Maximizing User Click - Optimizing Bid Advertising Using Reinforcement Learning and Deep Learning

Youssof Bendary, Keegan Veazey, Rajat Pratap Singh Bisht, Hamza Akmal Chaudary, Sarang Ajay Patel, Weijia Xiao

{bendary.yo, veazey.k, bisht.r, chaudary.h, patel.sarang, xiao.wei}@northeastern.edu

Northeastern University

December 5, 2024

Abstract

The evolution of online advertising has made real-time bidding (RTB) a key aspect of optimizing ad placements in competitive markets. This project explores a reinforcement learning-based approach to maximize user impressions while adhering to budget constraints, addressing the challenges of dynamic market conditions and the random, volatile nature of auction environments. By creating a simplified RTB environment and leveraging Q-Learning and Deep Q-Network (DQN) algorithms, our agent learns optimal bidding strategies that balance budget allocation with user priorities. The results demonstrate the potential of reinforcement learning to outperform traditional static optimization approaches, paving the way for data-driven advertising solutions.

Keywords— Q-Learning, DQN, Bidding Simulation

Introduction

The landscape of optimizing online advertising predominantly relies on real-time bidding [1]. To optimize advertisement placements, companies must address challenges such as dynamic market competition and auction volume while adhering to budget constraints. This project draws inspiration from prior research exploring reinforcement learning techniques for budget-constrained bidding, including Han et al., adapting their approaches to create a simplified RTB environment. Our goal is to use Q-Learning and DQN to train an agent to learn a bidding strategy that maximizes user impressions (ad clicks) within a set budget. Unlike prior studies focused on click-through rate or dynamic programming, we emphasize a single-agent system designed for practical adaptability in real-time contexts.

1 Related Work

In model-free ECPC [7], the authors propose a bidding strategy that leverages a combination of estimated conversion rate and batch reinforcement learning to optimize the ad bids dynamically on an hourly basis. The ECPC bidding is formulated as a Markov Decision Process (MDP). The method does not rely on model exploration and instead focuses on model-free reinforcement learning (RL), minimizing the extrapolation error by aligning policy actions with historical data. The MDP is characterized by states representing the auction environment, including bid prices, impressions, and budgets. The agent's actions correspond to bid adjustments and rewards are given by the number of clicks or impressions. The problem is tackled using Q-learning, where the Q-function $Q(s, a)$ estimates the expected future rewards for each state-action pair and is updated iteratively via the Bellman equation.

Further work by Di Wu et al. [2] explores a model-free RL approach specifically aimed at optimizing budget-constrained bidding. They use a variant of Deep Q-Network (DQN), where the Q-function is approximated using deep neural networks rather than a tabular method. This allows the algorithm to handle the high-dimensional state space typical in online auctions. The authors employ experience replay and target network stabilization techniques to mitigate instability during training. Their reward function is designed to maximize the number of impressions while adhering to budget constraints. Formally, the reward function r_t in their setting is given as:

$$r_t = I_t - \lambda \left(\frac{C_t}{B_t} \right),$$

where I_t represents the impressions obtained at time step t, C_t is the cumulative cost up to t, B_t is the total budget, and λ is a regularization parameter that penalizes excessive budget consumption.

A notable challenge in RTB systems is the stochastic nature of the environment, particularly the large state and action spaces resulting from combinatorial auction dynamics. Cai et al. [9] address this by leveraging dynamic programming for value function approximation, along with neural networks for policy learning. They use the policy iteration method, where the agent alternates between:

- **Policy evaluation**: Updating the value function $V(s)$ using the Bellman equation.
- **Policy improvement**: Adjusting the bidding strategy to maximize expected future rewards.

Their implementation benefits from techniques such as ϵ-greedy exploration which balance exploration and exploitation in learning optimal bidding strategies.

Since an ad impression has two states, the authors [8] consider clicks as binary variables and model clicks as Bernoulli trials, applying Bayesian inference with the beta distribution. This helps update click probability with real-time performance. This method uses both network-level and campaign-level data to provide more accurate bid recommendations.

The limitations of model-free RL are highlighted by the authors [9], who demonstrate that it can suffer from transition dynamics in large state spaces and its highly stochastic nature. Instead, they leverage the Markov Decision Process and dynamic programming to learn the optimal bidding policy for value function approximation. Neural networks are incorporated to handle large state spaces effectively.

