PulseRL: Enabling Offline Reinforcement Learning for Digital Marketing Systems via Conservative Q-Learning

Luckeciano Melo[*] [†] Deep Learning Brazil
Luana Martins[*] Deep Learning Brazil
Bryan Oliveira[*] Deep Learning Brazil
Bruno Brandão[*] Deep Learning Brazil

Douglas W. Soares Acordo Certo
Telma Lima Deep Learning Brazil, Federal University of Goiás, Brazil

Abstract

Digital Marketing Systems (DMS) are the primary point of contact between a digital business and its customers. In this context, the communication channel optimization problem poses a precious and still open challenge for DMS. Due to its interactive nature, Reinforcement Learning (RL) appears as a promising formulation for this problem. However, the standard RL setting learns from interacting with the environment, which is costly and dangerous for production systems. Furthermore, it also fails to learn from historical interactions due to the distributional shift between the collection and learning policies. For this matter, we present PulseRL, an offline RL-based production system for communication channel optimization built upon the Conservative Q-Learning (CQL) Framework. PulseRL architecture comprises the whole engineering pipeline (data processing, training, deployment, and monitoring), scaling to handle millions of users. Using CQL, PulseRL learns from historical logs, and its learning objective reduces the shift problem by mitigating the overestimation bias from out-of-distribution actions. We conducted experiments in a real-world DMS. Results show that PulseRL surpasses RL baselines with a significant margin in the online evaluation. They also validate the theoretical properties of CQL in a complex scenario with high sampling error and non-linear function approximation.

1 Introduction

Digital Marketing Systems (DMS) are essential tools to scale customer acquisition and relationship management. These online systems are primarily responsible for providing communication channels between the business and its customers, which is essential to understand the customer behavior during its journey.

In addition to providing important data for business analysis, these systems are very powerful tools for automated, data-driven decision-making. Indeed, channel optimization poses a precious and open challenge for current digital businesses [22, 1]: How should a DMS select the best communication actions to maximize customer satisfaction (and lifetime value) while maintaining the operational costs as low as possible?

[*] denotes equal contribution.

[†] Correspondence to: luckeciano@gmail.com

2nd Offline Reinforcement Learning Workshop at Neural Information Processing Systems, 2021.

Given this context of sequential decision making, Reinforcement Learning (RL) appears as a promising problem formulation for channel optimization in Digital Marketing Systems. Nevertheless, the standard RL training setting is very costly in high-dimensional problems since the agent must explore actions to build its dataset. Furthermore, the action-reward cycles can take days to happen in a DMS, which results in a very long time to build a complete customer trajectory. These challenges make the application of RL in production systems unfeasible for various business scenarios.

In this work, we present PulseRL, an alternative for the standard RL setting in Digital Marketing Systems which enables offline training techniques. In Offline Reinforcement Learning [34, 33, 12], the agent is trained by using historical interactions from previous system versions (i.e., other policies) without any online interaction prior to the deployment. Although this scenario imposes many algorithmic challenges [16], we mitigate them by building a channel optimization agent upon the pessimistic principle [4] using Conservative Q-Learning (CQL) [15], which reduces the effect of the distributional shift between the dataset and the learned policy.

We conducted comparison experiments in a real-world, production Digital Marketing System with millions of users. They showed that PulseRL stabilizes offline RL at scale and surpasses the performance of other agents based on standard RL algorithms or Imitation Learning by a significant margin. Therefore, our method poses a powerful solution for enabling RL in production systems, especially for DMS. We also released a new dataset to ensure reproducibility and as another contribution to the research community.

2 Related Work

Reinforcement Learning has been used in various areas that the Customer Relationship Management (CRM) problem contains regarding customer interactions [6, 14, 31, 29]. A common use of RL in CRM is in recommender systems, where an algorithm attempts to optimize which products or services will extract a positive response from the user [31, 29, 23, 24]. Though recommendations can fall under marketing systems, our approach focuses more on when to contact users to improve conversion rates. More specifically, we apply our work to the debt collection problem, analogous to a marketing system. To formulate it as a sequential problem, we rely on Markov Decision Processes (MDPs), which are the mathematical formulation of RL and focus on the sequential nature of customer interactions [21, 11, 17, 2, 18].

The use of RL in this problem is by no means a novel idea. Both Abe et al. and Miller et al. use Q-Learning with advantage updates to solve an MDP and improve tax collection. In these previous studies [17, 2, 18], the dimensionality of the problem becomes a pressing issue given the large amount of information available regarding the debtor and the debt itself. Abe et al. describe how the “feature space" is large enough to require function approximation and uses linear regression. In our case, we used neural networks with non-linearities as function approximators to deal with the large feature space.

The work that is closest to ours is from Geer et al.. Their work attempts to estimate the probability of debt repayment to decide whether or not to notify the debtor. The decision of notifying by phone call is much close to ours, of a text message. Geer et al. had access to information regarding both the debtor and the debt itself. They assembled their features in two sections, a debtor specific, with more established information, such as income and amount owed, and a historical section regarding how long ago was the last payment and the last notification [10]. This specific way of compiling the features is noteworthy as it allows for historical information to be compressed and considered at every step. The main difference between Geer et al.’s work and ours is that they directly computed probabilities and decided by looking a single step ahead, whilst ours uses RL trained with entire sequences of user interactions.

Lastly, our work has a particular focus on Offline RL. A large body of recent work developed theoretical understanding and algorithmic improvements [4, 12, 33, 34], but they performed empirical validation in toy problems or controlled benchmarks that steer away from production systems. On the other side, real-world applications of Offline RL typically rely on myopic methods such as contextual bandits [27, 7] or focus only on the off-policy evaluation aspect [30, 28], not on the complete Offline RL training. Our work merges both worlds by enabling the recent CQL algorithm in a real-world application.

