue-cost balancer dynamically selects optimal actions by balancing $Q(s_t, \mathbf{a})$ and $C(s_t, \mathbf{a})$, guided by real-time resource utilization.

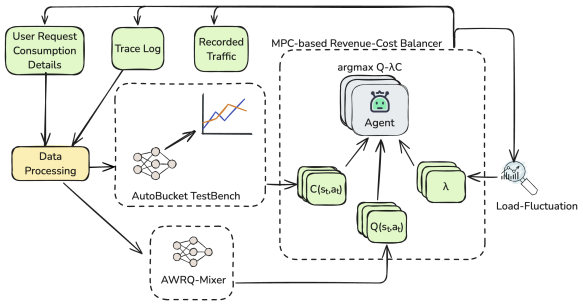


Figure 1: System architecture: multi-agent collaborative decision flow with Adaptive Weighting Recurrent Q-Mixer, AutoBucket TestBench, and MPC-Based Revenue-Cost Balancer.

3.3 Adaptive Weighting Recurrent Q-Mixer

The action value estimation module, AWRQ-Mixer, predicts the expected revenue of each request by jointly encoding user attributes, contextual information, and the inter-stage dependencies in the recommendation stages.

To illustrate why modeling these interdependencies matters, consider that the stages of a recommender pipeline are highly interdependent, with decisions made upstream directly constraining what can be achieved downstream. For example, changes in retrieval actions can alter the candidate pool and thus the final revenue, even when the ranking actions remain the same. However, due to the independent operation of these stages in separate service clusters with resource constraints (Zhou et al., 2023) (Cai et al., 2023), it is necessary to model their interdependencies while maintaining their ability to independently manage computation costs.

AWRQ-Mixer meets this requirement through three innovations: (1) an adaptive weighting recurrent Q ensemble that dynamically integrates multiple Q-value estimators; (2) a variance-guided credit assignment mechanism to allocate reward among actions; and (3) a softplus-based monotonicity constraint that ensures cooperative aggregation of agent-level values. The following subsections will detail each of these innovations.

3.3.1 Adaptive Weighting Recurrent Q (AWRQ)

Traditional DQN performs well in fully observable environments but is limited in partially observable environments due to incomplete state information. In such contexts, historical observations become crucial for accurately inferring the underlying state. Therefore, we employ DRQN to process sequences of observations over time: $h_t = \mathrm{GRU}\big(o_t, h_{t-1}\big), \quad Q(o_t, a_t) = \mathrm{MLP}(h_t)$. where o_t is the observation, and h_t is the hidden state capturing historical context.

However, a single Q-value estimator in DRQN is insufficient to capture the multi-faceted, cross-stage decision process. Inspired by ensemble learning principles, we extend DRQN by introducing parallel recurrent Q-value estimators to address uncertainty in the estimation process. Each agent instantiates a recurrent ensemble of K heads. For each head $k \in \{1, \ldots, K\}$, it outputs $Q^k_{\theta_g}(s_t, a_t)$, where $g \in \{1, \ldots, n\}$ indexes the agents. For brevity, Q_{θ_g} is henceforth written as Q_g whenever parameters are clear from context. Rather than averaging, we dynamically weight ensemble outputs according to their temporal difference (TD) errors, and we call this method Adaptive Weighting (AW).

At each training step, each Q-head's TD error is recorded. If a head exhibits a larger error on a given mini-batch, it is assigned a correspondingly larger weight among the heads. This emphasises under-performing heads, ensuring they receive greater focus during training and can be corrected more quickly. Empirically, this weighting scheme accelerates convergence and enhances robustness.

Consider a mini-batch in which each sample contains the state s_t, the action a_t, and the next state s_{t+1}. The loss for each Q-head is then computed as follows:

$$\mathcal{L}_{k,t} = \big(r_t + \gamma Q^k_{g'}(o_{t+1}, a_{t+1}) - Q^k_g(o_t, a_t)\big)^2 \tag{4}$$

where γ is the discount factor, and $Q_{g'}(o_{t+1}, a_{t+1})$ is the target Q-value for the next state-action pair, calculated by the target network.

The individual losses are normalized to compute the adaptive weight $\eta_{k,t}$ for each Q-head:

$$\eta_{k,t} = \frac{\mathcal{L}_{k,t}}{\sum_{k=1}^{K} \mathcal{L}_{k,t}} \tag{5}$$

Finally, the agent's Q-value is taken as the weighted sum:

$$Q_g = \sum_{k=1}^{K} \eta_{k,t}\, Q^k_g(o_t, a_t) \tag{6}$$

3.3.2 SOFTPLUS-BASED MONOTONICITY CONSTRAINTS (SMC)

In multi-agent recommendation pipelines, the joint action-value Q_{tot} must be non-decreasing in each agent's value Q_g to reflect their cooperative contribution. To enforce this, we define a mixing network $\mathcal{M}$ that aggregates individual Q-values into a joint value $Q_{tot} = \mathcal{M}(Q_1, Q_2, \ldots, Q_n, s_t)$. The mixing network parameters are generated by a hypernetwork taking s_t as input: $(\tilde{W}_1,\, b_1,\, \tilde{W}_2,\, b_2) = h_\psi(s_t)$.

Because the stages cooperate, the joint value must be monotonic non-decreasing in every agent's value. We enforce this by applying the Softplus transform, i.e., $W_i = \text{Softplus}(\tilde{W}_i) = \ln(1 + e^{\tilde{W}_i})$ for $i \in \{1, 2\}$, to ensure non-negativity.

The mixing network is trained by minimizing

$$\mathcal{L}\,(\theta_{\text{tot}}) = \left(r_t + \gamma Q_{\text{tot}'}\,(s_{t+1}, \mathbf{a}') - Q_{\text{tot}}\,(s_t, \mathbf{a})\right)^2 \tag{7}$$

3.3.3 VARIANCE-GUIDED CREDIT ASSIGNMENT (**VGCA**)

In environments with sparse or delayed rewards, such as large-scale recommendation systems, it's crucial to determine how each action contributed to the final reward. To address this, we introduce an auxiliary reward signal that mitigates sparse rewards and enhances training stability. The key insight is using variance across candidate actions to guide their contribution to the final reward.

$$w_t = \mathbb{V}\text{ar}_{j \in \mathcal{A}_t}\big[\,\mathbb{E}[\,R \mid a_t = j\,]\,\big] \tag{8}$$

where r_t is the reward and $\mathcal{A}_t$ is the discrete action space of a_t. During TD updates, the reward r_t of each action is scaled by w_t, amplifying learning signals for high-impact dimensions. This

adjustment ensures that agents whose actions induce greater variance, and therefore have a greater potential impact on revenue, receive proportionally stronger learning signals. This auxiliary signal serves as an additional guide to the model during training, helping it better identify and prioritize the most relevant actions, even in the absence of frequent immediate feedback. Algorithm 1 summarizes the overall training procedure for AWRQ-Mixer.

