An Automated Deep Reinforcement Learning Pipeline for Dynamic Pricing

Citation for published version (APA):

DOI:
10.1109/TAI.2022.3186292

Document status and date:
Published: 01/06/2023

Document Version:
Accepted manuscript including changes made at the peer-review stage

Please check the document version of this publication:
- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 15. Mar. 2024
An Automated Deep Reinforcement Learning Pipeline for Dynamic Pricing

Reza Refaei Afshar, Jason Rhuggenaath, Yingqian Zhang, and Uzay Kaymak

Abstract—Dynamic pricing problem is difficult due to the highly dynamic environment and unknown demand distributions. In this paper, we propose a Deep Reinforcement Learning (DRL) framework, which is a pipeline that automatically defines the DRL components for solving a Dynamic Pricing problem. The automated DRL pipeline is necessary because the DRL framework can be designed in numerous ways, and manually finding optimal configurations is tedious. The levels of automation make non-experts capable of using DRL for dynamic pricing. Our DRL pipeline contains three steps of DRL design, including MDP modeling, algorithm selection, and hyper-parameter optimization. It starts with transforming available information to state representation and defining reward function using a reward shaping approach. Then, the hyper-parameters are tuned using a novel hyper-parameters optimization method that integrates Bayesian Optimization and the selection operator of the Genetic algorithm. We employ our DRL pipeline on reserve price optimization problems in online advertising as a case study. We show that using the DRL configuration obtained by our DRL pipeline, a pricing policy is obtained whose revenue is significantly higher than the benchmark methods. The evaluation is performed by developing a simulation for the RTB environment that makes exploration possible for the RL agent.

Impact Statement—Dynamic Pricing problem has a great impact on the revenue of the sellers, and it emerges in many areas such as content distribution, retail and wholesaling, online advertising, and transportation businesses. The problem is to dynamically determine the price of items, promotions, services, etc. Existing mathematical methods typically assume some fixed information about the buyers and their willingness to pay. However, in most Dynamic Pricing problems, the demand distribution may change, which is difficult for mathematical methods to adapt. Furthermore, existing machine learning methods need to be redesigned if the problem properties change. The automated deep reinforcement learning method proposed in this paper assumes no information about the buyers and automatically develops the solution model. Hence, manual redesigning is unnecessary, and the method can be used in areas where expert knowledge is unavailable. According to the case study, our proposed method significantly increases the sellers’ revenue in the online advertising platform. This method is ready to support sellers in setting the prices for their items.

Index Terms—Automated Reinforcement Learning pipeline, Dynamic Pricing, Bayesian Optimization, AutoRL.

I. INTRODUCTION

In a typical Dynamic Pricing (DP) problem, a seller needs to derive a pricing policy that assigns a price for each of her products in order to maximize her expected total revenue. DP is a challenging problem because if the prices are too high, no buyer is willing to pay, and if the prices are low, the seller’ The DP problem emerges in different applications, each having its own demand distribution and pricing constraints [1].

The DP problem can be modeled as a sequence of decisions in a finite or infinite horizon in which previous prices are used for future decisions. Uncertainty in the environments and modeling as a sequential decision-making problem make Reinforcement Learning (RL) an appropriate approach for solving this problem. Hence, we develop an RL framework to derive an efficient pricing strategy for the DP problem. In general, the price is a continuous value even though it can be discretized. Moreover, the observation space is usually large, considering items and buyers’ properties. Thus, our proposed method is based on policy gradient [2]. A complex DP environment with large action and state spaces motivates us to use Deep Neural Networks (DNN) as a function approximator. Hence, our proposed method is based on Deep Reinforcement Learning (DRL), containing a policy and value networks.

To design a solution for DP based on DRL, some typical decisions have to be made before starting the training procedure. These decisions correspond to defining the DRL components such as decision moments, MDP modeling, and setting hyper-parameters. The DRL components are normally defined using expert knowledge. However, it might take several runs of the algorithm to test candidate configurations. Furthermore, the optimal configuration is not necessarily obtained by trial and error of limited candidate configurations. For these reasons, we aim to automate finding the optimal configuration of the DRL framework and develop a DRL pipeline for DP problems.

The components of a DRL pipeline are shown in Fig. 1. Unlike common practice where this pipeline is designed manually, our proposed DRL pipeline starts with automatically defining states and reward function. The process of state definition is transforming available information and features that might influence the performance of pricing into a state representation. Determining the reward function is performed by following the reward shaping approach proposed in [3]. Actions are drawn from probability density functions (PDFs) parameterized by the outputs of the policy network, and the policy PDF is jointly optimized during the hyper-parameters optimization procedure. We propose a novel approach, called Bayesian-Genetic Hyper-parameters Optimization, that defines a separate Bayesian Optimization (BO) framework for each value of expensive-to-run hyper-parameters. Although many works focus on automating a particular component of DRL, to the best of our knowledge, this is the first complete DRL pipeline that integrates the automation of different compo-
Fig. 1: Overview of the proposed automated DRL pipeline. Solid arrows show the main pipeline. Dashed arrows indicate the feedback loop and interactions between different components that automate the design of the DRL solution. The pipeline starts with MDP formulation and then algorithm selection and hyper-parameters optimization. The performance of the policy is used to tune the configuration. The arrow between algorithm selection and hyper-parameter optimization is dim because we assume a particular DRL algorithm to be included in the hyper-parameters optimizer.

We evaluate our DRL pipeline using a pricing problem in Real-Time Bidding (RTB). In order to provide the opportunity for exploration, we developed an RTB simulation that works based on RTB historical data. We consider a Multi-Armed Bandit (MAB) approach that only utilizes the Header Bidding Partners (HBPs) bids and Ad Exchange (AdX) responses as a benchmark method. We show that our proposed method significantly outperforms the MAB approach in terms of revenue. The contributions of this paper are as follows.