3 Preliminaries

In this section, we introduce the notation and formalize the idea of Offline Reinforcement Learning for debt notification in Digital Marketing Systems. We also formalize the Conservative Q-Learning algorithm under the Pessimism Framework.

3.1 Offline Reinforcement Learning

The RL problem formulation is formalized as a Markov Decision Process (MDP) $\mathcal{M}$ [26]. In an MDP, an *agent* is whatever form that makes decisions; at a time step t, it receives from the environment *state* $\boldsymbol{s} \in \mathcal{S}$ and chooses from a finite set of *actions* $\boldsymbol{a} \in \mathcal{A}$ by following a policy π. The environment changes, following a dynamics $T : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, \infty)$, representing the probability density of the next state. Finally, the environment sends a feedback *reward* $r : \mathcal{S} \times \mathcal{A} \to [-R_{max}, R_{max}]$. The ultimate objective of an RL agent is to maximize the cumulative reward, i.e., $\max \mathbb{E}[\sum_{t=0}^{T} \gamma^t r(\boldsymbol{s}, \boldsymbol{a})]$, where γ is a discount factor to account for delayed rewards.

In the offline setting, we train the agent by using a fixed dataset $\mathcal{D} = \{(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s'})\}$ of historical transitions from a collection policy π_c (or a diverse set of policies), inducing an empirical MDP $\overline{\mathcal{M}}$. This is different from the standard RL training where we sample direct interactions between the agent and the environment. There are no guarantees of policy optimality for the real MDP $\mathcal{M}$ in the offline scenario [5, 13]. Therefore, the goal is to find the best possible policy given the constraint of the fixed dataset [34].

3.2 Conservative Q-Learning and the Pessimism Principle

Standard value-based, off-policy RL algorithm often leads to poor performance in the offline setting. Firstly, they tend to overfit the fixed datasets [8]. Secondly, they also fail to bootstrap out-of-distribution actions [9] due to the distribution shift between the collection and training policies. These methods express these failures by overestimating the value function, being too "optimistic" [15]. Therefore, we need algorithms that can follow the *pessimism principle* [4]: we should choose the policy which acts optimally (given the dataset constraint) in the worst-case scenario.

Conservative Q-Learning (CQL) implements this principle by learning a conservative Q-function such that the expected value of a policy under this Q-function is lower-bounded. The learning process incorporates this lower bound with mathematical improvement guarantees. Next, we will show the optimization targets from CQL and present the theoretical guarantees regarding this conservatism property during evaluation and training.

Policy Evaluation: For evaluation, CQL trains the Q-function by performing the iterative process in Equation 1:

$$\hat{\mathcal{Q}}_{t+1} \leftarrow \arg\min_{\mathcal{Q}} \alpha \cdot \mathbb{E}_{\boldsymbol{s}\sim\mathcal{D},\boldsymbol{a}\sim\mu(\boldsymbol{a}|\boldsymbol{s})}[\mathcal{Q}(\boldsymbol{s},\boldsymbol{a})] + \frac{1}{2}\mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{s'}\sim\mathcal{D}}\Big[\big(\mathcal{Q}(\boldsymbol{s},\boldsymbol{a}) - \hat{\mathcal{B}}^{\pi}\hat{\mathcal{Q}}_t(\boldsymbol{s},\boldsymbol{a})\big)^2\Big] - \alpha \cdot \mathbb{E}_{\boldsymbol{s}\sim\mathcal{D},\boldsymbol{a}\sim\hat{\pi}_c(\boldsymbol{a}|\boldsymbol{s})}[\mathcal{Q}(\boldsymbol{s},\boldsymbol{a})], \tag{1}$$

where $\mathcal{Q}$ is the Q-function, α is a positive trade-off factor (hyperparameter), $\mu(\boldsymbol{a} \mid \boldsymbol{s})$ is a particular distribution of state-action pairs, and $\hat{\mathcal{B}}^{\pi}$ is an empirical Bellman operator that only backs up a single sample. The first two terms of Equation 1 are enough to ensure pessimism [15]. Nevertheless, using them solely as optimization targets would result in a point-wise lower-bounded Q-function, causing higher under-estimation. Instead, the additional term to maximize Q-value under the data distribution will tighten this bound while still ensuring the lower bound only in the expectation of this Q-function. We refer to [15] for the formal description of this lower bound, alongside the detailed assumptions and full proof.

Policy Learning: For learning, CQL chooses, at each iteration, the $\mu(\boldsymbol{a} \mid \boldsymbol{s})$ to approximate the policy that maximizes the current Q-function. A particular and convenient choice of learning target also adds a regularizer function $\mathcal{R}(\mu)$, as shown in Equation 2:

$$\min_{\mathcal{Q}} \max_{\mu} \alpha \cdot \mathbb{E}_{\boldsymbol{s}\sim\mathcal{D},\boldsymbol{a}\sim\mu(\boldsymbol{a}|\boldsymbol{s})}[\mathcal{Q}(\boldsymbol{s},\boldsymbol{a})] + \frac{1}{2}\mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}'\sim\mathcal{D}}\left[\left(\mathcal{Q}(\boldsymbol{s},\boldsymbol{a}) - \hat{\mathcal{B}}^{\pi}\hat{\mathcal{Q}}_t(\boldsymbol{s},\boldsymbol{a})\right)^2\right] - \alpha \cdot \mathbb{E}_{\boldsymbol{s}\sim\mathcal{D},\boldsymbol{a}\sim\hat{\pi}_c(\boldsymbol{a}|\boldsymbol{s})}[\mathcal{Q}(\boldsymbol{s},\boldsymbol{a})] + \mathcal{R}(\mu). \tag{2}$$