Bringing these elements together, AWRQ-Mixer extends a DRQN backbone with (1) Adaptive Weighting Recurrent Q, (2) Variance-Guided Credit Assignment, and (3) Softplus-Based Monotonicity Constraints. As illustrated in Figure 2, the framework models the recommendation stages as cooperative agents through AWRQ. The mixing network aggregates Q-values from AWRQ while enforcing monotonicity, guided by state information s_t to dynamically tailor mixing weights using a hypernet. At training time, we adopt centralized optimization of all agents, allowing us to capture global dependencies. At inference time, each agent can operate independently in a separate cluster, thus enabling scalable decentralized execution without extra cross-agent communication overhead. This design is particularly critical in production environments where stages such as retrieval and ranking often run in physically separate machine clusters.

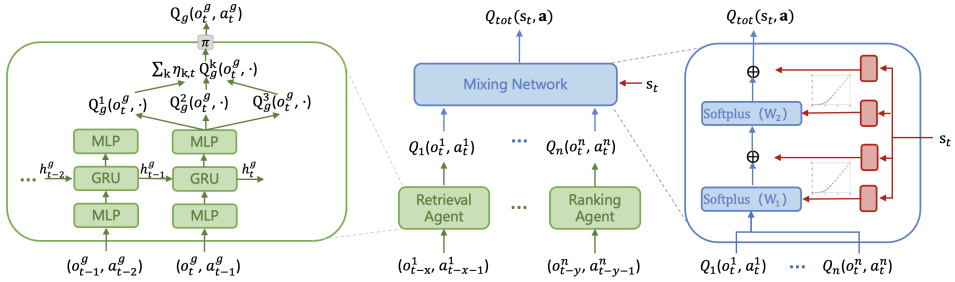


Figure 2: Adaptive Weighting Recurrent Q-Mixer (AWRQ-Mixer) Architecture in our framework.

Algorithm 1 Offline Training of AWRQ-Mixer

Input: Dataset $\mathcal{D}$, #iterations I, mini-batch size N, agent set $\mathcal{G}$, ensemble heads K, discount γ, hypernet h_ψ, target update frequency τ.

Initialize AWRQ $\{Q_g\}_{g\in\mathcal{G}}$ (each with K heads) and targets $Q_{g'}$ for each agent, initialize mixing network hypernet h_ψ.
for *iter* $= 1$ **to** I **do**
  Sample a mini-batch $\mathcal{B}$ of transitions $\{(s_t, o_t, a_t, R, s_{t+1})\}$ from $\mathcal{D}$.
  for t=1,...,T **do**
    $\forall t$: $w_t = \mathbb{V}\mathrm{ar}_{j\in\mathcal{A}_t}\big[\,\mathbb{E}[\,R \mid a_t = j\,]\,\big]$ // VGCA, **Eq.** (8)
    $r_t = w_t\, R$
  end for
  for $t = 1$ **to** T **do**
    AWRQ outputs K Q-values $Q_g^k(o_t, a_t)$.
    From **Eq.** (4) and **Eq.** (8):

$$\mathcal{L}_{k,t} = \big(r_t + \gamma\, Q_{g'}^k(o_{t+1}, a_{t+1})\big) - Q_g^k(o_t, a_t)\big)^2$$

    $\eta_{k,t} = \mathcal{L}_{k,t} / \sum_{k=1}^K \mathcal{L}_{k,t}$ // AW, **Eq.** (5)
    $Q_g = \sum_{k=1}^K \eta_{k,t}\, Q_g^k(o_t, a_t)$
    $\theta_g \leftarrow \arg\min_{\theta_g} \mathcal{L}_{k,t}$
    $\theta'_g \leftarrow \theta_g$ if iter $\bmod \tau = 0$
  end for
  $(W_1, b_1, W_2, b_2) \leftarrow h_\psi(s_t, \mathbf{a})$ with $W_1 = \mathrm{softplus}(\cdot)$, $W_2 = \mathrm{softplus}(\cdot)$ // SMC
  $Q_{\text{tot}} = \mathrm{ReLU}\big([Q_1, \ldots, Q_n]\, W_1 + b_1\big)\, W_2 + b_2$
  $\mathcal{L}_{\text{tot}} = (R + \gamma Q_{\text{tot}'}(s_{t+1}, \mathbf{a}') - Q_{\text{tot}}(s_t, \mathbf{a}))^2$
  $\psi \leftarrow \arg\min_\psi \mathcal{L}_{\text{tot}}$
end for

Output: Trained $\{\theta_g\}$ and ψ.

3.4 MPC-BASED REVENUE-COST BALANCER

The MPC-Based Revenue-Cost Balancer determines the optimal $\mathbf{a}^*$ that maximizes the expected reward in a given state s while keeping the computation resource utilization within the computation budget C_m. In prior works such as DCAF (Jiang et al., 2020) and SACA (Shunhui et al., 2021), binary search is employed to determine λ. Moreover, CRAS (Yang et al., 2021) and RL-MPCA (Zhou et al., 2023) utilize feedback-based methods to adjust λ for dynamic adaptation.

However, such feedback-based control inherently incurs one-step latency in responding to traffic fluctuations, leading to oscillatory resource misallocation when compensating for sudden traffic changes. To address this, we design a Model Predictive Control (MPC) (Mayne et al., 2000) framework that performs rolling-horizon optimization. By continuously predicting incoming traffic patterns and precomputing optimal λ trajectories, our approach enables proactive stabilization.

