MoTiAC: Multi-objective Actor-Critics for Real-time Bidding in Display Advertising

Chaoqi Yang
Shanghai Jiao Tong University
ycqsjtu@gmail.com

Junwei Lu
Shanghai Jiao Tong University
luke_13@sjtu.edu.cn

Xiaofeng Gao
Shanghai Jiao Tong University
gao-xf@cs.sjtu.edu.cn

Haishan Liu, Qiong Chen
Tencent Ads
{haishanliu, evechen}@tencent.com

Gongshen Liu
Shanghai Jiao Tong University
lgshen@sjtu.edu.cn

Guihai Chen
Shanghai Jiao Tong University
gchen@cs.sjtu.edu.cn

ABSTRACT

Online Real-Time Bidding (RTB) is known as a complex auction game where advertising inventory is bought on a per-impression basis. Large online advertising platforms, such as the one run by Tencent, connect advertisers to a wide portfolio of advertising channels. In doing so, the platform needs to optimize user response rate, advertisers' Return on Investment (ROI), Gross Merchandise Value (GMV) and other influential Key Performance Indicators (KPIs). The trade-off among various competing goals needs to be balanced on a massive scale. To tackle this problem, we propose a bidding algorithm based on Multi-objective Actor-Critics (in short, MoTiAC), as an extension to the Asynchronous Advantage Actor-Critic (A3C). In MoTiAC, agents update the global network asynchronously, which guides the network policy from multiple perspectives according to the respective objectives of the agents. To our best knowledge, this is the first objective-specialized actor-critic framework proposed for the bid optimization problem. We also report implementation details to achieve the parallelization of the model. We demonstrate the effectiveness of our method on real-world data. Experiments show that the model outperforms state-of-the-art bidding strategies.

CCS CONCEPTS

• **Information systems** → *Display advertising*; • **Computing methodologies** → *Multi-agent reinforcement learning*.

KEYWORDS

Real-Time Bidding, Reinforcement Learning, Display Advertising, Multi-objective Optimization

1 INTRODUCTION

The rapid development of Internet and smart devices has created a decent environment for advertisement industry. As a billion dollar business, online advertising has gain continuous attention in the past few decades, during which, Real-Time Bidding (RTB) becomes the most promising theme in this trend [43].

Bidding System. *Online users*, *advertisers* and *ad platforms* constitute the main players in real-time bidding. A typical RTB setup is illustrated in Fig. 1, which consists of publishers, supply-side platforms (SSP), data management platforms (DMP), ad exchange (ADX), and demand-side platforms (DSP) [38]. In one bidding round, when an online user's browsing activity triggers an ad request, the SSP sends the request to the DSP through the ADX, where eligible ads compete for the *impression*. Serving as a bidding agent, the DSP

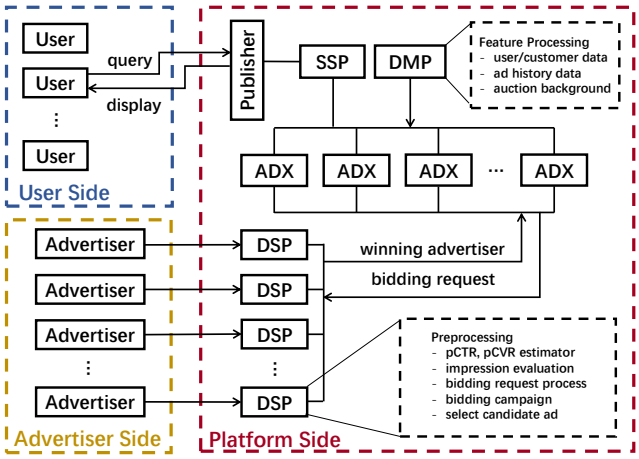

Figure 1: RTB System

represents advertisers to come up with an optimal bid and transmit the bid back to the ADX (e.g. usually within less than 200ms [43]), where the winner is selected for the Publisher to display and charged with Generalized Second Price (GSP) that is proven to encourage truthful auction [37].

In this RTB system, *bidding optimization* by DSPs is regraded as one of the most essential and profitable problem [45]. Unlike Sponsored Search (SS) [46], where advertisers make keyword-level bidding decision, DSPs in RTB setting need to calculate the optimal impression-level bid based on the user/customer information (e.g. age, gender, purchasing behavior, etc.), the ad information (e.g. content, historical engagement, budget, etc.), and the auction context (e.g. volume, bid landscape, time, etc.) in each single auction [45].

Real-world Challenges. To conduct real-time bidding, two fundamental challenges need to be addressed. In [38, 45, 47], researchers make a strong assumption that the data distribution is stationary over time. However, the sequence of user queries (e.g. those incurring impressions, clicks, or conversions) [46] is time-dependent, where each auction outcome influences both users and advertisers, leading to a variance in the next auction. Traditional algorithms usually learn an independent predictor and conduct fixed optimization that amounts to a greedy strategy, which often does not lead to the optimal return [4]. The Proportional-Integral-Derivative (PID) controller is also widely applied in real time bidding systems [44]. The drawback of the approach is that it is hard

to make use of user/ad features. To learn a time-dependent relationship and obtain an adaptive strategy, it is necessary to formulate the bidding sequence as a Markov Decision Process (MDP), where both the immediate feedback and the long-term reward are utilized to alleviate the instability and capture the transitions.

Agents with Reinforcement Learning (RL) address the aforementioned challenge to some extent [4, 13, 46]. The optimization goal, however, is limited to either revenue or ROI, which is only a part of the overall utility of the auction game. We posit that the utility is threefold, as outlined in the following: (1) for advertisers, they want to get more impressions or engagements [4] while the cumulative cost is kept within the budget; (2) for users, they want to receive ads of relevance that do not hinder normal browsing activities; and (3) for publishers, they want to balance the trade-off between demand and supply [41], as well as maintain a high and sustainable profitability. In summary, an optimal RTB bidding agent needs to consider *multiple objectives*, which is not adequately addressed in the existing literature.

In this paper, we formulate the continuous impression-level bidding as an MDP problem. Furthermore, inspired by the achievements of actor-critic (AC) framework in games [23], robotics [27], linguistics [19], and vision [11], we propose an extended multi-objective actor-critic method, *MoTiAC*, to optimize the various objectives in RTB simultaneously. MoTiAC employs several actor-critic networks with different objectives to interact with the same environment, and then update the global network asynchronously according to different reward signals, which learns a robust strategy under macro- and micro-exploration. To our best knowledge, this is the first multiple objective-specialized actor-critic framework. We also design a distributed implementation for industrial deployment. To demonstrate the effectiveness of MoTiAC, we report evaluation results on real-world datasets in Section 5.

Contributions. The contributions of our work can be summarized as follows:

- We formalize real-time bidding as a multi-objective MDP problem, and provide motivation to use RL as the solution
- We generalize actor-critic framework and propose a novel multi-objective AC framework, MoTiAC, to derive optimal bidding for RTB, which to our knowledge is the first in the literature
- We design a large-scale distributed implementation for the model
- We demonstrate the performance of MoTiAC using real-world industry data from Tencent Social Ads

2 RELATED WORK

Real-time Bidding. RTB is a multi-billion dollar business that has generated much interest in the research community. In order to improve user experience, extensive research has been devoted to predict user behaviors, e.g. Click Through Rate (CTR) [5, 14], Conversion Rate (CVR) [40, 42]. However, bidding optimization remains one of the most challenging problems in real-time bidding.