If we select $\mathcal{R}(\mu) = -D_{KL}(\mu \mid\mid \rho)$, where $\rho(\boldsymbol{a} \mid \boldsymbol{s})$ is a prior distribution, we can derive that $\mu(\boldsymbol{a} \mid \boldsymbol{s}) \propto \rho(\boldsymbol{a} \mid \boldsymbol{s}) \cdot \exp(\mathcal{Q}(\boldsymbol{s},\boldsymbol{a}))$. Furthermore, if we set $\rho = \mathcal{U}(\boldsymbol{a})$, Equation 2 simplifies to Equation 3:

$$\min_{\mathcal{Q}} \alpha\mathbb{E}_{\boldsymbol{s}\sim\mathcal{D}}\left[\log\sum_{\boldsymbol{a}}\exp(\mathcal{Q}(\boldsymbol{s},\boldsymbol{a})) - \mathbb{E}_{\boldsymbol{s}\sim\mathcal{D},\boldsymbol{a}\sim\hat{\pi}_c(\boldsymbol{a}|\boldsymbol{s})}[\mathcal{Q}(\boldsymbol{s},\boldsymbol{a})]\right] + \frac{1}{2}\mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}'\sim\mathcal{D}}\left[\left(\mathcal{Q}(\boldsymbol{s},\boldsymbol{a}) - \hat{\mathcal{B}}^{\pi}\hat{\mathcal{Q}}_t(\boldsymbol{s},\boldsymbol{a})\right)^2\right]. \tag{3}$$

With this particular choice of optimization target, we can show that CQL learns lower-bounded policy values at each iteration (assuming that the policy changes slowly). We can also derive nice properties regarding the learning updates, such as "gap-expandability" (which helps mitigate action distributional shift) and safe policy improvement guarantees. For detailed explanations, we refer to [15].

4 PulseRL: Formulation and Architecture

In this section, we present PulseRL's MDP formulation along with its central system components, including data processing, offline pre-training, online fine-tuning, inference, and monitoring pipelines. PulseRL provides the necessary tooling to evaluate the properties of the CQL framework in real-world production systems and at the scale of millions of users.

4.1 MDP Formulation

In the Debt Notification problem, users' evaluation and activation happen daily. Figure 1 illustrates this interaction. At every iteration, we aggregate information about every user and debt along with situational data to generate an observation. From this, we extract meaningful features to estimate the state that will feed the agent. We describe the state space in the Appendix.

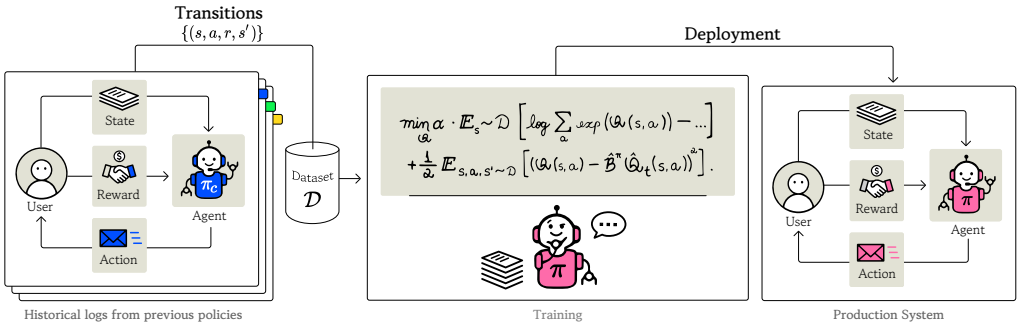

Figure 1: An illustration of how PulseRL works as an offline RL-based system. Firstly, it builds an offline dataset from (potentially unknown) collection policies π_c. Then, PulseRL trains its model by maximizing historical rewards using CQL training, resulting in the final policy π. Then, we deploy this agent to the production environment, and finally interacts with the real MDP.

Then, the agent chooses an action (activate each user or not). We send these decisions to the notification system, which will apply business constraints to them. These are internal "rules" that compose the environment dynamics. One rule example is: "the notification system shall not notify a user for two consecutive days.".

We commit the daily transitions the next day. We use the user, debt, and situational data changes to compose the reward and next observation for each user. We describe the reward function and its business explanation in the Appendix. Finally, we aggregate this transition with the previous historical logs to build the offline dataset. The agent behind those logs can be any decision-making system (e.g., supervised learning models, previous RL agents, or even humans).

We use the offline dataset to train the PulseRL agent using CQL. We generally pre-train the policies using Imitation Learning for other RL baselines, since starting with a random policy is unfeasible for the business scenario. We evaluate these models with offline metrics based on counterfactual estimations [7].

As the last step, we deploy the model to production. In this stage, the system follows the same daily interaction cycles, creating a new version of the dataset of historical logs. We then fine-tune the agents using the RL algorithms. However, each algorithm will use the logs differently for this task. On-policy methods require logs from the current policy to perform updates, while off-policy allows small replay buffers. For offline methods, there are no restrictions and we fine-tune using larger buffers.