MPC optimizes computation allocation over a finite time horizon N by continuously solving an optimization problem that balances business performance, latency constraints, and hardware efficiency. The optimization process aims to ensure that the system computation resource utilization remains close to the computation budget and minimizes fluctuations to maintain stability. The optimization goal at time t is defined as:

$$J = \min_{\substack{\lambda_{t+i} \\ 0 \le i \le N-1}} \sum_{i=0}^{N} \alpha^i \left\| \hat{C}_{t+i} - C_m \right\|^2 + \begin{cases} 0, & \text{if } \hat{C}_{t+i} < C_m, \\ \beta \sum_{i=0}^{N} \alpha^i \left\| \hat{C}_{t+i} - \hat{C}_{t+i-1} \right\|^2, & \text{otherwise.} \end{cases} \tag{9}$$

$$\text{s.t.} \quad \hat{C}_{t+i+1} = g(\hat{C}_{t+i}, s_{t+i}, \lambda_{t+i}) \quad 0 \le i \le N-1 \tag{10}$$

Here $\alpha^i \in (0, 1]$ is a decay weighting factor that reduces the weight of longer-term predictions, thus mitigating long-horizon errors. β is an oscillation damping factor penalizing abrupt computation resource utilization changes. $g(\cdot)$ is a learned system model mapping computation resource states $\hat{C}_{t+i}$, environment states s_{t+i}, and the revenue-cost balancer λ_{t+i} to future CPU loads. Specifically, $g(\cdot)$ is parameterized as a Deep & Cross Network (DCN) (Wang et al., 2017) trained on historical traffic logs to predict future computation loads. Offline evaluations show that it achieves high prediction accuracy with a Normalized Mean Square Error (NMSE) of 3.403×10^{-4} and an Average Percentage Error (APE) of 0.104.

By optimizing J, this objective ensures that computation resource utilization near C_m while reducing fluctuations. Only the first element of the computed optimal sequence $\{\lambda_t^*, ..., \lambda_{t+N-1}^*\}$ is applied at time t, ensuring proactive and stable resource allocation.

4 EXPERIMENTS

4.1 OFFLINE EXPERIMENTS

Our evaluation focuses on two core modules: the AWRQ-Mixer and the MPC-based Revenue–Cost Balancer. We evaluate the AWRQ-Mixer using both model metrics and simulated revenue derived from real-world logs, demonstrating its prediction accuracy and revenue uplift. For the MPC-based Balancer, we introduce utilization and overutilization rates as evaluation metrics to quantify its scheduling capability.

4.1.1 AWRQ-MIXER EXPERIMENT RESULTS

Experimental Setup. We evaluate the AWRQ-Mixer using multiple dimensions. To quantify ranking quality, we use **Spearman's Rank Correlation (r_s)** to measure the monotonicity between the model's predicted rankings and the true rankings. Evaluating new models in a live environment is often risky and resource-intensive. To enable fair and controlled comparisons under consistent computation budgets, we simulate revenue through a four-step protocol (detailed in Ap-

pendix E). We define the primary metric, **Relative Return Percentage (Return%)**, as the ratio of total expected revenue from the model's actions (Q_{model}) to the theoretical maximum (Q_{GT}): Return% $= \frac{\sum Q_{model}}{\sum Q_{GT}} \times 100\%$ Finally, to quantify stability, we report **Convergence** (environment steps to reach 95% validation return) and **Gradient-variance** (variance of the joint-Q gradient L2-norm). Specific simulation protocols and detailed metric formulations are provided in Appendix E.

Hyperparameter Analysis We optimized AWRQ-Mixer hyperparameters via grid search (Bergstra & Bengio, 2012); detailed settings are listed in Appendix G.

Baselines We compare AWRQ-Mixer with representative single-agent and cooperative multi-agent RL baselines. **1) Single-agent:** DQN(Mnih et al., 2015), DRQN(Hausknecht & Stone, 2015), DDQN(van Hasselt et al., 2016), Averaged Ensemble-DQN(Anschel et al., 2017), REM(Agarwal et al., 2020). **2) Multi-agent:** VDN(Sunehag et al., 2018), QMIX (Rashid et al., 2020b), Weighted QMIX (Rashid et al., 2020a). Details are provided in Appendix D.

Experiment Results We comprehensively analyze the real-world logs from the display advertising system. As shown in Table 1, AWRQ-Mixer achieves significant improvements on both evaluation metrics. Compared to REM, the single-agent model used in RL-MPCA (Zhou et al., 2023), AWRQ-Mixer boosts r_s by 3.4% and Return% by 8.0%. Multi-agent methods generally outperform single-agent ones, underlining the benefits of collaborative modeling.

Table 1: Comparison of offline evaluation results across different baseline models.

Method	r_s	Return (%)
Single-Agent		
DQN	0.859 ± 0.025	82.59 ± 4.22
DDQN	0.860 ± 0.019	85.46 ± 3.18
DRQN	0.862 ± 0.023	86.82 ± 3.67
Avg. Ens.-DQN	0.870 ± 0.015	87.48 ± 2.79
REM (RL-MPCA)	0.881 ± 0.014	89.47 ± 2.63
Multi-Agent		
VDN	0.896 ± 0.018	95.10 ± 1.94
Weighted QMIX	0.900 ± 0.015	95.65 ± 1.68
QMIX	0.902 ± 0.017	96.00 ± 1.63
AWRQ-Mixer (Ours)	$\mathbf{0.911 \pm 0.009}$	$\mathbf{97.26 \pm 1.01}$

Ablation Study We assess the contributions of three key components in AWRQ-Mixer by removing each one separately and comparing these variants to the full model. Table 2 presents the results. The full model attains the highest r_s (0.911) and Return% (97.26%), while converging in the fewest environment steps (1.1 M) and exhibiting the lowest gradient variance (0.18).

Table 2: Ablation study results of core components (AW, SMC, VGCA). Metrics: Spearman's r_s, Return%, Convergence steps (Millions), and Gradient Variance.