2 Project Architecture

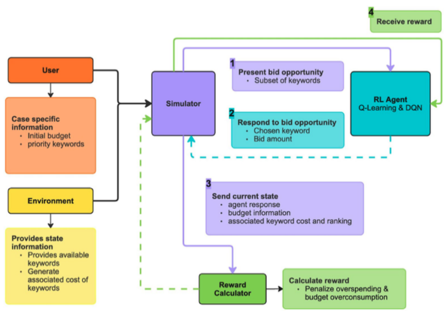

4 Receive reward
User
Case specific information
Initial budget
priority keywords
Environment
Provides state information
Provides available keywords
Generate associated cost of keywords
Simulator
1 Present bid opportunity
Subset of keywords
2 Respond to bid opportunity
Chosen keyword
Bid amount
RL Agent
Q-Learning & DQN
3 Send current state
agent response
budget information
associated keyword cost and ranking
Reward Calculator
Calculate reward
Penalize overspending & budget overconsumption

Figure 1: Project Workflow

Figure 1 illustrates the project workflow of the research process used in this project. The five key elements are as follows:

- **User**: Provides case-specific information, including the initial budget and priority keywords (keywords the user wants to target their bids for advertisements).
- **Environment**: Represents the external factors the system operates within and creates bidding cycle instances. The environment provides the available keywords and generates the associated costs for those keywords.
- **Agent**: Q-Learning or DQN reinforcement learning agent whose goal is to learn an optimal bid policy that balances the user's given budget and prioritizes case-specific keywords.
- **Simulator**: An intermediary between the user and the agent. It receives case-specific and state information from the environment and passes these to the agent as singular auction opportunities where a subset of keywords is available to bid on.
- **Reward Calculator**: Based on the agent's response to an auction (whether to place a bid, if so, chosen keyword and the price of the bid), the calculator evaluates the auction results and state information to calculate reward, penalizing overspending if applicable.

The key interaction in Figure 1 is the cycle between the agent, simulator, and reward calculator since the agent learns through the iterative process of making bids and receiving feedback from the calculator.

3 Model

3.1 Environment

Considering the scope of this project, simulating a hyper-realistic RTB environment would generate exponentially many states, causing a bottleneck. This was solved using statistical reasoning-based assumptions. These assumptions can also be considered hyperparameters for the learning model's environment constraints, which can be modified to fit the use case of any market sector.

3.1.1 Simplifying Assumptions

- Real-world keywords with semantic meanings are represented with placeholder keywords. This assumption allows the model to be generalized across diverse market sectors by mapping placeholder keywords to relevant semantic keywords, thus requiring minimal modifications to the project architecture.
- For maintaining simplicity and limiting the state-space, the 26 alphabets of the English language were chosen as placeholders. This assumption allows the comparison of the DQN agent and the Q-learning agent while minimizing the risk of state explosion to attain comparable and meaningful results.
- Instead of simulating multiple bids by a large subset of dummy users, a highest-bid generator is implemented in the environment. This simulates the current highest bid seen by the environment and is used to determine if the agent wins an auction. Because a large number of bids are placed on keywords in the real world, the bids follow a skewed normal distribution. It is assumed that bids are symmetric around a randomly generated mean value with a standard deviation equaling 20% of each keyword's respective mean value.

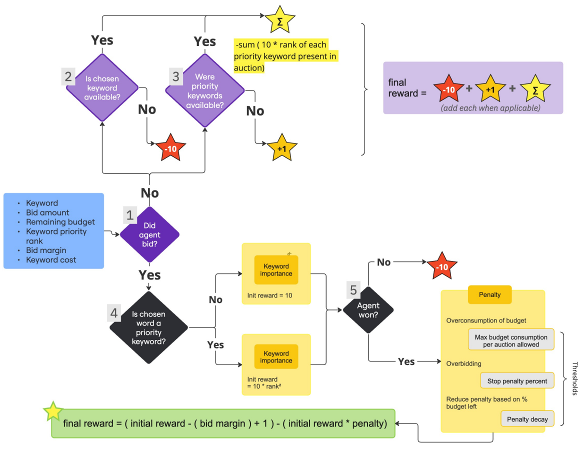
3.1.2 One Bidding Cycle

During a bidding cycle, the environment presents the opportunity to bid on 10 randomly selected keywords and provides the agent an option to either 'bid' or 'skip' the auction. If the agent chooses to bid on a keyword and wins, the environment returns the second-highest bid value seen by the environment along with the win flag set to True for reward calculation.