4.2 System Design

We set up the system as a scheduled pipeline, running jobs for database cleanup, user attribution, ETL, inference, and evaluation every day. We represent job tasks as a Luigi [3] workflow. Luigi is a Python library that manages task execution by defining a dependency graph between tasks. It also makes sure that each task only runs once and, whenever possible, in parallel. In the remainder of this subsection, we describe each pipeline job separately. Figure 2 illustrates the system pipeline.

Firstly, we store all the tables containing user information, debts, events, and reward logs in a data warehouse. Then, an ETL job processes these tables to generate the buffer of transitions. We define the tasks from this job by using the date as a parameter to avoid redundant preprocessing of the historical dataset every day. This way, Luigi can detect existing runs and preprocess only the new data. This parametrization enables parallel daily data processing, which is horizontally scalable and allows us to process long-term historical datasets in a few minutes.

Since we use Ray [20] for distributed training, we need to convert the tabular data to a format Ray algorithms understand. Using SQL, we perform preprocessing operations like missing data filling, normalization, and aggregation on the data warehouse side. We create a Ray Dataset using the query results. Finally, we store it in a data lake. This step defines how the available data composes observations, subsequent observations, rewards, and actions - complete RL transitions. For that reason, different versions of this step may exist, which can be used by MDP versions as needed.

The training job uses data generated by the data engineering step directly from the data lake. After performing a training session, the pipeline saves a checkpoint and evaluates the model. Finally, we save all the generated artifacts to the data lake. This step may define different training algorithms and hyperparameters, including dataset versions from the previous step. We consider each version defined by this step as a candidate model for inference.

We select candidate models for inference after training and evaluation. At inference time, we generate observations using the same logic we use in the dataset generation step. The production models then decide whether to activate each debtor, writing results back to the data warehouse. The pipeline generates logs, charts, and metrics. A monitoring service uses them to compute model performance and business metrics. We monitor all steps of the pipeline closely to ensure all components are working as expected.

5 Experimental Results and Analysis

In this Section, we build an experimental setting to evaluate how much the nice properties regarding policy evaluation and training hold in a real-world scenario and at scale, despite sampling error and function approximation. We also establish a relationship between these properties and the practical consequences in the production system. In the experimentation, we used Acordo Certo Environment

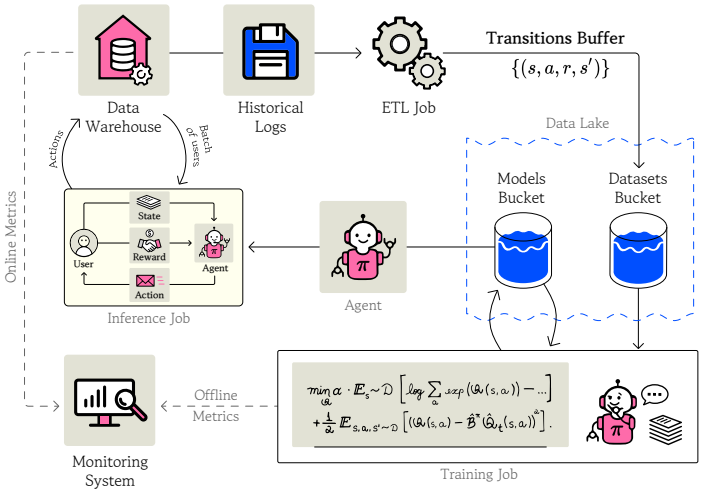

Figure 2: Illustration of PulseRL's system pipeline. We compose it with different big data storage models, a data transformation engine, a task manager, and specialized microservices for training and inference, which ensures scalability for handling millions of users on a daily basis. It also provides version control for source code, dataset, MDP's and RL agents.

as the Digital Marketing System. We describe it in Appendix A. We also released the source code for our experiments [3].

Dataset. We organized an experimental dataset carefully designed to not break any data privacy requirements from users nor partners. It has approximately 78 GB of compressed data in parquet files. We are releasing[4] it entirely as another contribution from our work to foster research in DMS, especially for offline RL agents in production systems. We detail the dataset composition in the Appendix.

For the experimental setting, we used a subsample of this dataset, where we selected 500,000 users to make it balanced and reduce the sparsity of the reward signals. We also applied the data processing schedule described in the previous Section and released this other dataset[5] to facilitate reproducibility.

Evaluation Metrics. We conducted both offline and online evaluations. We used offline evaluation for hyperparameter tuning and model selection. In this context, We computed off-policy estimators [7]: Importance Sampling (IS) and Direct Method (DM). We computed these metrics over a validation dataset of historical logs. We composed this dataset with 10,000 users randomly selected from the database.

We guided our model selection primarily by DM metric. IS metrics presented a considerable variance, mainly because the reward signals are significantly delayed and sparse in the context of the problem. DM metric was more stable and simpler to interpret, although biased. As future work, we consider designing a more robust offline evaluation method that presents less biased and variant estimations and a better correlation with online metrics.