In [26, 45], the authors regard bidding environment as a stationary setting, and propose a static optimization method to perform impression-level evaluation. However, real-world situations in RTB are often dynamic [4] due to the irrationality of user behavior [13] and different marketing plans [41] from advertisers. Other studies [8, 10] focus on sponsored search. They optimize bid price at the keyword level, usually based on budget pacing [41], bid generation and matching [3], and utility estimation [2]. In fact, keywords might not be the most relevant indicator, especially in real-time paradigm. Recently, [46] builds an SS-RTB generating model with RL to handle the changing environment, but still follows the static SS assumption.

Wang et al. [38] employs CNN to process ad texts that then passes the high-level semantic representation directly to a *Deep Q-Networks* (DQN) [24] agent. Zhao et al. [46] uses DQN to tackle instability in SS-RTB. MDP [7] is often used to help make sequential decisions in dynamic environments, either by value iteration [4, 33], or policy iteration [16, 18]. Another work [13] uses Deep Deterministic Policy Gradient (DDPG) [20] with competitive and cooperative agents. While RL can help handle the unstable environment [4, 46] in RTB, these works all limit themselves to optimize only a single objective.

Considering the varied goals of different players in RTB, we need a robust framework that can strike a balance between these multiple objectives. Therefore, we are motivated to propose a novel multi-objective RL model to maximize the overall utility in RTB.

Reinforcement Learning. *Markov Decision Process* (MDP) has been widely used to model and solve sequential decision problems in stochastic environments [21]. A typical MDP is defined as a 4-tuple (S, A, P_a, R_a), where S and A are finite sets of states and actions, respectively. $P_a(s, s') = Pr(s_{t+1} \mid s_t = s, a_t = a)$ is the probability that action a in state s at time step t will lead to state s' at the next time $t + 1$. $R_a(s, s')$ is the immediate reward signal after the transition from s to s' due to action a. Estimation of the transition probability P_a (i.e., the optimal policy) is the key of MDP, where two popular Dynamic Programming (DP) algorithms are well studied: value iteration [4, 33] and policy iteration [18].

In reinforcement learning (RL), intelligent systems are often modeled as MDPs. Combining policy gradient [35] and Q-Learning [39], actor-critic algorithm [17] was first proposed, using value function approximation (i.e., the critic) as the base reward to help guide the actor towards better performance. With the advancement of GPU and deep learning (DL), in [24], the authors put forth Deep Q-Networks (DQN) to play the atari game, utilizing the power of pixel convolution to make infinite states controllable in MDP. Instead of discrete actions [12, 29] in previous RL setting, [20, 32] enable RL agent with stochastic power and pushes DQN to the continuous action domain.

With the great achievement of RL frameworks in games [23], robotics [27], linguistics [19], and vision [11], a number of efforts in applying RL agents to RTB are reported, including the use of Q-Learning [4], DQN [38] and multi-agent actor-critic [13, 46]. With RL agents and the MDP formalization, the non-stationary characteristics of RTB can be addressed. However, most existing RL models are limited to optimize a single objective [23, 24].

There are some previous attempts to address multiple objectives using RL, where the multiple reward functions are mostly combined by either linear weights [9, 25] or complicated transformations [22] such as policy votes [31]. However, weighted combination is only valid when the objectives do not compete [30], while learning a

complicated transformation is usually slow and unstable in real world [36]. In this paper, we generalize the traditional actor-critic framework and propose MoTiAC to balance different objectives by employing different groups of actor-critics.

3 PRELIMINARIES

3.1 Ad Pricing Models

In online advertising realm, there are three main ways of pricing. Cost-Per-Mille (CPM) [13] is the first standard, where advertisers are charged based on one-thousand impressions. Since the income is occurred with an impression event, publishers (platforms) has the lowest risk in this way given the abundance of the event and the nonnecessity of a complex machinery to predict the user behavior. Cost-Per-Click (CPC) [45] is a performance-based pricing model, that is, only when user clicks the ad can the publisher get paid. However, due to the challenge of CTR prediction, both platforms and advertisers take some risks. Another model is Cost-Per-Acquisition (CPA) with payment attached to each conversion event. CPA is favored by advertisers since valuation is straightforward and budget planning is less challenging.

All these pricing models make either publishers or advertisers bear more risk in the auction, and thus a new pricing model that decouples the bidding action and the charging action is proposed by some online advertising platforms to strike a balance between both sides. For example, an ad may enter the auction with a CPC bid, and is charged on conversions (which is marketed as the Optimized Cost-Per-Acquisition model, or oCPA[1] by Tencent Social Ads). With this scheme, it is relatively easy for advertisers to express valuation and plan budget, while also making ad allocation feasible for publishers. A comparison of these pricing models is presented in Table 1.

Model	Bidding	Charge	Budget Ctrl. of Advertiser	Incoming Expe. of Publisher
CPM	impression	impression	hard	low
CPC	click	click	medium	medium
CPA	conversion	conversion	easy	high
oCPA	conversion	click	easy	medium

Table 1: Pricing Model Comparison

Example. If one ad gets 2000 impressions, 10 clicks and 2 conversions, and the agreed cost is \$0.5 per mille, \$0.2 per click and \$0.9 per conversion. Then in CPM / CPC / CPA / oCPA model, expected payment for advertiser is \$1.0, \$2.0, \$1.8, \$1.8, respectively. And for the publisher, actual income is \$1.0, \$2.0, \$1.8, \$2.0, respectively in each model.

3.2 Problem Definition

In the oCPA model, publishers charge based on *clicks*, while advertisers estimate budgets by *conversions*. The key point in this bidding scheme is that platforms should make optimal strategy to properly allocate overall impressions among ads, so as to make sure that the charged *clicks* compensate for *conversions* that the advertisers are expected to gain. Specifically,

[1] http://ads.tencent.com/

$$clicks \times CPC_{next} \approx conversions \times CPA_{target} \tag{1}$$

where CPC_{next} is the cost charged in the second-price auction [37] and CPA_{target} is set for each conversion. In our system, CPA_{target} is given by advertisers when creating an ad campaign. The mission for our bidding agent is to generate a CPC_{bid} price, adjusting the winner of each impression. We denote $I = \{1, 2, ..., n\}$ as bidding iterations, $A = \{ad_1, ad_2, ...\}$ as a set of all advertisements. For each iteration $i \in I$, $ad_j \in A$, our bidding agent will decide on a $CPC_{bid}^{(i,j)}$ to enter the auction. Then the ad with highest $CPC_{bid}^{(i,j)}$ wins the impression, and then receives the possible $clicks^{(i,j)}$ (which will be charged by $CPC_{next}^{(i,j)}$) and $conversions^{(i,j)}$ according to user behavior/preference. On the one hand, when CPC_{bid} is set higher, ads are more likely to win this impression, so as to get clicks or later conversions. On the other hand, higher CPC_{bid} means lower opportunities for other auctions given a limited budget. Therefore, considering the joint benefit of all advertisers, our first objective is thus deemed to be the *lower overall CPA*, which we formulate as the *CPA* goal,

$$\min \frac{\sum_{i \in I, ad_j \in A} clicks^{(i,j)} \times CPC_{next}^{(i,j)}}{\sum_{i \in I, ad_j \in A} conversions^{(i,j)}}, \tag{2}$$

Meanwhile, publishers need to earn return on investment by increasing *conversions*, which we hence put forth as the *Conversion* goal,

$$\max \sum_{i \in I, ad_j \in A} conversions^{(i,j)}, \tag{3}$$

This is a multi-objective optimization problem, and we need to not only control advertisers' budget, but make profitable decision for the total utility of the platform. Traditional RTB bidding strategy based on controllers or single-objective RL agent do not adequately address challenges presented in such setting. In this work, we design MoTiAC with a group of actor-critics for the *CPA* goal, and another group of actor-critics for the *Conversion* goal. In the next section, we give detailed description of the proposed MoTiAC method.