- We develop a complete DRL pipeline for DP problem that automates MDP modeling and hyper-parameters optimization procedures.
- We propose a novel hyper-parameter optimization method, called Bayesian-Genetic Hyper-parameters Optimization, that combines Bayesian Optimization and the selection operator of Genetic algorithm.
- We build a simulation for RTB systems based on real dataset.
- We explore the benefit of information in DP, by comparing the result of our DRL approach and a MAB approach that does not depend on the information of ad slots.

The paper is organized as follows. Section II reviews related work. Section III presents background knowledge. In section IV the Automated DRL pipeline for DP is elaborated. Section V describes a reserve price optimization problem. Section VI presents a simulation model, a set of experiments and results of applying our method to solve the reserve price optimization problem. Section VII concludes the paper.

II. LITERATURE REVIEW

A. Dynamic Pricing

Most of the work on DP leverages mathematical optimization models for deriving the pricing strategy. In [4] the joint problem of dynamic pricing, advertising, and inventory control is modeled using an objective function that contains a term corresponding to each problem in [5], the long-term profit is characterized as a function of inventory level for perfect and limited information cases, and dynamic programming is used to derive prices. Machine Learning (ML) methods are the other class of approaches for DP. DP is investigated in [6] where a maximum quasi-likelihood regression with lasso regularization is employed for unknown demands. ML methods are also leveraged for identifying demand patterns [7] and setting prices for businesses like Airbnb where there is no identical product [8].

Due to the high turnover of Online Advertising and Real-Time Bidding (RTB), DP of ad slots in display advertising has gained attention in the past few years. Multi-Armed Bandit (MAB) modeling is explored, and algorithms like Upper Confidence Bound (UCB) and Thompson Sampling are used for deriving a pricing policy [9], [10], [11]. Approaches based on survival analysis are developed in [12] and [13] in which the reserve price is set according to the probability of being outbid in RTB auctions. In [14], the reserve price is obtained by maximizing a custom objective function where the parameters are updated sequentially using the gradient ascent algorithm. In [15], RL is adopted for DP in sponsored search, which is the process of online advertising in search engines. Apart from reserve price optimization, RL is employed for other problems of RTB systems such as optimizing bidding strategy [16] and ad network ordering of waterfall strategy [17]. Our method differs from most of these work as we assume there is no information about demand distributions, and advertisers might dynamically change their bidding strategies over time.

B. Automated Reinforcement Learning (AutoRL)

In recent years, researchers developed various methods to automate MDP modeling, algorithm selection, and hyper-parameter optimization in AutoRL [18]. MDP modeling consists of defining states, actions, and rewards. Defining state representation is mainly related to modifying the raw observation of the environment to improve the final policy. These methods range from simple approaches like tile coding and coarse coding for linear function approximation [2], to more complex methods like structure2vec [19] and Pointer Networks [20] for graph combinatorial optimization problems, and state aggregation [21] for Knapsack Problem. Automating actions can be performed in different ways, such as discretizing continuous action space [22] and replacing the policy gradient formula with an equation that learns the policy distribution [23]. Curriculum Learning [24] and reward shaping [25] are the two main methods for automating reward function. Algorithm selection and hyper-parameters optimization are intertwined because an algorithm depends on the optimal hyper-parameters to work well. One popular way of automating algorithm selection is modeling the problem as MAB problem and assigning an action to each algorithm [26]. BO, which works well for automated machine learning, has been used for hyper-parameters tuning for RL algorithms [27] and for adjusting weights of different objectives in the reward function [28]. In [29], hyper-parameters of the RL algorithm and the network structure are jointly optimized using the Genetic algorithm
in which each individual is a DRL agent. Nevertheless, a pipeline that contains different automation levels is missing in the literature on AutoRL. Our work proposes an AutoRL pipeline for dynamic pricing.

### III. BACKGROUND

#### A. Dynamic Pricing Problem

Let \( t \in \mathcal{I} \) be an item or a product, and \( \mathcal{I} \) is the set of all available items. The owner aims to adjust a price \( a_t \) for each item, and the objective is to maximize \( \sum_a a_t \) over the set of selected items for pricing. There is a lower bound and an upper bound for the price of each item \( t \). Let \( \zeta^l_t \) be the lower bound that is a guaranteed revenue if \( t \) is not sold. This value is typically lower than the price, and the owner prefers to sell the item instead of returning and refunding it. In many pricing tasks, \( \zeta^l_t \) is zero, and the owner acquires no revenue if the item is not sold.

The upper bound for the price of item \( t \) is denoted by \( \zeta^u_t \), representing the buyers’ willingness to pay, i.e., no buyer would buy an item \( t \) if its price is higher than \( \zeta^u_t \). Typically, \( \zeta^u_t \) is unknown. Otherwise, the owner can easily set \( a_t = \zeta^u_t \) and maximize the revenue. Using the historical data, \( \zeta^u_t \) can be estimated by averaging or finding the maximum value. These estimations are not useful for setting the prices because the environment, including the buyers’ preferences and qualities of the items, is dynamic and subject to change over time. Hence, one single estimation for \( \zeta^u_t \) would not work. Besides, if the estimated \( \zeta^u_t \) is higher than the real \( \zeta^u_t \), the item remains unsold, which negatively affects the revenue. For these reasons, this work aims to adjust the price for each item considering no prior information about \( \zeta^u_t \).

In terms of available data for the owner, two different cases can be defined. In a standard case, \( \zeta^u_t \) is revealed to the owner after selling the item, and this value can be used for further processing. In another case, aggregated revenue is reported to the owner, and \( \zeta^u_t \) is unavailable per individual item. For each item, the owner only knows whether it is sold or not. This binary value is denoted by \( \beta_t \), which is one if the item is sold. An example of this case is the RTB systems, where the sold prices of the ad slots are not provided for an ad publisher, and it receives daily or hourly aggregated revenue.