In terms of online metrics, we evaluated agents' performance on success rates. Given a business goal k, we computed the conversion rate on k by using Equation 4:

$$conversion_k = \frac{\sum e_k}{n_{actions}}, \tag{4}$$

where $\sum e_k$ and $n_{actions}$ represent the sum of all events from a business goal k and the number of actions (notifications) executed by the agent, respectively. We considered two primary business goals:

[3] `https://github.com/dlb-rl/pulse-rl`

[4] Dataset link: `https://bit.ly/ac-dataset` ("raw" directory)

[5] Dataset for the experiments: `https://bit.ly/ac-dataset` ("processed" directory)

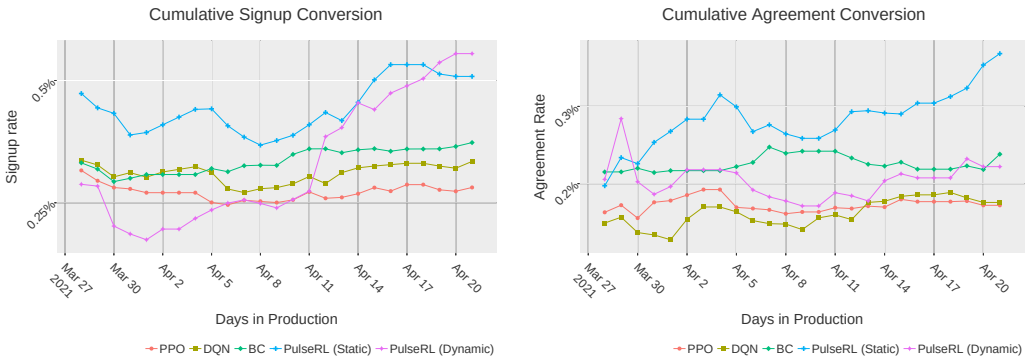

Figure 3: Results from online evaluation in the production system.

signup (when the user registers in the platform) and agreement (when the user consolidates a debt negotiation).

We started online evaluation with around 8,000 users for each model in production. Since the CQL-based models proved to be less expensive than the others, we increased the number of users they evaluated to 20,000 users.

Baselines. We compared PulseRL with three representative baseline methods:

- **BC**. Behavioral Cloning (BC), derived from MARWIL [32] implementation, aims to imitate the collected historical data's behavior policy.
- **DQN**. Deep Q-Network (DQN) [19], an action-value method that learns and selects actions based on the estimated value of an action in a particular state.
- **PPO**. Proximal Policy Optimization (PPO) [25], a policy gradient method where the goal is to learn a policy that can select actions without consulting the value function directly.

Start learning from random actions is very costly at scale. Therefore, both DQN and PPO agents deployed to production were pre-trained using the BC method. With this, they start closer to the behavior policy and thus mitigate activation cost and learning time issues.

5.1 Results and Analysis

As part of our empirical validation, we analyzed the following questions:

- **How does PulseRL perform in comparison with online RL algorithms in a production system?** – This question validates our hypothesis that offline RL is a better alternative than standard RL algorithms for production systems.
- **How is PulseRL compared to other methods in terms of distribution shift from Out-Of-Distribution (OOD) actions in production**? – This question validates how the conservative behavior of CQL is essential to promote safety and stability for production systems.
- **How do PulseRL Q-values compare to the Q-values from standard, off-policy RL algorithms?** – This question validates the initial hypothesis that standard, off-policy RL algorithms overestimate policy values, and CQL mitigates this problem. We present the analysis for this question in Appendix E.

During the online evaluation, we also performed an ablation study regarding online training for CQL. We denoted as "PulseRL Static" the version where we do not train after the first deployment, while "PulseRL Dynamic" keeps re-training every day, augmenting the dataset with newly available data.

For the first question, we present the online evaluation results for the signup and agreement metrics in Figure 3. It shows line charts representing the rates computed by Equation 4, where each day computes the success rate cumulatively over the past notifications.

PulseRL Static consistently outperforms all the baselines during the whole timeline. This result shows the strength of our method: it is capable of tolerating the distributional shift and generalizes to

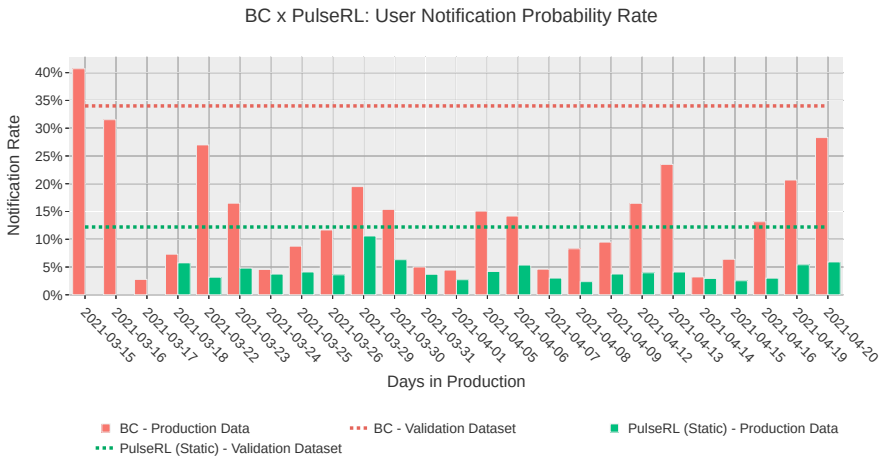

Figure 4: Notification Rates from BC and PulseRL (Static) over the production timeline.

the production distribution very well, without any online re-training. We also highlight that PulseRL found a better policy than the induced by the actions in the dataset (represented by the BC method). It suggests that PulseRL sticks together segments of suboptimal trajectories to perform a better trajectory.

PulseRL Dynamic exhibits mixed results. It presents the best final signup rate but performs similarly to baselines in the final agreement rate. We hypothesize that this occurred because the rewards from agreements are much more sparse than those from signups since the latter are much rarer events than the former. Therefore, the online training optimized the agent to improve signup instead of agreement. Indeed, all online methods performed worse for agreement than the static ones, which suggests that we should carefully tune the rewards terms weights for production re-training. Nevertheless, the best result for PulseRL Dynamic in signup rate shows interesting adaptability to maximize production rewards, while PPO and DQN failed to show that.