3.1.3 Simulator Instance

The simulator instance initializes a class object with a budget and a list of keywords ranked by the expected priority of a win. This class object communicates with the bidding environment, updates the current budget, and calculates rewards based on the results of each bidding cycle encountered. The simulator class object lasts for the entirety of an episode, which terminates when the remaining budget falls below a preset value or gets truncated due to multiple consecutive 'skip' actions.

3.2 Rewards



Yes
Yes
-sum (10 * rank of each priority keyword present in auction)
Σ
2
Is chosen keyword available?
No
-10
3
Were priority keywords available?
No
+1
final reward = -10 + +1 + Σ
(add each when applicable)
No
Keyword
Bid amount
Remaining budget
Keyword priority rank
Bid margin
Keyword cost
1
Did agent bid?
Yes
4
Is chosen word a priority keyword?
No
Keyword importance
Init reward = 10
Yes
Keyword importance
Init reward = 10 * rank²
5
Agent won?
No
-10
Yes
Penalty
Overconsumption of budget
Max budget consumption per auction allowed
Overbidding
Stop penalty percent
Reduce penalty based on % budget left
Penalty decay
Thresholds
final reward = (initial reward - (bid margin) + 1) - (initial reward * penalty)

Figure 2: Reward Calculator System for Real-Time Bidding Context

In reinforcement learning, the returned reward after an agent's action serves as feedback that guides an agent's learning process. The agents described in Section 3.3 (Q-Learning and DQN) work to maximize these rewards. Therefore, the rewards for this project needed to have high values for desirable behaviors, such as winning priority keywords without overspending, while also having low or negative rewards for undesirable behaviors, such as losing high-priority auctions. Figure 2 illustrates the reward calculator system implemented to train agents in a real-time bidding context.

The reward is calculated using the metadata returned from the simulator after an agent takes an action and is passed through penalty logic to penalize overspending (defined as overconsumption of the current budget value or overbidding):

- **Overconsumption**: When the agent bids using over 50% of their remaining budget, the final reward is penalized by 20% of the inital reward.
- **Overbidding**: When the agent bids greater than 150% of the keyword's actual cost, the final reward is penalized by 20% of the inital reward.
- If both cases are met, the final reward is penalized by 40% of the inital reward. The total penalty decays when the `stop_penalty_percent` threshold (default: 25%) is reached. At this point, the penalty is decayed by a parameter `stop_penalty_decay` (default: 0), meaning the penalty is completely removed.

This logic ensures high rewards for efficient spending while allowing the agent to use the allocated budget from the user fully.

3.2.1 Reward Scenarios

- **Winning an Auction**: The final reward is calculated as shown in Figure 2.
- **Skipping an Auction**: If no priority keyword is available, a small positive value (+1) is added to the cumulative reward to teach the agent that skipping such auctions is acceptable. But if there is at least one priority keyword, then the reward will penalize more as more priority keywords are presented and/or higher ranks they have.
- **Invalid Keyword Selection**: If the agent selects an invalid keyword (exists in the environment but not in the current auction subset), the reward is penalized by -10.
- **Losing an Auction**: The agent gets a constant penalty of -10 no matter if the keyword it decides to bid on is a priority keyword or not, to avoid scaring the agent away from trying out to bid on priority keywords.

The final reward in each scenario is the summation of the calculated values, guiding the agent to optimize bidding strategies effectively.

3.3 Q-Learning Agent

Algorithm 1: Q-learning Agent with Epsilon-Greedy Action Selection for Auction Bidding

```
Result: Optimal Q-values for bidding strategy
Initialize $Q\_table \leftarrow \emptyset$;
Initialize priority keywords and bids $bids[keyword] \leftarrow 100.0\ \forall\ keyword$;
Initialize $\epsilon, \gamma, \alpha$, actions;
foreach episode in training episodes do
    Reset simulator and initialize state $S$;
    Reset $bids \leftarrow \{keyword : 100.0\ \forall\ keyword\}$;
    while not done do
        Compute $S_{\text{hash}} \leftarrow (S)$;
        if $S_{hash} \notin Q\_table$ then
            $Q\_table[S_{\text{hash}}] \leftarrow$ zeros($|actions|$);
        end
        if $random() < \epsilon$ then
            Select random action $a \in [0, |actions| - 1]$;
        else
            Select action $a \leftarrow \arg\max_a Q\_table[S_{\text{hash}}][a]$;
        end
        Select keyword $K$ with highest priority available in current step;
        $bid\_amount \leftarrow max(0, bids[K] + actions[a])$;
        Execute $bid\_amount$ in simulator, receive reward $r$, and next state $S'$;
        Compute $S'_{\text{hash}} \leftarrow (S')$;
        if $S'_{hash} \notin Q\_table$ then
            $Q\_table[S'_{\text{hash}}] \leftarrow$ zeros($|actions|$);
        end
        Update: $Q\_table[S_{\text{hash}}][a] \leftarrow Q\_table[S_{\text{hash}}][a] + \alpha \times (r + \gamma \max_{a'} Q\_table[S'_{\text{hash}}][a'] - Q\_table[S_{\text{hash}}][a])$;
        $S \leftarrow S'$;
    end
    Decay $\epsilon$;
end
```