In our modeling, \( \zeta^u_t \) is obtained either from the real data or by simulation. The former corresponds to the first case where \( \zeta^u_t \) is revealed to the owner, and the latter corresponds to the second case where \( \beta_t \) is the only response for item \( t \). Each item has a set of features describing its properties. We define a set of \( K \) features that construct a feature vector for each item. Our proposed DRL framework uses these feature vectors as environment observations and decides the prices accordingly. The feature vector \( \vec{F}_t \) is \( (f_1, f_2, ..., f_K) \). This feature vector is not necessarily the optimal representation for the states in the DRL pipeline that is elaborated in section IV-A1.

#### B. Reinforcement Learning

Single-agent fully-observable finite-horizon MDP that we focus in this article is defined as a tuple \( (S, A, R, T, t, \gamma) \). In this tuple, \( S \) is the set of states, \( A \) is the set of actions, \( R \) is an instant reward, \( T \) shows the transition probability, \( \gamma \) is the discount factor, and \( t \) determines a decision point in time. We use the same notation for decision moments as the items in Dynamic Pricing modeling because the timesteps and decision moments in the RL modeling of Dynamic Pricing correspond to items. In other words, the decision moment \( t \) is deciding the price of item \( t \) in timestep \( t \). At each decision moment or timestep \( t \), the agent observes \( s_t \in S \) and takes action \( a_t \in A \). Performing \( a_t \), alters the state of the environment from \( s_t \) to \( s_{t+1} \) and returns a scalar reward \( R \) to the agent. The agent updates a policy \( \pi(.|s_t) \) using the scalar rewards that assign a probability value to each action. Using these probabilities, an action is selected by following a greedy or Softmax policy.

For a wide variety of RL tasks, the action space is continuous. It is impossible for continuous action space to assign a separate output for each action because the number of actions is very large. Continuous action spaces are typically handled by using policy gradient methods, and a PDF parameterized by the outputs of the policy function is defined for each action. According to [2], in policy gradient, the parameters of the policy function is updated using the gradient of some performance measure \( J(\theta) \) as \( \nabla J(\theta) = \mathbb{E}_\pi \sum_t Q(\pi(S_t, a)) \nabla \theta \pi(a|S_t, \theta) \). The policy gradient theorem establishes that the gradient of the performance measure \( J(\theta) \) pertains to the expected Q function and the gradient of the policy function over all possible actions.

In DP problems, an action is a price of an item, which is naturally a real number. We choose to use the PPO algorithm as an actor-critic method due to several reasons. First, it is applicable to continuous actions. Second, the price is normally in a predefined region, and PPO manages large updating of the policy network by applying a clip operation on the gradient. Third, it is flexible in terms of policy PDF, and different PDFs can be easily tested using the policy output. The objective function of PPO is:

\[
L_{\text{CLIP}}(\theta) = \mathbb{E}_t \left[ \min(\min(r_t(\theta)A_t), \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t) \right],
\]

where clip function transforms every values of \( r(\theta) \) by clipping them if they are either higher than \( 1+\epsilon \) or lower than \( 1-\epsilon \), and \( r(\theta) \) is the ratio of probability values obtained from current and old policies, i.e., \( r_t(\theta) = \frac{\pi_{\text{old}}(a_t|s_t)}{\pi(\theta|a_t|s_t)} \). We select a particular DRL algorithm and decide to optimize other components to reduce the complexity of the pipeline. However, the algorithm selection procedure can be included in the hyper-parameters optimization step by adding an identifier showing the type of algorithm. Typically, the environment is dynamic, and it is possible to have very different instant rewards. Different rewards lead to large loss values, which makes large jumps in policy updates. Therefore, clipping is necessary in this case to prevent the policies from large updating.

### IV. DEEP REINFORCEMENT LEARNING PIPELINE FOR DYNAMIC PRICING

The proposed DRL pipeline starts with automatically modeling the problem as an MDP, followed by hyper-parameter optimization.
A. MDP Modeling

The DRL pipeline models the DP problem as a finite-horizon fully observable MDP in which states, actions, and rewards are automatically defined. The MDP is episodic, and the length of all episodes is one because the price of each item is assumed to be independent of other items.

1) States: The items’ feature vectors are not necessarily suitable for representing the states in a DRL framework, requiring a pre-processing step. The DNNs used as policy and value networks receive numerical values, while there might be some categorical features in the \( F_t \). Therefore, the first pre-processing step is to convert all categorical values to numerical using One-Hot Encoding. Categorical features can have a large number of unique values, which makes the resulted One-Hot encoded feature vector very large. One way of solving this issue is to pick \( y \) most frequent unique values and group the others as a single name. In this case, the obtained feature vector contains \( y + 1 \) features. After converting categorical features, the missing values and outliers are handled using the most common values and box plot, respectively. As the final step, \( \zeta_t \) is added to the feature vector as extra information to improve decision-making quality. In sum, the state representation for an item contains the numerical features, One-Hot encoded of the categorical features, and \( \zeta_t \).

2) Actions: The action is the price for item \( t \) and the decision moments are when an item is available for pricing. The price is a real number, and hence its corresponding action is continuous. Two ways of handling a continuous action are either discretizing it or leveraging a policy distribution. Although discretization may reduce the complexity of the problem, it could lead to overgeneralization. For example, it is possible to group two very close actions as a single discrete action while they have quite diverse rewards. This especially happens in a pricing problem when \( a_t \) is slightly lower and slightly higher than \( \zeta_t \). Therefore, we opted for using policy gradient methods with a probability distribution for sampling actions, elaborated in Section IV-B.

3) Rewards: Several methods of defining the reward function in reserve price optimization are investigated in [3]. Similar definitions are valid for the general dynamic pricing problem. However, the main issue is that they set the price very close to \( \zeta_t \) to ensure the item is sold. The reward shaping approach in [3] solves this issue by prioritizing high prices using a weight for each interval of price values. The interval between \( \zeta_t \) and a fixed estimation of \( \zeta_t \) denoted by \( \zeta_{max} \) is divided into \( n \) equal sub-intervals. Then, two vectors of size \( n + 2 \) are defined for reward values and weights. Each entry of the reward vector corresponds to a sub-interval, and it is non-zero only if the price is in that interval. The definition of the reward vector and the weights are optimized in the hyper-parameters optimization step.