Variant	r_s	Return%	Convergence	Gradient-variance
AWRQ-Mixer	0.911 ± 0.009	97.26 ± 1.01	1.1 ± 0.10	0.18 ± 0.05
w/o AW	0.910 ± 0.010	97.17 ± 1.10	1.4 ± 0.20	0.54 ± 0.19
w/o SMC	0.908 ± 0.009	96.92 ± 1.18	1.3 ± 0.15	0.41 ± 0.12
w/o VGCA	0.906 ± 0.011	96.71 ± 1.37	1.7 ± 0.20	0.27 ± 0.08

1) AW: Removing AW decreases r_s and Return% only marginally (–0.1% and –0.09%), while the gradient variance triples (0.54 vs 0.18) and the number of steps to reach the same validation loss increases by 27%. The larger variance indicates that, without the TD-error–driven head re-weighting, noisy or mis-calibrated heads exert disproportionate influence, producing erratic updates and slowing convergence. **2) SMC:** Replacing Softplus with an absolute-value operator leads to higher gradient variance (0.41) and a 20% slower convergence, accompanied by a further drop in both r_s and Return%. Although absolute value enforces monotonicity, its discontinuity at zero amplifies gradient fluctuations around the decision boundary. The measured variance confirms this analytic expectation and explains the observed degradation. **3) VGCA:** Eliminating VGCA yields the largest decline in ranking quality (–0.51% r_s) and offline return (–0.55%), together with the slowest convergence

(1.7 M steps). By reallocating TD-errors according to action variance, VGCA accelerates credit propagation in sparse-reward regions. Without VGCA, the agents require more samples to achieve the same validation criterion, even though the raw gradient variance remains moderate (0.27).

Across all metrics, each component contributes additively. Their combination yields a consistently more stable, accurate, and faster-converging learner.

4.1.2 MPC-BASED REVENUE-COST BALANCER EXPERIMENT RESULTS

Evaluation Metrics We introduce two complementary metrics that capture distinct aspects of computation allocation: utilization rate μ and overutilization rate ν.

- Utilization Rate (μ): This metric quantifies the effective usage of available resources: $\mu = \frac{1}{T}\sum_{i=1}^{T}\frac{\min(\hat{C}_t, C_m)}{C_m}$. A higher μ indicates better computation resource utilization within limits.
- Overutilization Rate (ν): This metric measures the risk of exceeding the computation budget: $\nu = \frac{1}{T}\sum_{i=1}^{T}\frac{\max(\hat{C}_t, C_m) - C_m}{C_m}$. A lower ν indicates fewer stability violations and reduced risk of service degradation.

Balancing μ and ν is crucial for ensuring efficiency and stability. While increasing μ raises the risk of exceeding limits, reducing ν leads to conservative resource utilization.

Offline Experiment Results We compared the performance of feedback-based and MPC-based revenue-cost balancers. Table 3 shows that the MPC-based approach not only improves overall resource usage but also substantially lowers the risk of exceeding the computation budget.

Table 3: Feedback vs. MPC Revenue–Cost Balancer metrics (%).

Method	Utilization Rate	Overutilization Rate
Feedback	92.33 ± 0.40	2.91 ± 0.43
MPC-based	$\mathbf{95.29 \pm 0.23}$	0.64 ± 0.10

Hyperparameter Analysis Based on the sensitivity analysis detailed in Appendix F, we set $\alpha = 0.4$, $\beta = 8$, and $N = 10$, which empirically offered the best trade-off between predictive accuracy and runtime efficiency.

4.2 ONLINE A/B TEST RESULTS

We conducted a four-week online A/B test comparing five strategies: a static method, DCAF (Jiang et al., 2020), RL-MPCA (Zhou et al., 2023), our framework with a feedback-based Revenue-Cost Balancer, and our framework with an MPC-based balancer. The static approach uses fixed allocation rules through stress testing and practical experience, with predefined downgrades to manage traffic spikes. Table 4 reports impressions, clicks, revenue, gross merchandise volume (GMV), return on investment (ROI), click-through rate (CTR), and cost per mille (CPM), with revenue and GMV as our primary metrics. ROI is defined as $\mathrm{GMV}/\mathrm{Spend}$, where Spend denotes advertiser spend (i.e., the platform's revenue). Our near-real-time deployment adds virtually no additional latency. The decision-making process introduces only 7 ms of P99 latency and a negligible 0.2% increase in inference cost.

Table 4: Online A/B test results over Static (%). All metrics are reported as *Mean* $\pm$ *Std*.

Method	Revenue	GMV	Impr.	Clicks	ROI	CTR
Static	0.00	0.00	0.00	0.00	0.00	0.00
DCAF	+3.67 $\pm$ 0.20	+6.16 $\pm$ 3.68	+4.92 $\pm$ 0.04	+5.38 $\pm$ 0.09	+2.40 $\pm$ 3.69	+0.69 $\pm$ 0.10
RL-MPCA	+12.16 $\pm$ 0.28	+13.78 $\pm$ 4.03	+9.07 $\pm$ 0.03	+15.37 $\pm$ 0.10	+0.55 $\pm$ 4.04	+4.85 $\pm$ 0.10
Ours-FB	+14.93 $\pm$ 0.43	+15.65 $\pm$ 5.39	+11.67 $\pm$ 0.04	+17.79 $\pm$ 0.13	+0.64 $\pm$ 5.41	+5.58 $\pm$ 0.14
Ours-MPC	**+16.67** $\pm$ 0.24	**+18.18** $\pm$ 3.95	+14.24 $\pm$ 0.03	+19.51 $\pm$ 0.07	+1.29 $\pm$ 3.96	+5.22 $\pm$ 0.08

Our framework achieved statistically significant improvements across all key metrics while operating within existing computation resource constraints. After the four-week A/B test, our framework was rolled out to 100% of production traffic, where it now processes hundreds of billions of requests each day and has since supported multiple large-scale sales events. Its stability has also mitigated the need for continuous on-call support.

5 CONCLUSION

We propose a framework to maximize business revenue in large-scale recommender systems under strict computation constraints. By modeling recommendation stages as cooperative agents and integrating Centralized Training with Decentralized Execution, our framework effectively captures cross-stage dependencies while preserving independent decision-making. Additionally, we introduce an MPC-based revenue-cost balancer that proactively adjusts resource allocation, ensuring system stability under dynamic traffic conditions. Our extensive offline experiments and large-scale online deployment demonstrate that our framework significantly improves business revenue, achieving a 16.67% revenue increase with no additional computation resource. Future work may focus on enhancing model capabilities, expanding the action space, and exploring cross-domain applications.

REFERENCES

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 104–114. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/agarwal20c.html.

Kiam Heong Ang, G. Chong, and Yun Li. Pid control system analysis, design, and technology. *IEEE Transactions on Control Systems Technology*, 13(4):559–576, 2005. doi: 10.1109/TCST.2005.847331.