The Q-learning agent operates under the assumption that it has no prior knowledge of the auction environment or other bidders' strategies, relying solely on the feedback provided after each round. The agent's state is defined by the remaining budget and the bidding keyword in that round, and it selects actions to adjust bid values within predefined discrete increments. The agent maintains a Q-table, which maps hashed states—generated using the budget and available keywords during an auction—to Q-values for each action. This table represents the agent's learned policy over time. The decision-making process involves balancing exploration, where actions are chosen randomly, and exploitation, where the best-known action from the Q-table is selected, governed by an epsilon-greedy strategy with a decaying epsilon. The agent interacts with the auction simulator by bidding on the highest-priority keyword from the available list or skipping the round if none match its priority. Feedback is used to update the Q-values using the temporal difference (TD) update rule, which incorporates the reward received and the estimated value of the next state. Over time, epsilon decays to encourage a shift from exploration to exploitation as the agent learns the optimal bidding strategy. This approach, enhanced by efficient state hashing, allows the agent to handle the large state-action space characteristic of multi-round auction scenarios while adhering to budget constraints.

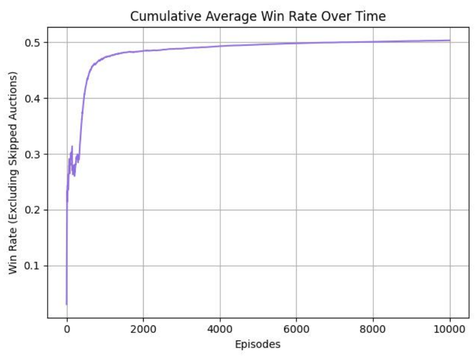
3.4 DQN Agent

Algorithm 2: Multi-task DQN Agent with Experience Replay and Epsilon-Greedy Action Selection

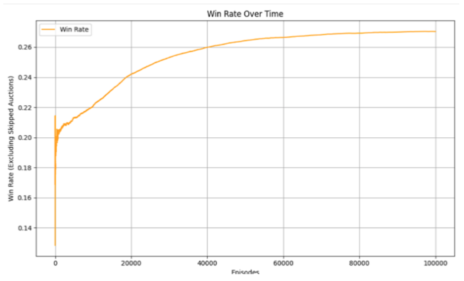
```
Result: Optimal bidding strategy (keyword selection and bidding price)
Initialize replay memory queue buffer $R$ to capacity $N$;
Initialize prediction network $Q_P$ with random weights $\theta_P$;
Initialize target network $Q_T$ with random weights $\theta_T = \theta_P$;
foreach episode in replay buffer initialization episodes do
    Select a random action $a$ (keyword selection or decision to not bid in the current auction);
    Select a random bidding price $b$ within the current budget;
    Run an auction step in the simulator using the decision of whether to bid, and if so, for what keyword and price;
    $obs \leftarrow$ observations (current available keywords and budget);
    $reward \leftarrow$ reward from taking the specific action;
    $hcb \leftarrow$ highest competitor bid;
    $done \leftarrow$ whether reached a termination state that the agent can no longer place a bid;
    $obs_{new} \leftarrow$ new observations after the auction step;
    Store transition ($obs$, $a$, $reward$, $hcb$, $done$, $obs_{new}$) in $R$;
    Reset the simulator if a termination state is reached;
end
Reset the simulator;
for step $s = 0, 1, 2, ...$ do
    if $random() \leq \epsilon$ then
        Select a random keyword selection / not bid action $a_s$ and a random price $p_s$ ;
    else
        Select an action $a_s \leftarrow \arg\max Q_P(obs_s)[bid\ action\ head]$;
        Choose a price $p_s \leftarrow Q_P(obs_s)[price\ prediction\ head]$;
    end
    Run an auction step in the simulator using $a_s$ and $p_s$;
    Reset the simulator if a termination state is reached;
    Store transition ($obs_s$, $a_s$, $reward_s$, $hcb_s$, $done_s$, $obs_{new_s}$) in $R$;
    Sample random minibatch of transitions ($obs_i$, $a_i$, $reward_i$, $hcb_i$, $done_i$, $obs_{new_i}$) from $R$
      $reward_{P_i} = Q_P(obs_i)[bid\ action\ head][a_i]$;

      $$y_i = \begin{cases} reward_i & \text{if episode terminates at step } j = i + 1 \\ reward_i + \gamma \max Q_T(obs_j)[bid\ action\ head] & \text{otherwise} \end{cases} \quad (1)$$

    $p_{win_i} = hcb + 1$;
    $p_i = Q_P(obs_i)[price\ prediction\ head]$ ;
    Perform a gradient descent step on the loss between $reward_{P_i}$ and $y_i$ , add the loss between $p_i$ and $p_{win_i}$ for
      auctions that the agent decides to bid, with respect to the prediction network parameters $\theta_P$;
    Reset $\theta_T = \theta_P$ every S steps;
    Terminate the loop when the number of training episodes is reached.
end
```