For the second question, we present Figure 4. It shows the daily notification rate for the BC and PulseRL static models in production. The notification rate computes the ratio between the number of notifications sent and the volume of users evaluated by the model. The dashed horizontal lines show the notification rate computed over the validation dataset prior to the deployment.

Since they are static models, the variance of daily rates is intrinsically related to the distributional shift from OOD actions. Both methods presented substantial disparity in notification rates from the validation dataset and in production. This disparity is an effect of the distributional shift. However, PulseRL presented much less variance in the notification rate, suggesting that it is more robust to changes in the data distribution. Furthermore, PulseRL consistently maintains a lower rate than BC due to the conservative behavior from CQL training. This effect is desirable once that more notifications induce more operational costs.

6 Conclusions and Future Work

In this work, we presented PulseRL, an RL-based production system to optimize communication channels in Digital Marketing Systems. We enabled Offline Reinforcement Learning by using the CQL pessimistic framework for policy evaluation and training. Furthermore, we designed PulseRL to be scalable for millions of users and to perform intensive data processing tasks, deployment, and inference at this scale.

Our experimental setting further validated the theoretical results from the CQL algorithm in this challenging scenario of a real-world production system, consistently outperforming all baselines. It also showed excellent practical advantages:

- PulseRL learns by maximizing rewards from historical logs, avoiding early deployments that could be potentially dangerous and costly in many business scenarios;

- After deployment, PulseRL presents a conservative behavior over OOD actions, which in practice reduced unnecessary notifications and the cost of the system; and
- Assuming a consistent offline metric, PulseRL can easily search and select hyperparameters by training in the historical logs, which is unfeasible for online RL algorithms requiring interactions in production.

As future work, we plan to improve offline evaluation by adopting more robust estimators to draw a better correlation with online metrics. It will enable a consistent iterative process to improve offline hyperparameter selection and experiment with new MDP formulations and data processing techniques.

References

[1] Naoki Abe, Naval Verma, Chid Apte, and Robert Schroko. Cross channel optimized marketing by reinforcement learning. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, page 767–772, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138881. doi: 10.1145/1014052.1016912. URL https://doi.org/10.1145/1014052.1016912.

[2] Naoki Abe, Prem Melville, Cezar Pendus, Chandan K. Reddy, David L. Jensen, Vince P. Thomas, James J. Bennett, Gary F. Anderson, Brent R. Cooley, Melissa Kowalczyk, Mark Domick, and Timothy Gardinier. Optimizing debt collections using constrained reinforcement learning. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, page 75–84, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450300551. doi: 10.1145/1835804.1835817. URL https://doi.org/10.1145/1835804.1835817.

[3] Erik Bernhardsson and Elias Freider. spotify/luigi, May 2014. URL https://github.com/spotify.

[4] Jacob Buckman, Carles Gelada, and Marc G. Bellemare. The importance of pessimism in fixed-dataset policy optimization. *CoRR*, abs/2009.06799, 2020. URL https://arxiv.org/abs/2009.06799.

[5] Jinglin Chen and Nan Jiang. Information-theoretic considerations in batch reinforcement learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1042–1051. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/chen19e.html.

[6] Peijin Cong, Junlong Zhou, Mingsong Chen, and Tongquan Wei. Personality-guided cloud pricing via reinforcement learning. *IEEE Transactions on Cloud Computing*, PP:1–1, 05 2020. doi: 10.1109/TCC.2020.2992461.

[7] Miroslav Dudík, John Langford, and Lihong Li. Doubly robust policy evaluation and learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 1097–1104, Madison, WI, USA, 2011. Omnipress. ISBN 9781450306195.

[8] Justin Fu, Aviral Kumar, Matthew Soh, and Sergey Levine. Diagnosing bottlenecks in deep q-learning algorithms. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2021–2030. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/fu19a.html.

[9] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2052–2062. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/fujimoto19a.html.

[10] Ruben Geer, Qingchen Wang, and Sandjai Bhulai. Data-driven consumer debt collection via machine learning and approximate dynamic programming. *SSRN Electronic Journal*, 01 2018. doi: 10.2139/ssrn.3250755.

[11] Ondřej Grunt, Jan Plucar, Markéta Štáková, Tomáš Janečko, and Ivan Zelinka. Modeling of marketing processes using markov decision process approach. pages 468–476, 09 2018. ISBN 978-3-319-68320-1. doi: 10.1007/978-3-319-68321-8_48.

[12] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel : Model-based offline reinforcement learning. *CoRR*, abs/2005.05951, 2020. URL https://arxiv.org/abs/2005.05951.

[13] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21810–21823. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/f7efa4f864ae9b88d43527f4b14f750f-Paper.pdf.

[14] Xiangyu Kong, Deqian Kong, Jingtao Yao, Linquan Bai, and Jie Xiao. Online pricing of demand response based on long short-term memory and reinforcement learning. *Applied Energy*, 271: 114945, 08 2020. doi: 10.1016/j.apenergy.2020.114945.

[15] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1179–1191. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/0d2b2061826a5df3221116a5085a6052-Paper.pdf.

[16] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *CoRR*, abs/2005.01643, 2020. URL https://arxiv.org/abs/2005.01643.

[17] Leon H. Liebman. A markov decision model for selecting optimal credit control policies. *Manage. Sci.*, 18(10):B–519–B–525, June 1972. ISSN 0025-1909. doi: 10.1287/mnsc.18.10.B519. URL https://doi.org/10.1287/mnsc.18.10.B519.