Oron Anschel, Nir Baram, and Nahum Shimkin. Averaged-dqn: variance reduction and stabilization for deep reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 176–185. JMLR.org, 2017.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(null):281–305, February 2012. ISSN 1532-4435.

Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, Inc., 1st edition, 2016. ISBN 149192912X.

Vanessa Cai, Pradeep Prabakar, Manuel Serrano Rebuelta, Lucas Rosen, Federico Monti, Katarzyna Janocha, Tomo Lazovich, Jeetu Raj, Yedendra Shrinivasan, Hao Li, and Thomas Markovich. Twerc: High performance ensembled candidate generation for ads recommendation at twitter. In Abraham Bagherjeiran, Nemanja Djuric, Kuang-Chih Lee, Linsey Pang, Vladan Radosavljevic, and Suju Rajan (eds.), *Proceedings of the Workshop on Data Mining for Online Advertising (AdKDD 2023)*, volume 3556 of *CEUR Workshop Proceedings*, Aachen, Germany, 2023. CEUR-WS.org. URL https://ceur-ws.org/Vol-3556/adkdd23-cai-twerc-ceur-paper.pdf.

Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, pp. 7–10, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450347952. doi: 10.1145/2988450.2988454. URL https://doi.org/10.1145/2988450.2988454.

Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pp. 191–198, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450340359. doi: 10.1145/2959100.2959190. URL https://doi.org/10.1145/2959100.2959190.

Yang Deng, Yaliang Li, Fei Sun, Bolin Ding, and Wai Lam. Unified conversational recommendation policy learning via graph-based reinforcement learning. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*,

SIGIR '21, pp. 1431–1441, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380379. doi: 10.1145/3404835.3462913. URL https://doi.org/10.1145/3404835.3462913.

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*, 2015.

Dom Huh and Prasant Mohapatra. Multi-agent reinforcement learning: A comprehensive survey, 2024. URL https://arxiv.org/abs/2312.10256.

Biye Jiang, Pengye Zhang, Rihan Chen, Binding Dai, Xinchen Luo, Yin Yang, Guan Wang, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. Dcaf: A dynamic computation allocation framework for online serving system. In *Proceedings of the 2nd Workshop on Deep Learning Practice for High-Dimensional Sparse Data with KDD 2020 (DLP-KDD'20)*, San Diego, CA, USA, August 2020. Association for Computing Machinery. Best Paper Runner-Up.

Junqi Jin, Chengru Song, Han Li, Kun Gai, Jun Wang, and Weinan Zhang. Real-time bidding with multi-agent reinforcement learning in display advertising. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, pp. 2193–2201. ACM, October 2018. doi: 10.1145/3269206.3272021. URL http://dx.doi.org/10.1145/3269206.3272021.

Yehuda Koren. Collaborative filtering with temporal dynamics. volume 53, pp. 447–456, 06 2009. doi: 10.1145/1557019.1557072.

Shichen Liu, Fei Xiao, Wenwu Ou, and Luo Si. Cascade ranking for operational e-commerce search. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17. ACM, August 2017. doi: 10.1145/3097983.3098011. URL http://dx.doi.org/10.1145/3097983.3098011.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6382–6393, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Xingyu Lu, Zhining Liu, Yanchu Guan, Hongxuan Zhang, Chenyi Zhuang, Wenqi Ma, Yize Tan, Jinjie Gu, and Guannan Zhang. Greenflow: a computation allocation framework for building environmentally sound recommendation system. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, IJCAI '23, 2023. ISBN 978-1-956792-03-4. doi: 10.24963/ijcai.2023/677. URL https://doi.org/10.24963/ijcai.2023/677.

Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H. Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pp. 1930–1939, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3220007. URL https://doi.org/10.1145/3219819.3220007.

D.Q. Mayne, J.B. Rawlings, C.V. Rao, and P.O.M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000. ISSN 0005-1098. doi: https://doi.org/10.1016/S0005-1098(99)00214-9. URL https://www.sciencedirect.com/science/article/pii/S0005109899002149.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Nicola Neophytou, Bhaskar Mitra, and Catherine Stinson. Revisiting popularity and demographic biases in recommender evaluation and effectiveness. In *Advances in Information Retrieval: 44th European Conference on IR Research, ECIR 2022, Stavanger, Norway, April 10–14, 2022, Proceedings, Part I*, pp. 641–654, Berlin, Heidelberg, 2022. Springer-Verlag. ISBN 978-3-030-99735-9. doi: 10.1007/978-3-030-99736-6_43. URL https://doi.org/10.1007/978-3-030-99736-6_43.

Zhou Qin, Kai Yuan, Pratik Lahiri, and Wenyang Liu. Cooperative multi-agent deep reinforcement learning in content ranking optimization. In Surya Kallumadi, Yubin Kim, Tracy Holloway King, Maarten de Rijke, and Vamsi Salaka (eds.), *Proceedings of the ACM SIGIR Workshop on eCommerce 2024 co-located with the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2024), Washington D.C., USA, July 18, 2024*, volume 3843 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2024a. URL https://ceur-ws.org/Vol-3843/paper_15.pdf.

Zhou Qin, Kai Yuan, Pratik Lahiri, and Wenyang Liu. Cooperative multi-agent deep reinforcement learning in content ranking optimization, 2024b. URL https://arxiv.org/abs/2408.04251.

Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted qmix: expanding monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020a. Curran Associates Inc. ISBN 9781713829546.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(178):1–51, 2020b. URL http://jmlr.org/papers/v21/20-081.html.

Shunhui, Jiahong, Songwei, Guoliang, Qianlong, and Lebin. Single-action computation allocation. https://tech.meituan.com/2021/06/17/waimai-ai-advertisement.html, 2021.

Santhosh Kumar Somarapu. Autoscaling strategies for stateful stream operators under bursty workloads. *International Journal of Communication Networks and Information Security (IJCNIS)*, 16 (1):428–448, Jan. 2024. URL https://ijcnis.org/index.php/ijcnis/article/view/8368.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, pp. 2085–2087, Richland, SC, 2018. International Foundation for Autonomous Agents and Multiagent Systems.