B. Bayesian-Genetic Hyper-Parameters Optimization

The next component in the DRL pipeline tunes and optimizes the hyper-parameters. In a DRL framework, hyper-parameters are the parameters that are assumed to be fixed during training, and they are normally adjusted by using expert knowledge. Hyper-parameters include learning rate, discount factor, eligibility trace coefficient, etc. As mentioned before, selecting the DRL algorithm can also be included in the optimization procedure by adding a new parameter.

Our hyper-parameters optimizer is based on Bayesian Optimization, which works well when evaluating by the original objective function is expensive. In order to increase the time efficiency of using BO for hyper-parameters optimization, we are inspired by the method presented in [30] and develop a novel optimization approach that leverages the idea of the selection of GA in BO. Similar to [30], the idea of our method is to combine BO and GA as an EA in order to increase time efficiency. Unlike [30], our method divides the hyper-parameters into two groups based on their evaluation cost and performs separate BO procedures for each group. A set of hyper-parameters for optimization are shown in table I.

Among these hyper-parameters, by changing the values of the learning rate, batch size, clip size, epoch number, \( n \), and \( w_j \), the running time remains the same. They are mostly scalar values, and changing scalar values would not alter the running time of the training procedure, assuming that the other hyper-parameters have not been changed. However, changing policy distribution, number of layers, number of nodes in each layer, and activation function influence the running time significantly, because computing the gradients and performing backpropagation algorithm for different PDFs, layers, nodes, and activation functions vary in terms of running time. For this reason, we divide the hyper-parameters into two groups. Group one contains learning rate, batch size, clip size, epoch number, reward vector, \( n \), and \( w_j \). Group two includes policy distribution, number of layers, number of nodes, and activation function. For simplicity, we focus on policy PDF in the rest of this section. However, separate BO frameworks can also be assigned to the number of layers or number of nodes while the other hyper-parameters of group two are fixed. The advantage of this division is that the DRL agent handles the hyper-parameters separately and avoids spending too much time testing different values of the second group’s hyper-parameters.

Assume that there are \( M \) candidate PDFs to use as policy distribution. For each candidate PDF, a separate BO framework with a Gaussian Process is developed to optimize the hyper-parameters of group one. We specify a time threshold instead of the number of timesteps for the budget of each BO framework because the required time for training the model with different PDFs varies. Let \( \Psi \) be the total time budget for running each BO framework. Since the running time of different policy distributions is different, the number of tested values for hyper-parameters settings is also different. Different BO frameworks can test a varying number of hyper-parameters. For example, since computing the gradient of Multi-Variate Normal (MVN) and inverse of the covariance matrix are time-consuming, the number of tested values for other hyper-parameters when MVN is used is significantly lower than the same value for Beta distribution. After running the BO frameworks for \( \Psi \) time units, each PDF has a particular number of hyper-parameter settings and their performance for their Gaussian Process.
Each BO framework has a private memory containing all the evaluated hyper-parameters and their performance. The next step in our proposed method is to pick the top $v$ hyper-parameters in terms of their performance and add them to a shared memory between all BO frameworks. This operation is similar to the selection of GA. Our Bayesian-Genetic algorithm combines Bayesian optimization and the selection operator of GA. Its purpose is to transfer the knowledge between independent BO frameworks to help slow policy distributions utilize the knowledge of faster policy distributions. The size of this memory is at most $M \times v$. This is an upper bound because a particular BO framework may have tested less than $v$ hyper-parameters.

**Algorithm 1** Bayesian-Genetic Hyper-Parameters Optimization

**Input:** The list of hyper-parameters  
**Output:** Optimized values of the input hyper-parameters

\begin{verbatim}
1: Initialize $M$ BO frameworks with $M$ Gaussian Processes $GP_m$, and $M$ empty private memories $M_m$
2: for $m \in \{1, 2, ..., M\}$ do
3:     repeat
4:         Select a set of random hyper parameters from $GP_m$ with the probability of $z$ or a set of hyper-parameters that maximizes the $GP_m$ with the probability of $1 - z$
5:         Run a PPO algorithm on the Pricing environment using the selected hyper-parameters and PDF $m$
6:         Add the hyper-parameters and performance to $M_m$
7:     until $\Psi$ time units past
8: end for
9: Run the selection operator of GA and Pick $v$ top hyper-parameters for each of $M$ BO frameworks and store them in a shared memory $M_{\text{shared}}$

10: for $m \in \{1, 2, ..., M\}$ do
11:     repeat
12:         Select a set of hyper parameters from $M_{\text{shared}}$
13:         Run a PPO algorithm on the Pricing environment using the selected hyper-parameters and PDF $m$
14:         Add the hyper-parameters and performance to $M_m$
15:     until All the hyper-parameters in $M_{\text{shared}}$ are selected
16: end for
17: return The optimal values for the hyper-parameters and the PDF with the highest performance
\end{verbatim}

As the last step, all the BO frameworks use the hyper-parameters in the shared memory to find their performance. In this step, the selected parameters are merely from the shared memory, and no exploration is performed. By finding the new hyper-parameters and performance pairs, new points are added to the Gaussian Process of each BO framework. Finally, the policy network and the hyper-parameters corresponding to the best performance are selected to run the DRL algorithm and derive the final policy. The hyper-parameter optimization algorithm is shown in Algorithm 1. Algorithm 1 starts with initializing $M$ BO framework with $M$ private memories $M_m$. Note that each BO framework corresponds to the hyper-parameters of group two, including the expensive to evaluate hyper-parameters, and the hyper-parameters in a particular BO framework belong to group one. In lines 3 to 8 the Bayesian Optimization algorithm is performed separately for each PDF until $\Psi$ time units are passed. In line 10, top $v$ hyper-parameters are selected using the Selection operator of GA from each private memory $M_m$ and they are stored in $M_{\text{shared}}$. Lines 12 to 16 run BO for the hyper-parameters in the shared memory without exploration, and line 18 returns the best found hyper-parameters and their corresponding PDF to use as policy distribution.