4 METHODOLOGY

In this section, we present our multi-objective actor-critic framework in detail. We try to answer the following questions. What is the advancement of MoTiAC comparing to traditional AC? Why can MoTiAC work well? How can we deploy it in a distributed ad system? The organization of this section is: (1) we present basic actor-critic framework in Sec. 4.1; (2) we propose *Reward Partition* to handle multiple rewards and justify its superiority in Sec. 4.2; (3) we well specify MoTiAC and show the algorithm in Sec. 4.3; (4) Multiple rewards are designed in Sec. 4.4; (5) implementation details for distributed systems can be seen in Sec. 4.5.

4.1 Actor-Critic Framework

Actor-critic algorithm is proposed first in [17], then [20, 32] generalizes the discrete action space to continuous one (DPG), and [23] advances it with multiple actor-critics updating the same global network asynchronously (A3C). A typical actor-critic reinforcement learning setting consists of:

- **state:** each state is a feature vector reconstruction of a bidding situation s. We use feature representations extracted from user profile, bidding environment and so on.
- **action:** in our model, action works as bidding for each ad based on the input state. Instead of using discrete action space [38], our model will output an action distribution, and then we sample actions according to its probability.
- **reward:** obviously, reward is a feedback signal from the environment to evaluate how good previous action is, which guides RL agent towards a better policy. In our model, we design multiply rewards based on multiple goals. Each actor-critic couple will observe one type of reward from environment, and together they achieve multiple objectives.
- **policy:** in an actor-critic thread, policy is present as $p(a_t \mid s_t)$, which denotes the probability to take action a_t under state s_t. Different from model-based MDP stated in Sec. 2, where transition probability is given by $p(s_{t+1} \mid s_t, a_t)$, actor-critic framework is totally model-free and thus is able to handle infinite states in real world [12, 23].

In an actor-critic architecture, the critic uses TD learning with a linear approximation and the actor is updated in an approximated gradient direction based on the information provided by the critic [17]. There exists a self-improving loop in this process: following the current policy π_θ, RL agent plays an action a_1 in state s_1, and receives reward signal r_1 from environment. Then this action will lead to a new state s_2, where agent repeats this round again. One trajectory τ can be written as $\{s_1, a_1, r_1, s_2, a_2, r_2, \ldots\}$. For each policy π_θ, we define the utility function as,

$$U(\pi_\theta) = E_{\tau \sim p_\theta(\tau)}[R(\tau)], \tag{4}$$

where $p_\theta(\tau)$ denotes the distribution of trajectories under policy π_θ, and $R(\tau)$ is a reward function over trajectory τ. After sampling N trajectories from policy π_θ, parameters θ will be updated after one or several rounds based on tuple (s, a, r) in each trajectory τ to maximize the utility $U(\pi_\theta)$, formally,

$$\theta \leftarrow \theta + \eta_1 \sum_{n=1}^{N} \sum_{i=1}^{T_n} R_i^{(n)} \times \nabla \log_\pi(a_i^{(n)} \mid s_i^{(n)}), \tag{5}$$

where, $R_i^{(n)} = \sum_{j=i}^{T_n} \gamma^{T_n - j} r_j^{(n)}$ is the cumulative discounted reward for state s_i, γ denotes decaying factor. Then critic calibrates the gradient by adding a baseline reward using value function $V(s; \theta_v)$ [6, 34] (MC-based policy iteration is used) as,

$$\theta \leftarrow \theta + \eta_1 \sum_{n=1}^{N} \sum_{i=1}^{T_n} (R_i^{(n)} - V(s_i^{(n)}; \theta_v)) \times \nabla \log_\pi(a_i^{(n)} \mid s_i^{(n)}), \tag{6}$$

4.2 Reward Combination & Partition

As is stated in 3.2, RTB business shall require multiple objectives. A natural way is to integrate multiple rewards into a single one and we call it *Reward Combination*. However, it turns out to be ineffective in real world [28]. Thus, we are motivated to propose *Reward Partition*. In this subsection, we will compare these two schemes and justify the superiority of *Reward Partition*.

Reward Combination. One intuitive way [9, 25] is to (1) compute a linear combination of the rewards, in which case each element of w_k quantifies the relative importance of the corresponding objectives: $R(s_i) = \sum_{k=1}^{K} w_k \times R_k(s_i)$; (2) define a single-objective MDP with the expected return equals to value function $V(s_i; \theta_v)$.

However, these implementations ignore the premise that one or both of the steps is impossible, infeasible, or undesirable [28]. (1) A weighted combination is only valid when objectives do not compete [30]. However, in RTB setting, objectives usually conflict in terms of each side, making this strategy blind in practice. (2) The intuitive combination will flatten the gradient with respect to each objective, and thus the agent is likely to limit itself within a narrow boundary of search space. (3) Explicitly combining objectives may not be flexible in some cases, especially in changing environment. Although we can integrate into a single-objective reward, this low-dimension representation may not be easily approximated by only one value function, thus learning can be slow and unstable [36].

Reward Partition. We propose *Reward Partition* scheme and design multiple reward functions with respect to multiple objectives. Each reward is employed in one worker group, where actor-critic couples will explore based on the same objective using a low-dimension state representation. In our scheme, we have one global network with an actor network and multiple critic networks. In the beginning of one iteration, actor-critics update local parameters from global network, and start exploration. Actor-critics from each worker group then will explore based on their own mission and push weighted gradients to the actor and one of the critics in the global network asynchronously.

Our scheme shows several advantages. First, different rewards are not explicitly combined in our model, and thus the conflicts should be addressed to some extent. Second, each actor-critic aims at only one objective, so our model should explore in a relatively larger space. Third, we do not use a complex reward combination, which is usually hard to learn in most of real cases [36]. Instead, we use multiple value functions to approximate multiple single rewards in small subsets of features. This strategy maps low-dimension feature representation to single reward, making our critics easy to learn.