Farah Tawfiq, Abdul Monem Rahma, and Hala Abdul wahab. Recommendation systems for e-commerce systems an overview. *Journal of Physics: Conference Series*, 1897:012024, 05 2021. doi: 10.1088/1742-6596/1897/1/012024.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), Mar. 2016. doi: 10.1609/aaai.v30i1.10295. URL https://ojs.aaai.org/index.php/AAAI/article/view/10295.

Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*, ADKDD'17, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450351942. doi: 10.1145/3124749.3124754. URL https://doi.org/10.1145/3124749.3124754.

Xun Yang, Yunli Wang, Cheng Chen, Qing Tan, Chuan Yu, Jian Xu, and Xiaoqiang Zhu. Computation resource allocation solution in recommender systems, 2021. URL https://arxiv.org/abs/2103.02259.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Xiangyu Zhao, Xudong Zheng, Xiwang Yang, Xiaobing Liu, and Jiliang Tang. Jointly learning to recommend and advertise. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pp. 3319–3327, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3403384. URL https://doi.org/10.1145/3394486.3403384.

Xiangyu Zhao, Changsheng Gu, Haoshenglun Zhang, Xiwang Yang, Xiaobing Liu, Jiliang Tang, and Hui Liu. Dear: Deep reinforcement learning for online advertising impression in recommender systems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(1):750–758, May 2021. doi: 10.1609/aaai.v35i1.16156. URL https://ojs.aaai.org/index.php/AAAI/article/view/16156.

Zhishan Zhao, Jingyue Gao, Yu Zhang, Shuguang Han, Siyuan Lou, Xiang-Rong Sheng, Zhe Wang, Han Zhu, Yuning Jiang, Jian Xu, and Bo Zheng. Copr: Consistency-oriented pre-ranking for online advertising. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, CIKM '23, pp. 4974–4980, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701245. doi: 10.1145/3583780.3615465. URL https://doi.org/10.1145/3583780.3615465.

Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pp. 1059–1068, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3219823. URL https://doi.org/10.1145/3219819.3219823.

Jiahong Zhou, Shunhui Mao, Guoliang Yang, Bo Tang, Qianlong Xie, Lebin Lin, Xingxing Wang, and Dong Wang. Rl-mpca: A reinforcement learning based multi-phase computation allocation approach for recommender systems. In *Proceedings of the ACM Web Conference 2023*, WWW '23, pp. 3214–3224. ACM, April 2023. doi: 10.1145/3543507.3583313. URL http://dx.doi.org/10.1145/3543507.3583313.

APPENDIX

A BACKGROUND ON MULTI-STAGE RECOMMENDER SYSTEMS

For completeness, we illustrate the general architecture of the industrial multi-stage recommender system in Figure 3. The system typically filters relevant items from a large candidate pool through a funnel of stages: Retrieval, Pre-Ranking, and Ranking, before displaying the final list to the user.

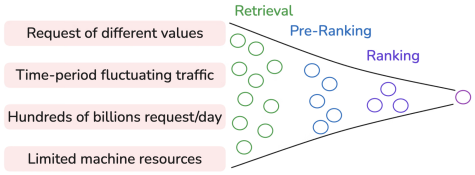


Figure 3: Overview of the general architecture of recommender systems, highlighting the key stages and components involved in processing user requests.

B LAGRANGIAN RELAXATION DETAILS

To satisfy the budget constraint in Eqs. (**??**)–(**??**) while maximizing total revenue in Eq. (1), we adopt a Lagrangian relaxation approach:

$$\mathcal{L} = \sum_{i=1}^{M} \sum_{\mathbf{a}\in\mathcal{A}} x_{i,\mathbf{a}}\, Q(s_t, \mathbf{a}) - \lambda_t \sum_{i=1}^{M} \sum_{\mathbf{a}\in\mathcal{A}} (x_{i,\mathbf{a}}\, C(s_t, \mathbf{a}) - C_m) \tag{11}$$

Where Revenue-Cost Balancer (λ_t) serves as a trade-off parameter balancing business revenue and computation cost. The constant term $\lambda_t C_m$ can be omitted in the maximization as it does not affect the optimization. Substituting the Lagrangian utility in Eq. (11) back into the original objective of Eq. (1) absorbs the cost constraint into the multiplier λ_t. The global problem becomes:

$$\max_{x_{i,\mathbf{a}}} \sum_{i=1}^{M} \sum_{\mathbf{a}\in\mathcal{A}} x_{i,\mathbf{a}} \big(Q(s_t, \mathbf{a}) - \lambda_t C(s_t, \mathbf{a})\big) \tag{12}$$

Because $x_{i,\mathbf{a}}$ is one-hot for each request, the double sum decomposes into independent per-request arg-max operations. Following the approach in (Zhou et al., 2023), each user request effectively solves a local decision subproblem:

$$\mathbf{a}^* = \arg\max_{\mathbf{a}\in\mathcal{A}} \left(Q(s_t, \mathbf{a}) - \lambda C(s_t, \mathbf{a})\right) \tag{13}$$

which is Eq. (3) in the main text.

C AUTOBUCKET TESTBENCH

Accurately estimating computation costs is essential for resource allocation in large-scale recommender systems. However, real-world cost labels are scarce, and the variability in action results across multiple stages complicates direct cost measurement at the request level. We propose a two-phase approach that predicts action results and derives their associated costs via cost mapping. This method leverages the observation that computation costs remain consistent for identical action results, even when the underlying requests differ, since computation cost is primarily dictated by code logic, model complexity, and retrieval processes.

Figure 4 presents the core architecture of our AutoBucket TestBench. Traffic logs from multiple sources are processed and subjected to simulated load testing, during which computation resource

utilization is recorded. The resulting labeled samples (e.g., queue length vs. cost) are aggregated and used to fit a regression model, producing a cost-mapping function. Combined with predicted action outcomes, this module enables accurate computation cost estimations.