V. CASE STUDY: RESERVE PRICE OPTIMIZATION IN REAL TIME BIDDING

Reserve Price Optimization is the process of adjusting the price of ad slots in RTB systems. Publishers place some blocks called ad slots on their websites, and they obtain revenue by selling those ad slots to the advertisers. When a user loads a publisher’s website, an impression is generated for each ad slot. These impressions are the publisher’s asset, and they are sold to the advertisers. Among different variants of RTB systems, we focus on the system containing HBPs and AdX. An overview of this system is shown in Fig. 2 [3].

![Fig. 2: An overview of RTB systems based on HBPs and AdX. 1) The publisher offers an impression to HBPs. 2) HBPs run auctions among advertisers. 3) The publisher receives the highest bids from HBPs. 4) A reserve price is set for AdX. 5) AdX runs an auction. 6) The sold/unsold response of the AdX auction is sent back to the publisher.](image-url)
Typically, the highest bid of HBPs is used as the reserve price. However, it is possible that AdX finds a bidder that outbids higher reserve prices. Thus, the problem is setting the reserve price for AdX to increase the total revenue. We apply our proposed method in section IV to this problem. The reserve price $a_t$ is set for each impression using its available information. These information are shown in table II.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slot ID ($\tilde{v}_t$)</td>
<td>The unique identifier of an ad slot</td>
</tr>
<tr>
<td>Webpage URL ($Y_t$)</td>
<td>The webpage containing the ad slot</td>
</tr>
<tr>
<td>Reserve Price ($\zeta_t$)</td>
<td>The reserve price for the impression</td>
</tr>
<tr>
<td>Location ($\ell_t$)</td>
<td>The location of the ad slot in the webpage.</td>
</tr>
<tr>
<td>Size ($\xi_t$)</td>
<td>The size of the ad slot (Width x Height)</td>
</tr>
<tr>
<td>Time ($\tau_t$)</td>
<td>Time and date of sending the ad request</td>
</tr>
<tr>
<td>Winner ($\beta_t$)</td>
<td>0: one of HBPs or 1: AdX</td>
</tr>
<tr>
<td>HBP Highest Bid ($\xi_t^{hbp}$)</td>
<td>The highest bid of HBPs</td>
</tr>
<tr>
<td>AdX Winning Bid ($\zeta_t^{AdX}$)</td>
<td>The winning bid of the AdX auction</td>
</tr>
</tbody>
</table>

We denote the highest bid of the HBPs by $\zeta_t^{HBP}$. This value is known to a publisher upon receiving the responses of the HBPs. The winning bid of AdX is denoted as $\zeta_t^{AdX}$. This value is unknown to the publisher, and the publisher only knows $\beta_t$ that indicates whether the impression is sold in the AdX auction. The AdX’s auction is a second price auction where the winner pays as much as the maximum of the second-highest bid and the reserve price. However, because there is no information about the second-highest bid in typical RTB historical data, we use the sum of reserve prices as a lower bound for the revenue.

Any reserve price between $\zeta_t^{HBP}$ and $\zeta_t^{AdX}$ can uplift the revenue. Since $\zeta_t^{AdX}$ is unknown, this adjustment is tricky. On the one hand, if the reserve price is low, the revenue is negatively affected. On the other hand, if $a_t$ is too large, the impression most likely remains unsold in the AdX auction, it goes to the HB channel, and the revenue is equal to $\zeta_t^{HBP}$. Our proposed method aims to handle this trade-off and set the reserve price in order to maximize the revenue. In the following sub-sections, the components of our DRL pipeline are defined for reserve price optimization.

### A. MDP Modeling

We start to apply our DRL pipeline to the reserve price optimization problem by defining the MDP components. The state is a vector consists of the information illustrated in table II and $\zeta_t^{HBP}$ as explained in section IV-A. This feature vector is $\chi^{bid} = (\tilde{v}_t, Y_t, j, \xi_t, \ell_t, \tau_t, \beta_t, \xi_t^{hbp})$. We convert the Slot id, URL, Location, and Size to numerical values using the One Hot Encoding method. Since the number of unique URL values is huge, we picked the most frequent URLs, namely those covering 80% of all URLs, and grouped the rest as an additional label. This results in a feature vector of size 667. The action $a_t$ is the reserve price obtained by sampling from the policy PDF. Since the parameters of candidate PDFs for policy distribution such as Beta, Gamma, Gaussian, and MVG distributions are always positive, we use the Softplus function as the activation function, i.e. $f(x) = \ln(1 + e^x)$. This activation function is also used in [31] that replaces the Gaussian distribution with the Beta distribution for continuous actions. In this case, the activation function is removed from the list of hyper-parameters of the Bayesian-Genetic hyper-parameters optimization module. The reward function is defined using the reward shaping approach of [3]. Different definitions are tested in [3] and the reward function of (1) is selected as it has better performance in terms of total revenue. We use the same definition for the reward vector as this definition is obtained by searching over some possible definitions. In this way, the reward vector is also removed from the list of hyper-parameters in the hyper-parameters optimization module.

$$r_{t,j} = \begin{cases} a_t - \zeta_t^{i} & a_t < l_i \\ \beta_t(a_t - \zeta_t^{j}) & l_j \leq a_t \leq l_{j+1}, \\ \varsigma_{\max} - a_t & a_t > \varsigma_{\max} \end{cases}$$ (1)

In (1), $r_{t,j}$ is the $j^{th}$ entry of the reward vector for item $t$, $l_i$ is $\zeta_t^{HBP}$ as the lower bound of the first interval, and $l_j$ is the lower bound of the $j^{th}$ interval. We use this reward shaping approach to automate the reward function. Unlike [3] where the weights are tuned by testing a fixed set of numbers, we use Bayesian Optimization in Bayesian-Genetic optimizer to optimize the weights of the reward vector.