In addition, we present mathematical analysis for *Reward Partition* in terms of parameters update and value function approximation. If we attach the weights of *Reward Combination* to the gradients in *Reward Partition*, the parameters updating strategy should be identical on average. For *Reward Combination*, actor parameter θ is updated by,

$$\theta \leftarrow \theta + \eta_1 \sum_i \left(\left(\sum_{k=1}^{K} w_k \times R_k(s_i) - V(s_i; \theta_v) \right) \times \nabla \log_\pi(a_i \mid s_i) \right),$$

while in *Reward Partition*, expected gradient is generated as,

$$\theta \leftarrow \theta + \eta_1 \sum_{k=1}^{K} w_k \times \left(\sum_i (R_k(s_i) - V_k(s_i; \theta_v)) \times \nabla \log_\pi(a_i \mid s_i) \right),$$

$$\theta \leftarrow \theta + \eta_1 \sum_i \left(\left(\sum_{k=1}^{K} w_k \times (R_k(s_i) - V_k(s_i; \theta_v)) \right) \times \nabla \log_\pi(a_i \mid s_i) \right).$$

It is obvious that the difference lies in advantage part, thus the effect of parameters update exactly depends on how well and precisely the critic can learn from its reward. By learning in a decomposed

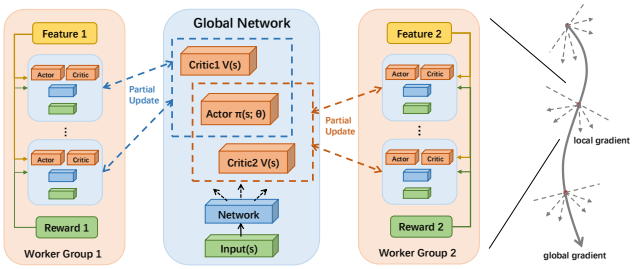

Figure 2: MoTiAC Framework

level, our multiple easy-to-learn critics should advance intuitive linear combination, which means by using this updating policy, *Reward Partition* can yield a better critic and thus a better actor.

4.3 MoTiAC Algorithm

Fig. 2 shows our model architecture. As stated in Sec. 3.2, there are two objectives in our problem. We extend traditional A3C framework, and replace unified workers with multiple objective-specialized actor-learners. On the one hand, for *objective 1* (*CPA* goal), several workers interact with environment to achieve lower real *CPA*, while on the other hand, another batch of workers feed gradients to the global network based on *Conversion* goal. Different objective-specialized workers that run in parallel are likely to explore different parts of the environment. Also for each objective, one can explicitly use different policies in each actor-learner to maximize this diversity [23]. Under macro- and micro-level exploration, MoTiAC is guaranteed to achieve a better policy.

Macro-level Exploration. As stated above, optimization of two objectives is a non-trivial setting in our problem. [48] proposed a weighted function to balance global convolutions and local convolutions in graph-based semi-supervised classification. In loss back-propagation, we also introduce a similar weighted function $\lambda(t)$ to shift priority across objectives, specifically,

$$\theta \leftarrow \theta + \eta_1 \times \lambda^{(j)}(t) \times \nabla E_{\tau \sim p_\theta}[Reward_1(\tau)],\ \forall ad_j \in A,$$
$$\theta \leftarrow \theta + \eta_1 \times [1 - \lambda^{(j)}(t)] \times \nabla E_{\tau \sim p_\theta}[Reward_2(\tau)], \quad (7)$$

where $\lambda^{(j)}(t)$ is a priority function on time t in range $(0, 1)$ and is tailored for each $ad_j \in A$. Eqn. (7) well handles multi-objective conundrum: (1) it updates gradients to global net with respect to *objective 1* and *objective 2* simultaneously; (2) Also, each ad's special situation and fluctuation is properly captured.

Based on priority factor $\lambda(t)$, together with the strategy of running different exploration policies in different groups of workers, the overall changes being made to the global actor parameters θ are likely to be less correlated and more objective-specified in time, which means our model can make *wide* exploration and achieve a balance between multiple objectives with a global overview.

Micro-level Exploration. In our framework, one group of workers can learn to adopt different policies simultaneously towards the same objective. This parallelism enables agents to experience a variety of different states at any given time-step [23]. For example, given an ad with high *CPA*, these paralleled agents will control biddings within different levels, and consequently updates the global network to some extent in light of their respective rewards. Later, when a similar occasion comes, the global network recalls this memory and will choose a best action according to previous instances.

In addition to precise calibration of parameters, this micro-level mechanism reduces training time and enables us to use on-policy reinforcement learning with more stability, which places sequential decision making under *deep* exploration.

Aggregated Loss Function. In MoTiAC framework, we use a continuous action space, so instead of a single value, our model outputs action distribution for inference. Therefore, loss for a single learner is gathered from actor policy θ, critic network θ_v and action distribution entropy, which is agreed to improve exploration by discouraging premature convergence to sub-optimal [23],

$$\begin{aligned} L_{\theta,\theta_v} &= \eta_1 \times L_{actor} + \eta_2 \times L_{critic} + \eta_3 \times L_{action} \\ &= \eta_1 \times E_{\tau \sim p_\theta(\tau)}[R(\tau)] + \eta_2 \times \frac{1}{2} \sum_{s_i \in \tau} \|V(s_i; \theta_v) - R_i\|^2 \\ &\quad + \eta_3 \times \sum_{s_i \in \tau} E_{a \sim p_{\theta_v}(s_i)}[-\log(a)]. \end{aligned} \quad (8)$$

where R_i is objective-specified. Then after one iteration (e.g., 10-minute simulation) we use Adam Optimizer [15] to compute gradients for each actor-critic couple, and push the weighted gradients to the global net. With multiple actor-learners applying online updates in parallel, global network exploits and explores to achieve a robust balance between multiple objectives in Algo. 1.

Algorithm 1: Training for each actor-critic in MoTiAC

// Assume global shared parameters vectors θ and θ_v;
// Assume objective-specific parameter vectors θ'_k and $\theta'_{v,k}$, k=1,2;
initialize step counter $t \leftarrow 1$; epoch T_{max}; discounted factor γ;
while $t < T_{max}$ **do**
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$;
  Synchronize specific parameters $\theta'_k = \theta$ and $\theta'_{v,k} = \theta_v$;
  Get state s_t;
  // Assume ad set $A = \{ad_1, ad_2, \ldots\}$;
  for $ad_j \in A$ **do**
    repeat
      Perform a_t according to policy $\pi(a_t \mid s_t; \theta)$;
      Receive reward r_t w.r.t *objective* k;
      Reach new state s_{t+1}; $t \leftarrow t + 1$;
    until *terminal state* s_t;
    for $i \in \{t-1, \ldots, 1\}$ **do**
      $r_i \leftarrow r_i + \gamma \times r_{i+1}$;
      // Accumulative gradient w.r.t θ'_k;
      $d\theta'_k \leftarrow d\theta'_k + \eta_1 \sum(r_i - V(s_i; \theta'_{v,i})) \times \nabla \log_\pi(a_i|s_i)$;
      // Accumulative gradient w.r.t $\theta'_{v,k}$;
      $d\theta'_{v,k} \leftarrow d\theta'_{v,k} + \eta_2 \sum \partial \|r_i - V(s_i; \theta'_{v,k})\|^2 / \partial \theta'_{v,k}$
        $+\eta_3 \times \sum Entropy(a; s_i, \theta'_{v,k}) / \partial \theta'_{v,k}$;
    end
    // Asynchronously update θ and θ_v with $d\theta'_k$ and $d\theta'_{v,k}$;
    // Using $\lambda^{(j)}(t)$ or $1 - \lambda^{(j)}(t)$ w.r.t objective k;
    $\theta \leftarrow \theta + \lambda^{(j)}(t) \times d\theta'_k$ and $\theta_v \leftarrow \theta_v + \lambda^{(j)}(t) \times d\theta'_{v,k}$;
  end
end

4.4 Reward Design in RTB

In real-time bidding environment, we have multiple objectives as stated in Sec. 3.2. Well motivated by A3C, we design different actor-learners on different goals.