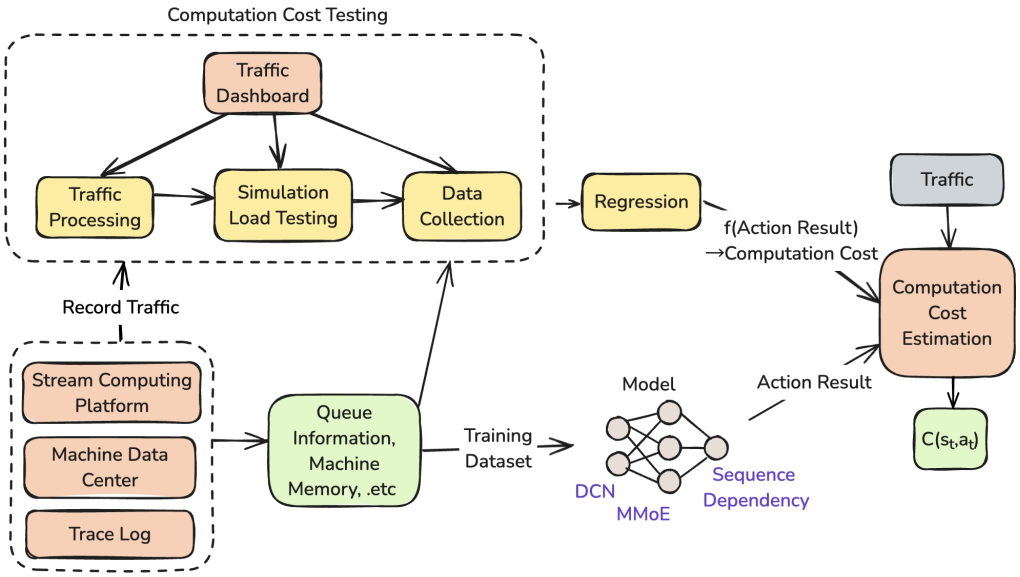


Figure 4: AutoBucket TestBench enables multi-stage computation cost estimation through traffic simulation, regression analysis, and sequence-aware modeling.

Action Results Prediction. Given user and contextual features, we employ a DCN (Wang et al., 2017)+MMoE (Ma et al., 2018) hybrid architecture to predict action results. The Deep & Cross Network (DCN) captures nonlinear feature interactions, while the Multi-gate Mixture of Experts (MMoE) layer enables parameter sharing among predictions. This multi-task learning framework operates within the $S \times A$ space, leveraging abundant logged interaction data for supervised training.

Computation Cost Estimation. With predicted action results, we estimate the corresponding computation cost via a cost mapping.

- **Elastic Queue**. For actions determining queue truncation lengths, we use empirical tests under varied queue lengths to measure computation cost, and then fit a regression model.
  1. **AutoBucket**. Simulation traffic is grouped into buckets based on request value and queue length to capture the variability in computation cost across diverse traffic conditions.
  2. **TestBench**. The TestBench processes requests in each bucket at a fixed queries-per-second (QPS). The TestBench is deployed on n machines, each with N_{cores} computation resource cores, and the computation resource utilization $p\%$ is recorded during these tests. The computation cost per request for each bucket is computed as:

$$\text{Computation Cost per Bucket} = \frac{p\% \times n \times N_{\text{cores}}}{QPS} \tag{14}$$

     A monotonic regression model is then used to fit the relationship between queue lengths and computation cost.
- **Elastic Model**. For actions that involve switching between different models, computation costs are derived from independent measurements of each switch configuration. Similar to Elastic Queue, Elastic Models are tested under controlled conditions to measure their computation cost.
- **Elastic Channel**. The computation cost is derived as the cumulative sum of the computation costs across all selected actions.

Error Analysis. To address potential concerns regarding the reliability of the cost estimator, we conducted a rigorous offline error analysis on the AutoBucket TestBench across various truncation actions. The results demonstrate a strong monotonic relationship between the estimated and actual computation costs, with Spearman's rank correlation ranging from 0.726 to 0.908. Furthermore, the Mean Absolute Percentage Error (MAPE) remains exceptionally low (MAPE $\leq 0.2\%$) across all evaluated actions. This minimal estimation error confirms that the AutoBucket TestBench provides a highly accurate and robust cost foundation for the downstream decision-making pipeline.

D BASELINES

Baselines We compare AWRQ-Mixer against two categories of baselines.

1) Single-Agent Methods: We include **DQN** (Mnih et al., 2015), which employs a single deep network to approximate Q-values in high-dimensional state spaces. We also consider its variants: **DRQN** (Hausknecht & Stone, 2015) extends DQN with a recurrent mechanism to handle partial observability; **DDQN** (van Hasselt et al., 2016) decouples action selection from evaluation to mitigate Q-value overestimation; **Averaged Ensemble-DQN** (Anschel et al., 2017) aggregates multiple Q-networks to reduce variance and enhance stability; and **REM** (Agarwal et al., 2020) combines Q-networks through random convex mixtures for richer exploration.

2) Multi-Agent Methods: We compare against **VDN** (Sunehag et al., 2018), which decomposes the global Q-value into a sum of per-agent Q-values for cooperative policies. We also compare with **QMIX** (Rashid et al., 2020b), which employs a mixing network to ensure monotonic relationships among individual Q-values to improve coordination, and **Weighted QMIX** (Rashid et al., 2020a), which uses dynamic weighting in QMIX to emphasize heterogeneous agent contributions.

E DETAILED EXPERIMENTAL SETUP

E.1 MODEL EVALUATION METRICS

To quantify the monotonic relationship between the model's predicted rankings and the true rankings, we use Spearman's Rank Correlation (r_s): $r_s = \frac{\mathrm{cov}(\mathrm{R}(X),\mathrm{R}(Y))}{\sigma_{\mathrm{R}(X)}\sigma_{\mathrm{R}(Y)}}$ where $\mathrm{cov}(\mathrm{R}(X),\mathrm{R}(Y))$ denotes the covariance between the ranks of variables X and Y, and $\sigma_{\mathrm{rank}(X)}$, $\sigma_{\mathrm{rank}(Y)}$ are the corresponding standard deviations. Higher values of r_s indicate stronger positive correlations, reflecting high agreement between the predicted and actual orderings.

E.2 TRAINING STABILITY METRICS

To quantify optimisation stability we report two extra metrics:

- Convergence. The number of environment steps (in millions) required for a model to reach 95% of its final validation Return% (averaged over 5 random seeds). A smaller value indicates faster sample efficiency.
- Gradient-variance. $g_t = \|\nabla_\theta \mathcal{L}_t\|_2$ be the L2-norm of the joint-Q network gradient. For a sliding window of 1000 updates, we compute the variance and then average across all windows and seeds. Lower variance implies smoother gradient flow and more stable learning.