### B. Hyper-Parameter Optimization

For the architecture of DNN, the number of layers and the number of nodes in each layer are fixed to 2 and 64, respectively, because little differences are observed by varying them. Since the state space is finite and the number of unique states is not very high, a more complex structure of the neural network might not help. The other hyper-parameters, including Learning rate, Batch Size, Clip rate, Epoch Number, and Policy PDF, are used in algorithm 1. Candidate values of these hyper-parameters are based on common values used in DRL implementations. Candidate values of the learning rate are between 0.0001 and 0.01. Batch size is an integer between 20 and 500, epoch number is between 1 and 6, and candidate values for the clip rate of the PPO algorithm are between 0.1 and 0.4. Lastly, $v$ and $\varsigma$ are 10 and 0.3, respectively.

For the parameters of the reward shaping approach, without loss of generality, we define the weights as real numbers between zero and one, and the number of intervals is an integer in the set $\{3, 4, 5, 6\}$. Finally, four PDFs, including Beta, Gamma, univariate Gaussian, and multivariate Gaussian distributions, are the candidate PDFs for policy distribution. In this way, there are four separate BO frameworks and four Gaussian Processes that are used in algorithm 1 to find the best hyper-parameters. The time budget $\Psi$ is twelve hours. During this time, the BO framework corresponding to MVN managed to test 20 configurations as the slowest BO framework. On the other side, the BO framework corresponding to the Beta distribution is the fastest, and tested around one hundred configurations. After running this algorithm, the learning rate values, batch size, clip rate, and epoch number are 0.0021, 108, 0.16, and 2, respectively, and the optimal policy PDF is MVN. The number of intervals in the reward shaping approach
is 5, and the weights of interval zero to interval six are 0.64, 0.13, 0.87, 0.28, 0.49, 0.57, 0.61, respectively. These values are used to train the policy network. It is worth mentioning that the obtained values for the hyper-parameters are not tested during the time budget of the MVN BO framework. These values are optimal for the Beta distribution and they show superior performance when they are tested for MVN in lines 12 to 16 of algorithm 1. This is an advantage of using the Bayesian-Genetic hyper-parameter optimization algorithm.

VI. EXPERIMENTS AND RESULTS

We share our code online 1. Since a DRL pipeline agent needs to explore the environment, which occasionally sets sub-optimal prices during the learning phase, significant revenue might be lost if the actual RTB environment and AdX auction are used for the training. Thus, we opted for developing a simulation model for AdX to provide the opportunity of exploration for the agent. The RTB historical data used for developing the simulation model and evaluating our method is provided by our industrial partner, and it contains the information of the impressions, ζt HBP and βt,.

A. RTB Simulator

This simulator receives an ad request containing the information of impression t together with αt, and returns a binary value βt showing whether the auction has a winner. To determine the winner of the auction, we require ζt AdX which is not included in typical RTB historical data. Although ζt AdX is unknown by the publisher, it is possible to estimate a lower bound for this value using RTB historical data. First, all the impressions where their βt is one are retrieved. These impressions go to AdX, and their reserve prices are ζt HBP. Since AdX is the winner, cAdX t is higher than the reserve price for these impressions. Hence, ζt HBP is a lower bound for ζt AdX. We use these lower bounds to generate ζt AdX. Then we group the impressions by their features and obtain a list of HBP ζt for each feature. The outliers are detected using a boxplot, and the bids higher than the upper quartile are removed from each list. After this, a separate parametric PDF is fit for each list of ζt HBP. The PDF type is fixed for all the lists, although they may have different parameters. We tested different PDFs on a set of randomly selected impressions, and the top eight PDFs in terms of error are shown in table III.

The distributions are compared based on the Residual Sum of Squares (RSS), \( RSS = \sum_{i=1}^{n} (\zeta_{t, \text{HBP}} - \zeta_{t, \text{AdX}})^2 \), where \( \zeta_{t, \text{AdX}} \) is obtained by sampling from a particular PDF. We also define \( \#^{\text{max}} \) as the number of unique set of features that each PDF has the least RSS.

Based on both RSS and \( \#^{\text{max}} \), we selected Log-Normal Distribution in the simulation. The histogram of bids and fitted Log-Normal distributions for two randomly selected feature sets are illustrated in Fig. 4, which shows most bids are between 0 and 0.1, and the Log-Normal distribution fits well with the bids.

Since actual values of ζt AdX are unknown, it is not possible to evaluate them in the simulation. We draw the heatmap of

\[
\begin{array}{|c|c|c|c|}
\hline
\text{PDF} & \text{RSS} & \#^{\text{max}} \\
\hline
\text{Log Normal Distribution} & 1224.65 & 224 \\
\text{Generalized Extreme Value} & 1194.30 & 208 \\
\text{Double Weibull} & 2352.66 & 204 \\
\text{Beta Distribution} & 1435.56 & 90 \\
\text{Gamma Distribution} & 1347.84 & 77 \\
\text{Pareto Distribution} & 1350.98 & 31 \\
\text{Exponential Distribution} & 1173.08 & 21 \\
\hline
\end{array}
\]

\text{TABLE III: comparison of probability density functions.}

Fig. 3: Heatmap of ζt AdX generated by simulator vs. ζt HBP obtained from historical data for the impressions where βt = 1. Each cell shows the number of impressions with a particular ζt AdX and ζt HBP. The brighter cells show higher frequencies.

generated ζt AdX and actual ζt HBP in Fig. 3, which shows the majority of ζt AdX are higher than their corresponding ζt HBP. This observation is well-aligned with our purpose where ζt HBP are lower bounds for the generated ζt AdX.