Objective 1: minimize overall *CPA*. One of our goals in this problem is allocating impression-level bids in every auction round, so that each ad will get enough opportunities for display and later get clicks or conversions, which makes CPA_{real} close to CPA_{target} (for the benefit of advertisers) given by advertisers at the beginning. Final reward for this agent pool is designed as,

$$CPA_{real}^{(j)} = \frac{\sum_{i=1}^{n} clicks^{(i,j)} \times CPC_{next}^{(i,j)}}{\sum_{i=1}^{n} conversions^{(i,j)}}, \ \forall ad_j \in A, \tag{9}$$

$$Reward_1\left(\frac{CPA_{real}^{(j)}}{CPA_{target}^{(j)}}\right) = \begin{cases} -x^2, & 1.2 < x \\ 1 - x, & 1 < x \leq 1.2 \\ 1, & x \leq 1 \end{cases}. \tag{10}$$

Since it is not necessary to meet CPA_{target} at each moment, we record cumulative real *CPA* to adjust bid for each ad per auction campaign, in order to achieve this goal at the final moment (i.e., it is not a moment-level notion but an overall result). When real *CPA* is lower than target one, reward signal will be positive 1; and when it is a little bit higher, we return a small negative value assuming that advertisers can still accept this price; however, when real *CPA* deviates from CPA_{target} by a large margin, reward will be set a large negative signal according to the power of bias.

Objective 2: maximize conversions. Another mission for agents is to enlarge conversions as many as possible under the condition of not destroying CPA_{real}, so that platform can stay competitive and run a sustainable business. Final reward in this part is shown as follows,

$$conversions^{(j)} = \sum_{i=1}^{n} conversions^{(i,j)}, \ \forall ad_j \in A, \tag{11}$$

$$Reward_2\left(\frac{conversions^{(j)}}{conversions_{base}^{(j)}}\right) = \begin{cases} 1, & 1 < x \\ x, & 0.8 < x \leq 1 \\ x - 0.8, & x \leq 0.8 \end{cases}, \tag{12}$$

where *conversions* is a cumulative notion until auction $i \in I$. In train, we can get $conversions_{base}$ from history statistics summed to auction $i \in I$ for each $ad_j \in A$ to scale reward. When conversion numbers is higher than the base one, the agent will receive a positive reward; then if conversions remain a little bit lower, we still consider it a good adjustment, because in this moment real *CPA* must be lower accordingly; however, when the policy is radical with extremely few conversions, we should also stop it to meet the *CPA* requirement and reduce the loss of advertisers.

4.5 System Deployment & Implementation

Real world ad exchange data with magnitude of TB requires distributed schema and parallel computation. Considering memory load and system efficiency, we provide a fully distributed offline MoTiAC deployment scheme in this section.

Functional Modules. In our ad systems, data preprocessing is conducted within multiple steps/modules (e.g., pCVR/pCTR prediction module, conversion back-flow crawler, auction server), which are distributed among machines. The core module is *Data Manager*, which serves directly for each distributed actor-critic. On the one hand, it coordinates data from other modules, which is then fed as auction environment for training. On the other hand, bidding results will be gathered and processed to generate reward signal, guiding model optimizer with correct gradient. Functional modules are mostly disposable, and thus the challenge we need to address lies in model training and cross-machine/process data communication, which are the most time-consuming parts.

Flowed Data Source. To directly load one-day ad click log means killing $\sim 100GB$ memory, which shouldn't be allowed in real world application. In our implemented architecture, we use multiple readers to simultaneously read the original logs once a batch (e.g., 200 ads), process the information and then transform it into a smaller package, which will later be stored into the shared memory. When data manager prepares back-end information, it will refresh this shared memory after processing, making more room for ad packages. The whole data set shall be processed within this flow. Reading original data from log files appeals to be unavoidable and has been the bottle-neck of our problem. However, multi-reader setting addresses this trade-off and mitigates it to some extent with the cost of more cpu power.

Parallel Cluster. In stead of using multi-thread [23], in our system we implement distributed MoTiAC in multi-process. Each actor-critic agent will be assigned a machine/process, and it takes actions and receives rewards to either reduce real *CPA* or raise conversions. Agents runs in parallel, and they communicates with data manager and other modules using cross-machine/process connection by Redis. Due to the sparsity of reward in real application, we use a threshold to control activity for each ad, in order to make sure that accumulative conversions/cost is enough for adjustment.

Most of the parts in online system are identical to our experiments, especially the model training procedure. Therefore, experiment results shall present the efficiency and effectiveness of this distributed systematic architecture.

5 EXPERIMENTS

In experiment, we will use five-day real world data to answer the following questions. Q1: how well can MoTiAC perform in general? Q2: how does $\lambda(t)$ impact the performance of MoTiAC?

5.1 Experiment Setup

Dataset. Our experiments are conducted over real world click logs collected in Tencent Social Ads system from Jan. 7th 2019 to Jan. 11th 2019. We perform a five-folder cross-validation, i.e. using 4 days for training and another one day for testing. Ads are either in html5 web pages (H5) or applications of mobile devices (App). Each day, we will have $\sim$ 10000 ads with $\sim 13GB$ click logs and other supporting materials. Following the rule of data collection in our company, click logs have already been preprocessed. We list the relevant files used in our experiments.

- **click logs:** each advertisement has one log file with multiple rows, each row contains impression-level bidding information when this ad has been clicked, such as date, product type, site set, ad id, pCVR, pCTR, click time, target cpa, etc.

- **back flow conversion logs**: conversion numbers given by advertisers at the end of each time period.

Due to large demands and supplies, our platform conducts bidding auctions frequently. Thus in our experiment, time granularity is set to be 10 minutes (144 time periods for a day), which is much shorter than 1 hour [13]. In real world bidding campaign, one tricky part is that *back flow conversions* are usually not correct. For example, if for one day, 50 users have bought a product, but the back flow conversion number may appear to be only 20 at the end of the day. The asynchrony between real conversions and back flows make it a hard problem to handle. Therefore, in the experiment, we filter out ads with less than 50 real conversions and ads with less than 20 back flow conversions, we also use empirical back flow rate to compute the estimated conversion for model usage.

Problem Re-definition. We refine our problem in this section based on our data. In Sec. 3.2, we claim that our two objectives are: (1) minimize overall *CPA*; (2) maximize conversions. Since gross merchandise value (GMV) is a common indicator for platform earnings, and it turns out to be proportional w.r.t conversions. Cost is the money paid by advertisers, which also appears to be a widely accepted factor in online advertising. Therefore, without loss of generality we reclaim our two objectives to be: (1) maximize $\frac{GMV}{Cost}$ (*CPA* goal) and (2) maximize *GMV* (*Conversion* goal). Note that *GMV* is also known as revenue, while $\frac{GMV}{Cost}$ as ROI.

5.2 Compared Approaches

We carefully select related methods for comparison, and adopt the same settings for all compared bidding methods with 200 iterations. Details about implementation can be seen in Appendix A.2.