The DQN agent utilizes the Double Q-Learning concept (having 2 DQNs during training) to mitigate the Q-value overestimation issue and provide more stability[10]. A main novelty in the agent model is incorporating a multi-task learning model architecture, where one output head decides the bid action (whether to bid, if so, which keyword to bid on), and the other predicts a bidding price that aims to win the auction.

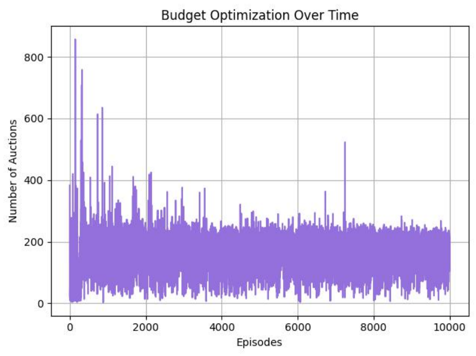
4 Results



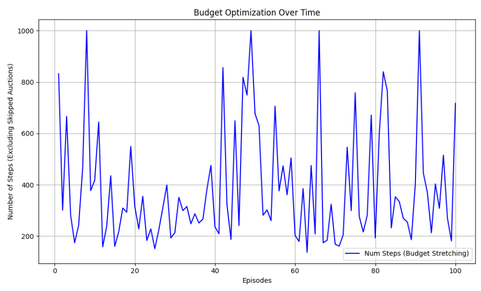
(a) Average Win Rate for the DQN agent



(b) Average Win Rate for the Q agent



(c) Budget Optimization for DQN agent



(d) Budget optimization for Q agent

Table 1: Comparison of DQN Agent performance with Q Agent performance.

The plots (b) and (d) above correspond to the Q-Learning agent. Across both plots, the x-axis represents episodes. Plot (b) shows the cumulative average win rate (win rate defined as total wins divided by total auctions participated in). Plot (d) depicts the average win rate for the Q agent and tracks the agent's win rate across 10,000 episodes. The win rate steadily increases from approximately 0.14 to 0.26, demonstrating the agent's ability to adapt to the environment and improve its bidding strategy over an extended training period. Plot c depicts budget optimization over time, and evaluates the number of auctions participated in by the agent on the y-axis. The number of auctions across episodes fluctuates a lot and does not stabilize as in the DQN.

The plots (a) and (c) above correspond to the DQN agent performance. The win rate increases quickly in the first thousand episodes and stabilizes around 0.45, increasing at a linear rate to 0.51 by the 10,000th episode. Plot (c) depicts the budget optimization over time. After the first thousand episodes, the number of episodes condenses to a range of 150-200 episodes.