B. Identifying Important Impressions

We want to focus on the bids that adjusting proper reserve prices can highly increase revenue. For this purpose, we leverage the idea of [32] in which a binary prediction model is developed to classify whether a bid is important or not. Important impressions are those that the difference between ζt HBP and ζt AdX is higher than a threshold. We divide data into different days and use the impressions of one selected day for training and the next day for testing. The target value in our model is defined by setting a threshold on the difference between ζt HBP and ζt AdX. This threshold is 0.7 as it is the rounded average of unique differences of ζt HBP obtained from the historical data and ζt AdX which is generated by the simulator. A Random Forest classifier is developed to identify the important impressions that uses the same data as the main DRL pipeline. The performance of the classifier is acceptable, with F1-score of 0.7545 and an accuracy of 0.7399, respectively. After finding the important impressions, the DRL pipeline is used for setting their reserve prices.

C. Performance Measures and Benchmarks

We evaluate our DRL pipeline using the following measures [3]: (1) \( \sum_{t} \zeta_{t, \text{AdX}} \): The sum of \( \zeta_{t, \text{AdX}} \) for all impressions of testing data. These values are obtained by the simulator. (2)
The sum of \( \zeta^HBP_t \) for all impressions. (3) \( \sum \alpha_t \): The sum of all reserve prices as a lower bound for the revenue of the impressions. This value is \( \zeta^HBP_t \) if \( \beta_t = 0 \), otherwise it is obtained from the policy network. (4) \( \%_{\alpha_t} \): The performance ratio, measured by \( \sum \alpha_t / \sum \zeta^A_t \). (5) \#^H: The number of impressions that each of the benchmark methods provides the highest reserve price.

The benchmark methods used for comparing our DRL pipeline are as follows.

- **DRL-PL**: The revenue obtained by sum of the reserve prices of our DRL pipeline.
- **H4-2** [3]: This heuristic is developed with reward shaping. The interval between \( \zeta^HBP_t \) and \( \zeta^{\text{max}}_t \) is divided into 4 equal sub-intervals and the reserve price is the lower bound of the second interval. These values work best among several tested intervals and selected lower bound.
- **SA-PM** [13]: It assumes no information about \( \zeta^HBP_t \) and predicts \( \zeta^HBP_t \) and \( \beta_t \). A survival analysis modeling is developed to find the highest reserve price that is most probably outbid in AdX auction.
- **MAB-TS**: This MAB method is elaborated next.

**Multi-Armed Bandit (MAB) algorithms** have been used for optimizing the reserve price of impressions in RTB systems. These methods do not use historical data of the impressions, and they adjust the parameters of each action through methods like UCB and Thompson Sampling. Since these methods do not depend on the data and the function used for providing reserve prices is much simpler than the neural networks, their training times are much lower than our DRL pipeline. Hence, one question that may arise is if MAB algorithms can optimize reserve prices, why do we need complex methods with large running time, such as the DRL framework? In order to explore the benefit of using data and DRL pipeline, we consider the MAB method for reserve price optimization presented in [10] as a benchmark for comparison. This work presents UCB and Thompson Sampling algorithms, and both focus on the same variant of the RTB system as we do. Since the Thompson Sampling (MAB-TS) algorithm has lower cumulative regret than the UCB method, we use this method for comparison.

In MAB-TS, reserve prices come from a fixed set of values, and MAB-TS assigns a PDF for each reserve price. MAB-TS considers a single ad slot and develops appropriate probability distributions. Since the DRL pipeline assumes the ad slot as a feature and is capable of deriving reserve prices for different ad slots, we cannot use the original MAB-TS for comparison. We adapt MAB-TS and DRL pipeline by developing a separate MAB-TS method for each ad slot to have a fair comparison.

The publisher owns ad slots in the adapted MAB-TS. A separate MAB framework is developed for each ad slot. MAB-TS for one single ad slot in [10] associates a Beta distribution to each action, and the parameters of this distribution are updated using \( \alpha_t \) and \( \beta_t \). The fixed set of actions in our modeling are \([0.01, 0.02, \ldots, 0.15]\). These values are selected because \( \zeta^A_t \) is between 0.01 and 0.15 excluding the outliers, and most of the bids are rounded values with two decimal places. MAB-TS starts with sampling and updating the parameters of a beta function. When an impression is available, a value is drawn using the beta distribution for each reserve price. The reserve price corresponding to the action with the maximum value sampled from the beta distributions is selected for sending to AdX if this reserve price is higher than \( \zeta^HBP_t \). Finally, the parameters of the selected beta distribution are updated.

**D. Results**

The results of comparing the benchmark methods explained in VI-C are shown in table IV. This table contains the performance metrics for 5000 randomly selected impressions that are predicted as important according to the prediction model of section VI-B. The value of \( \beta_t \) for each of the methods is derived using the average error of the simulation as a window. In other word, \( \beta_t \) is one if the reserve price is between \( \zeta^A_t + e \) and \( \zeta^A_t - e \) where \( e = 0.006 \) is the rounded average error of the simulation.

**TABLE IV**: Performance metrics of different benchmarks.

<table>
<thead>
<tr>
<th>Method</th>
<th>( \sum \zeta^A_t )</th>
<th>( \sum \zeta^HBP_t )</th>
<th>( \sum \alpha_t )</th>
<th>( %_{\alpha_t} )</th>
<th>#^H</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRL-PL</td>
<td>658.3633</td>
<td>106.5798</td>
<td>466.291</td>
<td>70.82%</td>
<td>3736</td>
</tr>
<tr>
<td>H4-2</td>
<td>658.3633</td>
<td>106.5798</td>
<td>428.7597</td>
<td>65.12%</td>
<td>823</td>
</tr>
<tr>
<td>SA-PM</td>
<td>658.3633</td>
<td>106.5798</td>
<td>220.0498</td>
<td>33.42%</td>
<td>0</td>
</tr>
<tr>
<td>MAB-TS</td>
<td>658.3633</td>
<td>106.5798</td>
<td>405.8767</td>
<td>61.62%</td>
<td>441</td>
</tr>
</tbody>
</table>

As illustrated in table IV, the sum of the reserve prices obtained from DRL-PL is around 5% higher than the best heuristic approach. Although DRL-PL requires a procedure of offline training that might take some time, this method is used based on the greedy policy according to the output of the policy network. Thus, both DRL-PL and H4-2 methods can perform in real-time without any additional latency.