[18] Gerard Miller, Melissa Weatherwax, Timothy Gardinier, Naoki Abe, Prem Melville, Cezar Pendus, David Jensen, Chandan K. Reddy, Vince Thomas, James Bennett, Gary Anderson, and Brent Cooley. Tax collections optimization for new york state. *Interfaces*, 42(1):74–84, 2012. ISSN 00922102, 1526551X. URL http://www.jstor.org/stable/41472730.

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[20] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging ai applications. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, page 561–577, USA, 2018. USENIX Association. ISBN 9781931971478.

[21] Oded Netzer and James Lattin. A hidden markov model of customer relationship dynamics. *Marketing Science*, 27, 03 2008. doi: 10.1287/mksc.1070.0294.

[22] Ajay Sampat. Building lyft's marketing automation platform, Jun 2019. URL https://eng.lyft.com/lyft-marketing-automation-b43b7b7537cc.

[23] Marlesson R. O. Santana, Luckeciano C. Melo, Fernando H. F. Camargo, Bruno Brandão, Anderson Soares, Renan M. Oliveira, and Sandor Caetano. Mars-gym: A gym framework to model, train, and evaluate recommender systems for marketplaces. In Giuseppe Di Fatta, Victor S. Sheng, Alfredo Cuzzocrea, Carlo Zaniolo, and Xindong Wu, editors, *20th International Conference on Data Mining Workshops, ICDM Workshops 2020, Sorrento, Italy, November 17-20, 2020*, pages 189–197. IEEE, 2020. doi: 10.1109/ICDMW51313.2020.00035. URL https://doi.org/10.1109/ICDMW51313.2020.00035.

[24] Marlesson R. O. Santana, Luckeciano C. Melo, Fernando H. F. Camargo, Bruno Brandão, Anderson Soares, Renan M. Oliveira, and Sandor Caetano. Contextual meta-bandit for recommender systems selection. In Rodrygo L. T. Santos, Leandro Balby Marinho, Elizabeth M. Daly, Li Chen, Kim Falk, Noam Koenigstein, and Edleno Silva de Moura, editors, *RecSys 2020: Fourteenth ACM Conference on Recommender Systems, Virtual Event, Brazil, September 22-26, 2020*, pages 444–449. ACM, 2020. doi: 10.1145/3383313.3412209. URL https://doi.org/10.1145/3383313.3412209.

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[26] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press, 2nd edition, 2018. ISBN 9780262039246. URL https://books.google.com.br/books?id=sWV0DwAAQBAJ.

[27] Adith Swaminathan, Akshay Krishnamurthy, Alekh Agarwal, Miro Dudik, John Langford, Damien Jose, and Imed Zitouni. Off-policy evaluation for slate recommendation. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3632–3642. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6954-off-policy-evaluation-for-slate-recommendation.pdf.

[28] Georgios Theocharous, Philip S. Thomas, and Mohammad Ghavamzadeh. Personalized ad recommendation systems for life-time value optimization with guarantees. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, page 1806–1812. AAAI Press, 2015. ISBN 9781577357384.

[29] Georgios Theocharous, Yash Chandak, Philip S. Thomas, and Frits de Nijs. Reinforcement learning for strategic recommendations. *CoRR*, abs/2009.07346, 2020. URL https://arxiv.org/abs/2009.07346.

[30] Philip S. Thomas, Georgios Theocharous, Mohammad Ghavamzadeh, Ishan Durugkar, and Emma Brunskill. Predictive off-policy policy evaluation for nonstationary decision problems, with applications to digital marketing. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 4740–4745. AAAI Press, 2017.

[31] Philip S. Thomas, Georgios Theocharous, Mohammad Ghavamzadeh, Ishan Durugkar, and Emma Brunskill. Predictive off-policy policy evaluation for nonstationary decision problems, with applications to digital marketing. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 4740–4745. AAAI Press, 2017. URL http://aaai.org/ocs/index.php/IAAI/IAAI17/paper/view/14550.

[32] Qing Wang, Jiechao Xiong, Lei Han, peng sun, Han Liu, and Tong Zhang. Exponentially weighted imitation learning for batched historical data. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/4aec1b3435c52abbdf8334ea0e7141e0-Paper.pdf.

[33] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. MOPO: model-based offline policy optimization. *CoRR*, abs/2005.13239, 2020. URL https://arxiv.org/abs/2005.13239.

[34] Tianhe Yu, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn. COMBO: conservative offline model-based policy optimization. *CoRR*, abs/2102.08363, 2021. URL https://arxiv.org/abs/2102.08363.

A AC Environment

AC is an online debt negotiation platform aiming to help 63 million debtors (68% of the families in Brazil) to pay off their debts and reach their financial well-being. The platform accommodates debt contracts from around 30 partners from different segments such as banks, retail stores, telecommunication services, education institutions, credit card issuers, and essential public services. AC resembles a marketplace where users can check debt contracts in real-time from available partners. Furthermore, the platform also contact users via SMS and email, notifying them about their debts. It needs to select consumers and their contact information that are highly likely to sign up and close an agreement to pay their debts. Since there are costs related to each message or email linked to the notification, there is an interest at AC to define the best strategies to select those to be notified, achieving overall profit, improving customer satisfaction, and preventing the partner's and AC's image to be damaged to the public.

B MDP Description

In this section, we describe the state space and reward functions modeled for the debt notification problem.

B.1 State Space

We compose the state space with three main sources:

- **User Profile Data**: Comprises debtor available data that changes little over time. Table 1 describes user profile features.