- **Proportional-Integral-Derivative (PID):** [1] is a widely used policy in feedback control system, which produces the control signal from a linear combination of the proportional factor, the integral factor and the derivative factor. PID is free from training. In Tencent online ad system, PID is currently used to control sequential bidding.
- **Advantage Actor-Critic (A2C):** [23] proposed a well-known actor-critic framework in RL history. However, basic A2C can only handle single objective situation, we used combined reward function in experiment comparison.
- **Deep Q-Network (DQN):** [4, 13] used DQN with a single target under assumption of consistent state transition. We combine reward functions, and also use the same discrete action space (interval is 0.01) like [38].
- **Aggregated A3C (Agg-A3C):** [23] proposed a asynchronously updated framework in RL history. A3C is chosen to show the superiority of network asynchronously updating. In experiment, without loss of generality we use the following aggregated reward function in A2C, DQN and Agg-A3C:

$$reward = \lambda(t) \times Reward_1 + (1 - \lambda(t)) \times Reward_2 \qquad (13)$$

  where $\lambda(t)$ is set via customized priority. We design various $\lambda(t)$ policies and analyze their sensitivity in Sec. 5.5.
- **Objective1-A3C (O1-A3C):** To show the advantages of using multi-objective actor-critics, we implement our model solely based on *objective 1*. This baseline should only consider one side and achieve its extreme.
- **Objective2-A3C (O2-A3C):** same settings for *objective 2*.
- **Multi-objective Actor-Critics (MoTiAC):** This is the algorithm we proposed to solve multi-objective RTB problem. MoTiAC is supposed to reach a harmonious bidding optimal. In experiment, we implemented it using customized priority. The details of implementation can be seen in Appendix A.2.

5.3 Evaluation Protocol & Machine

First, we need to make it clear that due to *Tencent Privacy Law*, we cannot report the exact performance of PID, so the result of comparison will be shown on its basis (*each value below represents $\frac{value}{value_{PID}}$). Since we have a multi-objective setting, result of two objectives will be shown at the same time with detailed explanation and analysis.

In our experiment, huge memory load are required. We implement all the experiments with PyTorch and two 128GB memory machine with 56 CPUs.

5.4 AtoQ1: General Experiment

We report basic comparison of MoTiAC and other approaches in Table 2. Note that all results are in the basis of PID, and values in parenthesis show improvement/reduction percentage.

Model	Revenue	Cost	ROI
A2C	1.0019 (+0.19%)	1.0366 (+3.66%)	0.9665 (-3.35%)
DQN	0.9840 (-2.60%)	0.9765 (-2.35%)	1.0076 (+0.76%)
Agg-A3C	1.0625 (+6.25%)	1.0952 (+9.52%)	0.9702 (-2.98%)
O1-A3C	0.9744 (-2.56%)	0.9580 (-4.20%)	1.0170 (+1.70%)
O2-A3C	1.0645 (+6.45%)	1.0891 (+8.91%)	0.9774 (-2.26%)
MoTiAC	1.0421 (+4.21%)	1.0150 (+1.50%)	1.0267 (+2.67%)

Table 2: Comparative Result based on PID

Objective Comparison. Though we have multiple goals, it is easy to judge the effectiveness of these models over PID. A2C is shown to be the worst one, though it gains a slightly higher revenue (conversion goal) over PID, but ROI (*CPA* goal) in this method is much lower comparing to PID, which is unaccepted. This result proves that one single agent cannot fully capture the dynamics in RTB environment. Though based on a hybrid reward, DQN fails to maintain the balance of revenue and ROI. Compared to O1-A3C, DQN outputs a similar revenue but lower ROI by about 1%. We suspect that the discrete action space may limit the policy to some local and unstable optimal. Agg-A3C seems to perform the opposite. In the training process, we observe that Agg-A3C can beat PID in a certain margin, but it cannot show advantage during testing. In our experiment, it shares a similar performance as O2-A3C with high revenue and low ROI. The reason might be (1) environment changes in different days, we cannot fix the formula of reward aggregation for these method; (2) direct combination of reward is unable to help us achieve a better situation. Nevertheless, compared to A2C, this result tells the superiority of multi-workers.

It is worth noticing that two ablation models O1-A3C and O2-A3C present two extreme situations. O1-A3C gives pretty high ROI, but results in low revenue, and vice versa for O2-A3C. Our proposed MoTiAC is the best one, since it outperforms PID online algorithm

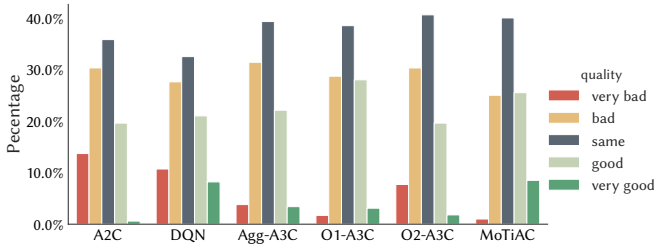

Figure 3: Bidding Score Distribution over PID

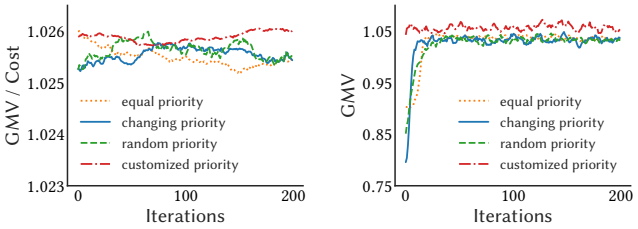

Figure 4: Result Under Different Priority

in terms of both revenue and ROI, which means publisher should earn more incomes, also advertisers will pay less for relatively more conversions. By running different workers in parallel, multi-objective actor-critics helps break the limitation in Agg-A3C.

We also show the bidding quality of these models over PID in Fig. 3. We evaluate each ad, and then group then into five categories: very bad, bad, same, good and very good. The result is based on the relationship between CPA_{real}, CPA_{target}, GMV and $Cost$, and the detailed evaluation metrics can be found in Appendix A.1.

Distribution Comparison. With Fig. 3, the comparison between different approaches should be clear. We can conclude that A2C and O2-A3C demonstrates a similar result: though most of the ads lies in "same", category, but in total, there are more bad ads than good ones, which means PID performs better in terms of multi-objective RTB problem over these two methods. O1-A3C has a balanced distribution, which means its perfromance is very similar to online PID, in part because they both aims at minimizing real CPA. DQN appears to be a radical actor. It tends to make bidding towards either very good or very bad and suffers a lot from fluctuation, which reproves its property of instability. On the contrary, Agg-A3C has a mild performance, it shares the same pattern with O1-A3C, but slightly worse. In other words, Agg-A3C also performs very similarly to PID, meaning that the combined reward does not really work in our experiment.

From this figure, it is obvious that in terms of bidding score distribution, online PID algorithm is slightly better than or at least similar to any other baseline. The reason might be that PID is a basic control strategy and only utilizes the most important signals, so that the impact from small noise of environment will not influence the overall performance. In our proposed MoTiAC, several easy-to-learn critics can also eliminate the influence from noise, and it turns out to have a desirable improvement over PID with more ads in the right "good" side and less ads in the left "bad" side. Of course, the ideal case should be that all ads lies in "good", "very good" or at least the "same" category. However, in RTB dynamic setting, for now industry is far from solving the RTB problem [38], in all, we think the achievement our MoTiAC made should be cutting-edge.

5.5 AtoQ2: Sensitivity Analysis

To give a comprehensive view of MoTiAC, we have tried different hyper-parameters in our model, and will discuss its sensitivity to some extent. We mainly consider the interesting variants of $\lambda(t)$.

***λ(t)* Sensitivity.** In this experiment, each group consists of five workers. Four $\lambda(t)$ variants are compared:

- equal priority: $\lambda(t) = \frac{1}{2}$;
- changing priority: $\lambda(t) = exp(-\alpha \cdot t)$, where α is a time scalar;
- random priority: $\lambda(t) = random([0, 1])$;
- customized priority: $\lambda^{(j)}(t) = \frac{O_1}{O_1+O_2}$, where O_1 and O_2 are two objectives, $\forall ad_j \in A$.