MAB-TS method has a total revenue of 405.8767 which is 61.62% of the sum of all \( \zeta^A_t \). Like DRL-PL, this method takes some time for training, and the process of providing a reserve price for incoming impressions can be performed in real-time. The performance of this method in terms of \( \%_{\alpha_t} \) is around 9% lower than our proposed DRL-PL, which is remarkable when the number of impressions is very large.

Although SA-PM uplifts the revenue in comparison with using \( \zeta^HBP_t \) as the reserve price, it provides the lowest reserve price among all benchmarks. One possible reason behind this observation is that this method assumes no information about HB responses and predicts \( \zeta^HBP_t \). This prediction is not entirely reliable according to the performance of the prediction.
model reported in [13]. Therefore, most of the reserve prices are not successful, and the revenue of the impression is equal to $\zeta^HBP_t$. This method takes time to train the prediction models and a survival analysis model. Deriving a reserve price using these models is performed in real-time because no further training is needed after developing the models.

Fig. 5 shows the cumulative regret of DRL-PL and the benchmark methods. Regret is defined as the sum of the differences between the reserve price of a particular method and the value of $\zeta^HBP$ over all impressions. SA-PM has the highest cumulative regret that is compatible with the results of table IV in which the total revenue of this method is the lowest. The lines corresponding to H4-2 and MAB-TS show that their cumulative regrets are close to each other. As DRL-PL has the highest revenue, it also has the lowest regret, which is clearly observable in Fig. 5.

To explore the reserve prices of an individual impression, the reserve prices of DRL-PL and other methods are drawn in Fig. 6. The x-axis corresponds to the reserve price of DRL-PL, and the y-axis shows the reserve price of the benchmark methods. Fig. 6a shows that the reserve price of DRL-PL is slightly higher than H4-2 for most of the impressions. This figure also shows that if H4-2 has a better reserve price, the difference between the reserve prices of DRL-PL and H4-2 is high. Since in important impressions, the difference between $\zeta^HBP_t$ and $\zeta^AdX_t$ is large, regret is high if AdX’s auction fails to outbid the reserve price. This happens only for small reserve prices, and H4-2 rarely performs better than DRL-PL for high reserve prices. In Fig. 6b, DRL-PL and MAB-TS are compared at the impression level. Since the set of reserve prices is fixed for MAB-TS, horizontal lines are justified in this figure. Most of the points are below the line $y = x$, showing that the reserve prices of DRL-PL are higher than MAB-TS for the majority of the impressions. When the higher reserve price is unsuccessful, the revenue is $\zeta^HBP$ which is rather small. Finally, Fig. 6c shows the superior performance of DRL-PL in comparison with SA-PM. Except for a few impressions with small reserve prices, for all other impressions, the reserve prices of DRL-PL are higher than SA-PM. According to these figures, using DRL-PL can significantly improve the revenue of an ad publisher in RTB systems based on HB and AdX.

**E. Computational Burden**

Three parts of our proposed method require separate computational resources. The first part is the hyper-parameters optimization module. We use two machines with Central Processing Unit (CPU) and Graphics Processing Unit (GPU). The configurations of the machines are the same; however, their operating systems are Ubuntu and Windows 10. The CPU of each machine has four cores and eight logical processors with 2.80 GHz processing speed. The GPUs are Intel HD Graphics 630 with 8 GB memory, and the core speed is reported as 300 - 1150 (Boost) MHz. Two BO frameworks are run concurrently on these two machines, and each one has 12 hours budget. Hence, the hyper-parameters module with the current configuration takes 24 hours for testing candidate values. For Beta, Gaussian, and Gamma distributions, each iteration, including the RL training step, takes between 10 to 15 minutes. This value for MVN is considerably higher, and it rounds to 50 minutes most of the time.

The second part is running a single RL procedure for 120 thousand timesteps with the acquired values for learning rate, epoch number, batch size, and PPO clip rate. It takes as much time as running a single candidate value of MVN and finishes training in around 50 minutes. Finally, the price-setting step works by receiving item information, following the computation in a policy network, and providing the price. This process could be performed in real-time because the layers of the policy network are fixed, and a series of finite mathematical computations obtain the price.

**VII. Summary and Conclusion**

This paper presented a DRL pipeline for DP problems that automates the process of MDP modeling and hyper-parameters optimization. As a case study, we employed our DRL pipeline to derive the reserve prices of the impressions in the RTB system based on HBPs and AdX. Our results show that the expected revenue can be significantly increased by employing our DRL pipeline for adjusting the reserve prices. This achievement is very important for ad publishers who highly rely on the revenue of advertising.

Our DRL pipeline automatically explores the space of MDP modelings and hyper-parameters and results in the DRL configuration, which provides the highest aggregated reward on a dynamic pricing problem. Through these results, we learned that the expensive-to-evaluate configurations are the critical points in designing an automated DRL pipeline. Configurations associated with the neural network structure, action PDFs, and DRL algorithm highly alter the running time, requiring careful consideration. In addition, our automated DRL pipeline showed that the near-optimal configurations might be obtained in the early stages of meta-learning. Smartly setting the time budget for the DRL pipeline is an interesting direction for future research.

**ACKNOWLEDGMENT**

This work was supported by EU EUROSTARS (Project E! 11582). The authors would like to thank the Headerlift Team and the Triodor R&D Team from Azerion and Triodor in collecting the data for this research.
Fig. 6: Comparing individual reserve prices of DRL-PL vs. other benchmark methods.

REFERENCES