5 Discussion

The results clearly demonstrate the efficacy of reinforcement learning techniques in optimizing bidding strategies for real-time bidding (RTB) environments. The performance of the Deep Q-Network (DQN) agent, as illustrated in the plots, highlights its ability to effectively adapt to the dynamic nature of online auctions. The cumulative average win rate for the DQN agent shows a rapid improvement early in the training process, stabilizing around a win rate over 50%. This stabilization reflects the agent's capacity to consistently place effective bids in a competitive environment. Conversely, the Q-learning agent demonstrates a slower and more gradual improvement in win rate, ultimately stabilizing at a lower threshold. This difference underscores the advantages of using neural network-based approaches, such as DQN, to approximate value functions in high-dimensional state spaces, where traditional tabular methods like Q-learning face limitations.

The budget optimization plots further illustrate the comparative strengths of the two approaches. The DQN agent exhibits a more stable and predictable pattern in budget utilization over episodes, which is indicative of its ability to balance exploration and exploitation while adhering to budget constraints. On the other hand, the Q-learning agent shows higher variability in budget optimization, suggesting challenges in managing the trade-off between exploring new bidding strategies and exploiting learned ones. Despite this variability, the Q-learning agent's steady progression highlights the adaptability of simpler models when applied to constrained environments, albeit with reduced efficiency compared to DQN.

Together, these results validate the framework's design and the effectiveness of reinforcement learning in managing complex, real-time bidding scenarios. The comparative analysis between DQN and Q-learning also provides valuable insights into the scalability of such approaches. While DQN demonstrates superior performance, the Q-learning agent's results suggest that even resource-constrained implementations can achieve meaningful outcomes, making them suitable for less computationally intensive applications.

5.1 Future Work

For future work, the framework can be extended to incorporate multi-agent systems, where multiple agents with competing interests participate in the same bidding environment. This would allow for the exploration of cooperative and adversarial dynamics in auction scenarios. Additionally, integrating real-world keyword mappings and user behavior models would enhance the realism and applicability of the model to specific market sectors. Exploring transfer learning approaches to adapt pre-trained agents to new environments or campaigns without retraining from scratch could further improve efficiency. Lastly, incorporating advanced reward mechanisms, such as hierarchical or multi-objective reward systems, may yield insights into optimizing complex goals, such as balancing revenue maximization with user engagement. These extensions would pave the way for more robust and scalable real-time bidding solutions.

6 Joint Team Distribution

Project team members were assigned specific components: Rajat (environment), Hamza (bid simulator), Keegan (rewards), Sarang (Q-Learning), Weijia (DQN), and Youssof (main file and logger). They all actively participated in weekly meetings, pushed code updates, supported each other, and collaborated on the presentation and paper.

7 Code Availability

Here is our public Github repo (click me) : `https://github.com/rajatpratapbisht/Ad_Bidding_RL_Agent` with all the codes, results, model configurations, and replication instructions!

References

[1] Cai, Han, et al. "Real-time bidding by reinforcement learning in display advertising." Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, 2 Feb. 2017. Available: `https://doi.org/10.1145/3018661.3018702`.

[2] Wu, Di, et al. "Budget constrained bidding by model-free reinforcement learning in display advertising." Proceedings of the 27th ACM International Conference on Information and Knowledge Management, 17 Oct. 2018. Available: `https://doi.org/10.1145/3269206.3271748`.

[3] S. Li, C. Yuan, and X. Zhu, "Optimizing Enhanced Cost Per Click via Reinforcement Learning Without Exploration," 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, 2021, pp. 1-8. doi: `https://doi.org/10.1109/IJCNN52387.2021.9534126`.

[4] Skaik, Y. (2015). The bread and butter of statistical analysis "t-test": Uses and misuses. *Pak J Med Sci*, 31(6), 1558-1559.

[5] Fisher, R. A. (1992). The arrangement of field experiments. In *Breakthroughs in statistics: Methodology and distribution* (pp. 82-91). New York, NY: Springer New York.

[6] Carroll, R. J., & Schneider, H. (1985). A note on Levene's tests for equality of variances. *Statistics & Probability Letters*, 3(4), 191-194.

[7] Drolias, B. (2007). *Pay-Per-Click: The Complete Guide*. Lulu. com.

[8] Mazyavkina, N., Sviridov, S., Ivanov, S., & Burnaev, E. (2021). Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134, 105400.

[9] Student. (1908). The probable error of a mean. *Biometrika*, 1-25.

[10] Mnih, V., Kavukcuoglu, K., Silver, D. *et al.* (2015). Human-level control through deep reinforcement learning. *Nature* 518, 529–533. https://doi.org/10.1038/nature14236