E.3 REVENUE SIMULATION PROTOCOL

Evaluating new models in a live environment is often risky and resource-intensive. To enable fair and controlled comparisons under consistent computation budgets, we simulate revenue in four steps:

1. **Ground Truth Estimation:** We train an ensemble model on historical data, including train and test data, to approximate true revenue (achieving $r_s = 0.95$).
2. **Uniform Computation Cost:** We derive action distributions from the test data that reflect the total computation cost, and use them as the action quota.

3. **Action Allocation:** We train the baseline models on the training dataset, and compute the action values $\hat{Q}(s, \mathbf{a})$ for all action-state pairs in the test dataset. Sort all $(s, \mathbf{a})$ pairs by $\hat{Q}$, then iteratively assign the highest-valued action without exceeding the action quota.
4. **Revenue Evaluation:** We compute the total expected revenue using the ensemble model and assess performance using Relative Return Percentage: $\text{Return}\% = (\sum Q_{\text{model}} / \sum Q_{\text{GT}}) \times 100\%$. We verified that Return% and r_s correlate strongly (r = 0.93) and both have the same ordering as online Revenue, which shows the offline metrics have strong validity

F Hyperparameters Sensitivity Analysis

F.1 AWRQ-Mixer Hyperparameters

We conduct an offline sensitivity analysis for two key hyperparameters in AWRQ-Mixer: discount factor γ (see Table 5) and the number of ensemble heads K (see Table 6). Each configuration is evaluated over 5 random seeds.

- Discount Factor γ. We selected a γ value of 0.9. In reinforcement learning, γ represents the trade-off between long-term and immediate rewards, determining how much importance the model places on future rewards. From Table 5, we observe that when γ is set to 0.90, the model exhibits the highest stability and performance. In contrast, other values such as 0.70 and 0.99, show slightly lower results. Therefore, we ultimately chose 0.9 for γ to ensure a balance of strong performance and system stability.

Table 5: Discount Factor γ Sensitivity

γ	r_s	Return%
0.70	0.9096($\pm$0.0076)	97.18($\pm$0.97)
0.80	0.9107($\pm$0.0083)	97.22($\pm$1.01)
0.90	0.9112($\pm$0.0086)	97.26($\pm$1.01)
0.99	0.9105($\pm$0.0109)	97.21($\pm$1.05)

- Number of ensemble heads K. Table 6 illustrates the effects of different ensemble head numbers on performance. As the value of K increases, we see a gradual improvement in both r_s and Return%. Specifically, when K is set to 200, the r_s value reaches 0.9120, with a Return% of 97.30. However, while K = 200 yields the best performance, increasing the number of ensemble heads significantly boosts computational complexity. After considering the trade-off between computational resources and model effectiveness, we selected K = 20, which provided nearly optimal results while minimizing unnecessary computational overhead.

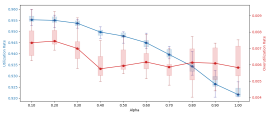
Table 6: Ensemble Head K Sensitivity

K	r_s	Return%
1	0.9065($\pm$0.0095)	96.78($\pm$1.11)
5	0.9081($\pm$0.0089)	96.95($\pm$1.02)
20	0.9112($\pm$0.0086)	97.26($\pm$1.01)
100	0.9118($\pm$0.0087)	97.29($\pm$1.01)
200	0.9120($\pm$0.0085)	97.30($\pm$1.00)

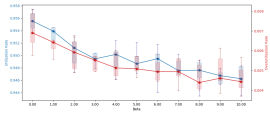
F.2 MPC-based Revenue-Cost Balancer Hyperparameters

We investigate three key hyperparameters: decay-weighting factor α, oscillation damping factor β, and prediction horizon N.

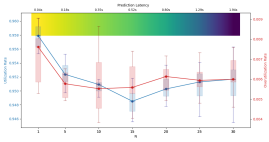
- α. A small α prioritizes current returns, which can lead to more aggressive resource utilization but can intensify fluctuations. Conversely, a large α may cause underutilization by overemphasizing future risks. As shown in Figure 5a, the overutilization rate stabilizes around $\alpha = 0.4$, indicating balanced short/long-term trade-offs. We choose $\alpha = 0.4$ as the parameter value.



(a) Decay-Weighting Factor α



(b) Oscillation Damping Factor β



(c) Prediction Horizon N

Figure 5: Hyperparameter analysis results for α, β and N.

- β. β is employed to limit the fluctuation of computation resource utilization. When β is too high, the system becomes overly conservative, lowering utilization. Conversely, an extremely small β can result in unstable spikes. Based on the results in Figure 5b, we choose $\beta = 8$ to achieve a trade-off between resource utilization and stability.
- **N**. While a larger N often improves long-term optimization, it simultaneously increases forecasting errors and latency overhead. Conversely, a smaller N may fail to capture future dynamics. Figure 5c indicates that $N = 10$ provides an effective balance between predictive accuracy and runtime efficiency.

G HYPERPARAMETERS

The following Table 7 lists the hyperparameters we used in experiments.

Table 7: Hyperparameter settings for our framework.

Hyperparameters	Value
AWRQ-Mixer	
Learning rate	0.01
GRU hidden size	256
Discount factor γ	0.9
Target-network update frequency τ	100
Ensemble size K	20
ε-greedy exploration rate	0.05
Size of hidden layer in the network	[512, 256]
Optimizer	Adam
Dropout rate	0.2
Weight initializer	glorot uniform
Batch size	2048
MPC-based Revenue-Cost Balancer	
Decay-Weighting Factor α	0.4
Oscillation Damping Factor β	8
Prediction Horizon N	10

H ONLINE SERVING OF OUR FRAMEWORK

Algorithm 2 shows our framework's online serving process for a given request.

Algorithm 2 Online Serving of our framework

Input: Trained networks $\{Q_g\}$, cost estimator $C(\cdot)$, MPC Revenue-Cost Balancer that outputs λ_t

if periodic update moment, $\lambda_t \leftarrow$ MPC Revenue-Cost Balancer
for each incoming request **do**
  Observe state s_t
  $\mathbf{a}^* \leftarrow \arg\max_{\mathbf{a}\in\mathcal{A}} \left(Q(s_t, \mathbf{a}) - \lambda C(s_t, \mathbf{a})\right)$
  Execute joint action $\mathbf{a}^*$
end for