Table 1: State Space – User Profile

Feature	**Description**
Age	Integer representing the user age
Gender	Categorical variable for gender information
Profession	Categorical variable for user profession
Education	Categorical variable for user education level
Presumed Income	Integer representing the user presumed income
IRS Status	Binary variable that describes whether user's IRS status is regular or not
Social Welfare Status	Binary variable that describes whether the user is part of a Social Welfare program or not
Living State	Categorical variable containing the State where the user lives.
User Signup	Binary variable that describes whether user is already registered in the debtor platform or not

- **Historical Data**: Information about the contract's situation. We call it historical because it involves tracking past events such as payments and agreements. We compose the debt information with data regarding the total amount of debt, to whom, and how old it is. Since we cannot process an infinite amount of different debts, we will have two main subdivisions of this data. The first will regard the total amount of debt (Total Debt Data) as a whole with all different debts put together. The second will regard the N largest debts with more specific information (Specific Contract Data). Table 2 presents the features of historical data.

Table 2: State Space – Historical Data

Group	Feature
Total Debt Data	Total amount of original debt
	Total amount of current debt
	Approximate days since last payment
	Last payment day specs (day, month, week)
	Discounted sum of all payments
	Total paid debt
	Days since last notification
	Last day activate specs (day, month, week)
Specific Contract Data	Days since contract started
	Product Type
	Original amount of contract
	Current amount of contract
	Amount of last payment
	Discounted sum of payments
	Amount already paid
	Approximate days since last payment
	Last payment day specs (day, week, month)
	Days since last located
	Last located day specs (day, week, month)
	Days since last visualized
	Last visualized day specs (day, week, month)
	Days since last agreement
	Last agreement day specs (day, week, month)

- **Situational Data**: Information regarding the current moment/day. We compose this contextual data with categorical variables representing the Weekday, Month day, and Week of the Month.

B.2 Reward Function

We consider the reward function in Equation 5 for our MDP formulation:

$$R_t(a) = \varphi_r \cdot \mathbb{1}_r + \varphi_d \cdot \mathbb{1}_d - \epsilon - \gamma \cdot \mathbb{1}_a, \tag{5}$$

where:

- φ_r is the expected lifetime value associated to that customer when he registers in the debtor platform;
- $\mathbb{1}_r$ is a binary function which returns 1.0 if the customer performs a registration in the debtor platform and 0.0 otherwise;
- φ_d is the expected lifetime value associated to the customer when he make an agreement to pay the debit in the debtor platform;
- $\mathbb{1}_d$ is a binary function which returns 1.0 if the customer performs an agreement to pay a debit in the debtor platform and 0.0 otherwise;
- ϵ is the "survival" penalization. This is a negative reward given at each state to ensure that the agent should find solutions with short trajectories;

- γ is the cost associated to send a notification; and
- $\mathbb{1}_a$ is a binary function which returns 1.0 if the agent sends a notification and 0.0 otherwise.

We compute the expected lifetime values φ_r and φ_d by using a prediction model. We trained it using a different dataset of previous customers who performed registrations and agreements in the debt platform. We also set $\epsilon = 0.001$ and $\gamma = 0.033$.

C Dataset Description

The dataset portrays the interactions between contracts, users, and the platform. There are three primary sources of data:

- The contracts received from the partners containing the debt value and debt delay;
- The website events logged from user interactions; and
- The restricted users' demographic information.

In addition, we added the users' expected lifetime values and their signup/agreement scores as meaningful features.

D Hyperparameters

The list of all hyperparameters used in this work can be found in this link: `https://figshare.com/s/9543ca7f00fa7d0914dd`.

E How do PulseRL Q-values compare to the Q-values from standard, off-policy RL algorithms?

To analyze this question, we trained a DQN agent using the historical logs to build up the experience replay buffer. We chose DQN because it is the off-policy RL baseline in our experimental setting. Since this question compares offline training from standard off-policy algorithms and PulseRL, we do not pre-train it with BC. Figure 5 presents the distribution of max Q-values over the training steps.

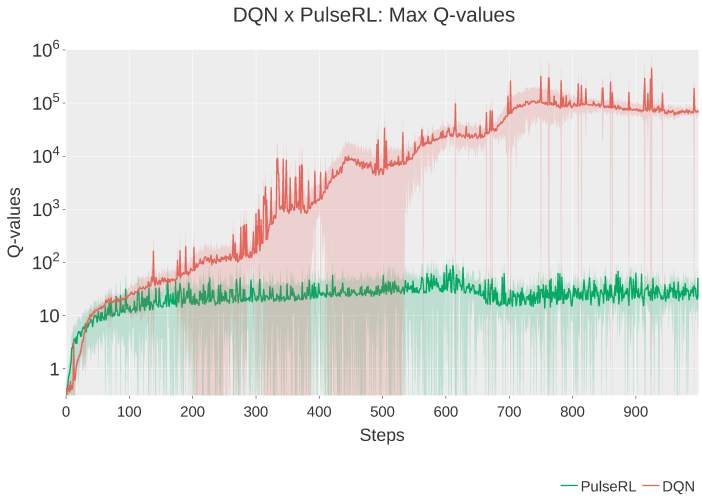

Figure 5: Distribution of max Q-values over the training phase. Note that the y-axis is on a logarithmic scale. We computed 95% bootstrap confidence intervals over 3 seeds.

DQN training overestimates Q-values over 1000x more than PulseRL at the end. On the other side, PulseRL maintains the max Q-value approximately constant throughout the whole training. These

results show that the CQL training used in PulseRL mitigates the problem, as induced by Theorem 1. Therefore, this result also suggests that the CQL framework extends to a real-world, high-dimensional problem.