As is shown in Fig. 4, we draw objective results $\frac{GMV}{Cost}$ and GMV for each iteration. The first three strategies are designed from an overall view regardless of different ad situations. It turns out that they are very similar in both objectives, and can gain a decent improvement over PID case by around +2.5% in $\frac{GMV}{Cost}$ and +3% in GMV. To be more specific, in *equal priority*, curve of $\frac{GMV}{Cost}$ generally drops when we use more iterations, which might stem from that fixed equal weights cannot fit the dynamic environment. For *changing priority*, it is interesting that $\frac{GMV}{Cost}$ first increases then decreases with respect to priority shifting. Because different priority leads to different optimal. In *random priority*, curves turns out to dramatically change in a small interval, since priority also fluctuates in random. On the contrary, *customized priority* case, from a micro-level view, sets priority in ad granularity and dominates the first three strategies by an increasingly higher CPA achievement by +2.6% and better GMV by around +4.3%.

Based on this observation, we can safely conclude that it is better to set $\lambda(t)$ a special function according to ad property. However, in general this experiment shows that different strategies will not bring a tremendous difference to the result, and our MoTiAC model should be robust to $\lambda(t)$ from a large-scale.

6 CONCLUSION

In this paper, we propose Multi objective-specific Actor-Critics (MoTiAC) for the benefit of both advertisers and advertising platforms in real-time bidding. To our best knowledge, MoTiAC is the first to utilize objective-specialized actor-critics to solve multi-objective bid optimization. By formulating the bidding process as an MDP, our model can learn a state transition relationship and obtain an adaptive bidding strategy in a dynamic environment. In addition, we design a *reward partition* scheme in MoTiAC, which is demonstrated to have better performance over traditional *reward combination* in multi-objective RL problem.

To optimize multiple objectives, MoTiAC employs several actor-critic with different missions to interact with the same environment, and then update the global network asynchronously according to different reward signals, which learns a robust strategy under macro- and micro-exploration. We also design a distributed architecture for the industrial deployment of the model, balancing the

trade-off between memory usage and data load. We conduct extensive experiments on real-world industrial dataset. Empirical results shows that MoTiAC outperforms the state-of-the-art bidding algorithms, and can generate +4% lift in revenue and +2.5% in ROI for Tencent's advertising platform.

REFERENCES

[1] Stuart Bennett. 1993. Development of the PID controller. *IEEE control systems* 13, 6 (1993), 58–62.

[2] Christian Borgs, Jennifer Chayes, Nicole Immorlica, Kamal Jain, Omid Etesami, and Mohammad Mahdian. 2007. Dynamics of bid optimization in online advertisement auctions. In *Proceedings of the 16th international conference on World Wide Web*. ACM, 531–540.

[3] Andrei Broder, Evgeniy Gabrilovich, Vanja Josifovski, George Mavromatis, and Alex Smola. 2011. Bid generation for advanced match in sponsored search. In *Proceedings of the fourth ACM international conference on Web search and data mining*. ACM, 515–524.

[4] Han Cai, Kan Ren, Weinan Zhang, Kleanthis Malialis, Jun Wang, Yong Yu, and Defeng Guo. 2017. Real-time bidding by reinforcement learning in display advertising. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 661–670.

[5] Junxuan Chen, Baigui Sun, Hao Li, Hongtao Lu, and Xian-Sheng Hua. 2016. Deep ctr prediction in display advertising. In *Proceedings of the 2016 ACM on Multimedia Conference*. ACM, 811–820.

[6] Thomas Degris, Patrick M Pilarski, and Richard S Sutton. 2012. Model-free reinforcement learning with continuous action in practice. In *American Control Conference (ACC), 2012*. IEEE, 2177–2182.

[7] Manxing Du, Redouane Sassioui, Georgios Varisteas, Mats Brorsson, Omar Cherkaoui, et al. 2017. Improving Real-Time Bidding Using a Constrained Markov Decision Process. In *International Conference on Advanced Data Mining and Applications*. Springer, 711–726.

[8] Eyal Even Dar, Vahab S Mirrokni, S Muthukrishnan, Yishay Mansour, and Uri Nadav. 2009. Bid optimization for broad match ad auctions. In *Proceedings of the 18th international conference on World wide web*. ACM, 231–240.

[9] Eli Friedman and Fred Fontaine. 2018. Generalizing Across Multi-Objective Reward Functions in Deep Reinforcement Learning. *arXiv :1809.06364* (2018).

[10] Ariel Fuxman, Panayiotis Tsaparas, Kannan Achan, and Rakesh Agrawal. 2008. Using the wisdom of the crowds for keyword generation. In *Proceedings of the 17th international conference on World Wide Web*. ACM, 61–70.

[11] Chris Gaskett, Luke Fletcher, and Alexander Zelinsky. 2000. Reinforcement learning for a vision based mobile robot. In *IEEE IROS 2000*, Vol. 1. 403–409.

[12] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[13] Junqi Jin, Chengru Song, Han Li, Kun Gai, Jun Wang, and Weinan Zhang. 2018. Real-Time Bidding with Multi-Agent Reinforcement Learning in Display Advertising. *arXiv preprint arXiv:1802.09756* (2018).

[14] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. 2016. Field-aware factorization machines for CTR prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 43–50.

[15] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[16] Daphne Koller and Ronald Parr. 2000. Policy iteration for factored MDPs. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 326–334.

[17] Vijay R Konda and John N Tsitsiklis. 2000. Actor-critic algorithms. In *Advances in neural information processing systems*. 1008–1014.

[18] Alessandro Lazaric, Mohammad Ghavamzadeh, and Rémi Munos. 2016. Analysis of classification-based policy iteration algorithms. *The Journal of Machine Learning Research* 17, 1 (2016), 583–612.

[19] Jiwei Li, Will Monroe, Tianlin Shi, Sébastien Jean, Alan Ritter, and Dan Jurafsky. 2017. Adversarial learning for neural dialogue generation. *arXiv preprint arXiv:1701.06547* (2017).

[20] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[21] Shiau Hong Lim, Huan Xu, and Shie Mannor. 2013. Reinforcement learning in robust markov decision processes. In *Advances in Neural Information Processing Systems*. 701–709.

[22] Daniel J Lizotte, Michael H Bowling, and Susan A Murphy. 2010. Efficient reinforcement learning with multiple reward functions for randomized controlled trial analysis. In *Proceedings of the 27th ICML*. 695–702.

[23] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[25] Ramakanth Pasunuru and Mohit Bansal. 2018. Multi-reward reinforced summarization with saliency and entailment. *arXiv preprint arXiv:1804.06451* (2018).

[26] Claudia Perlich, Brian Dalessandro, Rod Hook, Ori Stitelman, Troy Raeder, and Foster Provost. 2012. Bid Optimizing and Inventory Scoring in Targeted Online Advertising. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '12)*. ACM, 804–812.

[27] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. 2017. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542* (2017).

[28] Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. [n. d.]. A survey of multi-objective sequential decision-making. *JAIR 2013* ([n. d.]).

[29] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv:1707.06347* (2017).

[30] Ozan Sener and Vladlen Koltun. 2018. Multi-task learning as multi-objective optimization. In *Advances in Neural Information Processing Systems*. 525–536.

[31] Christian R Shelton. 2001. Balancing multiple sources of reward in reinforcement learning. In *Advances in Neural Information Processing Systems*. 1082–1088.

[32] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic policy gradient algorithms. In *ICML*.

[33] Matthijs TJ Spaan and Nikos Vlassis. 2005. Perseus: Randomized point-based value iteration for POMDPs. *JAIR* 24 (2005), 195–220.

[34] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[35] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*. 1057–1063.

[36] Harm Van Seijen, Mehdi Fatemi, Joshua Romoff, Romain Laroche, Tavian Barnes, and Jeffrey Tsang. 2017. Hybrid reward architecture for reinforcement learning. In *Advances in Neural Information Processing Systems*. 5392–5402.

[37] Hal R Varian. 2007. Position auctions. *international Journal of industrial Organization* 25, 6 (2007), 1163–1178.

[38] Yu Wang, Jiayi Liu, Yuxiang Liu, Jun Hao, Yang He, Jinghe Hu, Weipeng Yan, and Mantian Li. 2017. LADDER: A Human-Level Bidding Agent for Large-Scale Real-Time Online Auctions. *arXiv preprint arXiv:1708.05565* (2017).

[39] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.

[40] Hong Wen, Jing Zhang, Quan Lin, Keping Yang, Taiwei Jin, Fuyu Lv, Xiaofeng Pan, Pipei Huang, and Zheng-Jun Zha. 2018. Multi-Level Deep Cascade Trees for Conversion Rate Prediction. *arXiv preprint arXiv:1805.09484* (2018).

[41] Jian Xu, Kuang-chih Lee, Wentong Li, Hang Qi, and Quan Lu. 2015. Smart pacing for effective online ad campaign optimization. In *Proceedings of the 21th ACM SIGKDD*. 2217–2226.

[42] Yuya Yoshikawa and Yusaku Imai. 2018. A Nonparametric Delayed Feedback Model for Conversion Rate Prediction. *arXiv preprint arXiv:1802.00255* (2018).

[43] Shuai Yuan, Jun Wang, and Xiaoxue Zhao. 2013. Real-time bidding for online advertising: measurement and analysis. In *Proceedings of the Seventh International Workshop on Data Mining for Online Advertising*. ACM, 3.

[44] Weinan Zhang, Yifei Rong, Jun Wang, Tianchi Zhu, and Xiaofan Wang. 2016. Feedback control of real-time display advertising. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. ACM, 407–416.

[45] Weinan Zhang, Shuai Yuan, and Jun Wang. 2014. Optimal real-time bidding for display advertising. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1077–1086.

[46] Jun Zhao, Guang Qiu, Ziyu Guan, Wei Zhao, and Xiaofei He. 2018. Deep Reinforcement Learning for Sponsored Search Real-time Bidding. *arXiv preprint arXiv:1803.00259* (2018).

[47] Han Zhu, Junqi Jin, Chang Tan, Fei Pan, Yifan Zeng, Han Li, and Kun Gai. 2017. Optimized cost per click in taobao display advertising. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2191–2200.

[48] Chenyi Zhuang and Qiang Ma. 2018. Dual Graph Convolutional Networks for Graph-Based Semi-Supervised Classification. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. 499–508.

A ADDITIONAL MATERIALS

A.1 Pricing Evaluation Metrics

We present our bidding quality evaluation metrics below: Algo. 3 is the evaluation strategy based on our ad system, and in Algo. 2, we specify the *Score* function used in Algo. 3.

Algorithm 2: Score Function

Input: $\Delta loss$
Output: *score*
function $Score(F)$**:**
  if $\Delta loss \leq 0.1$ **then**
    Return 0 // similar performance
  else if $\Delta loss \leq 0.2$ **then**
    Return 1 // good / bad performance
  else
    Return 2 // very good / bad performance
  end
  // sign will be given outside
end

A.2 Implementation Details

In this appendix section, we shall discuss our implementation details for all baselines and our MoTiAC. All baselines except PID use 17 original features for 200 iterations.

- **PID**: control signal is given by difference across two intervals in our experiment, in order to making sure less fluctuation and stable result. Ratio for proportional factor is 0.04 , for the integral factor is 0.25 and for the derivative factor is 0.002. We also use manual min-max adjustment to bound the result.
- **A2C, Agg-A3C**: in A2C, we use one couple of actor-critic. We use two hidden layers with 256 and 128 dimensions in neural network. Output for *Actor* are one-dimension scalars μ and σ (a Gaussian Distribution), and then actions will be sampled from this distribution with boundaries. Output for *Critic* is one-dimension scalar. *State* is a normalized 17-dimension feature vector. *Reward* is set to be,

$$reward = \lambda(t) \times Reward_1 + (1 - \lambda(t)) \times Reward_2 \qquad (14)$$

  where $\lambda(t)$ is set via customized priority. We design various $\lambda(t)$ policies and analyze their sensitivity in Sec. 5.5. Agg-A3C has the same architecture, but is on extension of A2C with 5 workers running in parallel and updates global net asynchronously.
- **DQN**: all settings in DQN are the same as in A2C and Agg-A3C, except for *Actor*. The output of *Actor* has 100 dimension w.r.t 100 values ranging from 0 to 1, and 0.01 as its interval.
- **O1-A3C, O2-A3C**: all settings in O1-A3C and O2-A3C are the same as in A2C and Agg-A3C, except for reward. *Reward* in O1-A3C and O2-A3C is set according to their objectives. Detailed formula of reward can be found in 4.4.
- **MoTiAC**: we also use two hidden layers with 256 and 128 dimensions for all nets in MoTiAC. However, the difference is that we have two critics in global nets with the same structures. And for each update period, one group of workers can only push their gradients to actor and one of the critics. We set 5 worker as an objective group in experiments. Also, MoTiAC is implemented with multiprocessing and shared memory.

Algorithm 3: Pricing Evaluation over PID.

// We use $Value_{method}$ to denote the value given by other approaches, and $Value_{PID}$ denotes the value given by PID;
// Pricing score is given per ad;
for $ad_j \in A$ **do**
  // *CPA* calculation;
  $\Delta cost = cost_{method} - cost_{PID}$;
  $\Delta conversion = conversion_{method} - conversion_{PID}$;
  $\Delta CPA = (\Delta cost + 0.5 \times CPA_{target}) \,/\, (\Delta conversion + 0.5)$;
  $CPA_{PID} = cost_{PID} \,/\, (conversion_{PID} + 0.01)$;
  $CPA_{method} = cost_{method} \,/\, (conversion_{method} + 0.01)$;
  // Bias calculation;
  $\Delta bias = \Delta CPA \,/\, CPA_{target} - 1$;
  $bias_{PID} = CPA_{PID} \,/\, CPA_{target} - 1$;
  $bias_{method} = CPA_{method} \,/\, CPA_{target} - 1$;
  // Enter the loop;
  if $bias_{PID} \leq 0.2$ **then**
    $\Delta loss = \Delta cost \,/\, cost_{PID}$;
    $score = Score(\Delta loss)$;
  else
    if $bias_{method} \leq 2 \bigwedge bias_{PID} \geq 0.3$ **then**
      $score = 2$;
    end
    $\Delta loss = \Delta cost \,/\, cost_{PID}$;
    if $\Delta bias \geq 0.3$ **then**
      $score = -Score(\Delta loss)$;
    else
      $score = Score(\Delta loss)$;
    end
  end
  Return score for ad_j;
end