CRAB: Churn Reduction Agent Based via Reinforcement Learning

ALESSANDRO SBANDI, University of Rome La Sapienza, Rome, Italy
NINA KAPLOUKHAYA, TIM S.p.A., Roma, Italy
FEDERICO SICILIANO, University of Rome La Sapienza, Rome, Italy
ARIS ANAGNOSTOPOULOS, University of Rome La Sapienza, Rome, Italy
FABRIZIO SILVESTRI, University of Rome La Sapienza, Rome, Italy

The application of offline reinforcement learning (RL) in recommendation systems has proven valuable in improving user engagement. However, a significant challenge in real-world scenarios is the delayed nature of user feedback, which can degrade algorithm performance and complicate the task of reducing user churn, i.e. losing clients to competitors. In this paper, we present an end-to-end framework called Churn Reduction Agent-Based (CRAB), which is specifically designed to efficiently incorporate delayed feedback into the recommendation process. CRAB uses a conservative Q-learning model to recommend offers to customers, incorporating delayed churn feedback directly into the reward function of the RL model. This allows the RL agent to optimize recommendations not only based on immediate acceptance probabilities, but also considering long-term user retention. Our approach also uses additional auxiliary model predictions to improve the representation of customer states in the RL environment, which significantly improves the framework's performance. We validate our method using large-scale proprietary data from a real-world telecommunication company, addressing the complexities of scalability and practical implementation. Extensive offline experiments show that CRAB outperforms existing methods, including both internal company policies and state-of-the-art approaches. In addition, our online testing results confirm the framework's effectiveness in delivering superior recommendations and effectively reducing user churn by 4.95%.

CCS Concepts: • **Information systems** → **Recommender systems**; **Personalization**; • **Theory of computation** → **Sequential decision making**; • **Applied computing** → *Telecommunications*.

Additional Key Words and Phrases: Recommendation systems, Reinforcement learning, Personalization, AI for Industrial Applications, Telecommunications

1 Introduction

Recommendation systems have become an integral part of many major companies, including Google [11, 14], Facebook [48], Amazon [59], and Netflix [24]. They are used in a wide range of applications, such as e-commerce [54], entertainment [9, 41], and news [36], where they help tailor content to user preferences.

Unlike e-commerce or social media platforms, where the primary goal may be to increase engagement or sales, telecommunications companies prioritize reducing customer churn rather than actively pursuing upselling opportunities [30]. In this context, churn refers to the phenomenon of customers switching from one service provider to another [32], resulting in substantial financial losses. This behavior can be caused by dissatisfaction, increased costs, inadequate quality, missing features, and privacy issues [57].

Authors' Contact Information: Alessandro Sbandi, University of Rome La Sapienza, Rome, Lazio, Italy; e-mail: alessandro.sbandi@gmail.com; Nina Kaploukhaya, TIM S.p.A., Roma, Lazio, Italy; e-mail: ninakaplouh@gmail.com; Federico Siciliano, University of Rome La Sapienza, Rome, Italy; e-mail: federico.siciliano@uniroma1.it; Aris Anagnostopoulos, University of Rome La Sapienza, Rome, Lazio, Italy; e-mail: aris@diag.uniroma1.it; Fabrizio Silvestri, University of Rome La Sapienza, Rome, Lazio, Italy; e-mail: fsilvestri@diag.uniroma1.it.

ACM 2770-6699/2026/5-ART
https://doi.org/10.1145/3814953

To alleviate this problem, this paper addresses a specific and challenging industry use case: a telecommunications provider where recommendations are delivered through live call center interactions. When a customer calls to cancel service, report a fault, or ask a question, the system must rank a catalog of hundreds of commercial offers and present them to the human operator, who then selects which to propose.

This setting imposes three challenges: extreme feedback sparsity, a dynamic catalog that regularly introduces new offers, and a delayed outcome signal. In online systems, due to the data collection ingestion time, a substantial amount of user feedback may not have occurred, leading to their absence in the training data. The importance of addressing this delayed feedback is particularly pronounced in this sector, complicating the process of developing effective customer retention strategies.

Applying Reinforcement Learning to Recommender Systems (RLRS) has attracted considerable interest due to its ability to adapt to users' behaviour over time. Unlike traditional recommendation methods that may focus only on immediate interactions, RLRS can optimize the long-term user engagement by learning from sequential user-system interactions [1, 52]. This makes RLRS particularly suited to environments where user preferences are dynamic and complex, as it can provide personalized and timely recommendations.

Offline reinforcement learning has become popular in recent years [15, 42], as it allows models to be trained using historical data without the need for live interaction, making it practical for industries where real-time user feedback is either unavailable or impractical. Despite its potential, offline RL presents its own set of challenges, such as overfitting and distributional shifts due to the static nature of the training data [37]. In addition, a significant issue in RLRS is the handling of delayed feedback [67], where there is a delay between the action taken by the system and the resulting user feedback. This delay can introduce bias and make it difficult to estimate the true value of an action.

In light of this, our primary goal is to reduce customer churn, ensuring high, consistent performance in recommendations simultaneously. In this paper, we present a novel approach to recommendation systems in the telecommunications industry, with a focus on preventing churn. We propose an end-to-end framework called **C**hurn **R**eduction **A**gent-**B**ased (CRAB), which uses offline RL to optimize the recommendation of offers to customers. By incorporating a carefully designed reward function that penalizes churn events more severely than offer refusals, our model aims to generate personalized recommendations that not only appeal to the customer, but also mitigate the risk of churn. Our method also includes the integration of additional customer state representations derived from various company models to improve the accuracy and relevance of the recommendations.

To validate the effectiveness of our approach, we conduct extensive offline and online experiments on real data from millions of users in a telecommunications company. Our results show that CRAB outperforms existing company policy and state-of-the-art methods, and offers a promising solution for reducing customer churn (-4.95%). Moreover, we provide diverse recommendations, achieving over 50% Coverage@5.

The main contributions of this work are as follows:

- We introduce a new perspective on recommendation systems by prioritizing customer churn reduction, providing a novel application of offline RL in a critical industry context.
- We propose and validate a practical methodology for enriching RL state representations by incorporating auxiliary predictions from pre-existing, independently trained business models. This is a powerful and scalable alternative to complex joint-training, with significant implications for industrial RL applications.
- Our method is rigorously evaluated through offline experiments and an online A/B test within a major telecommunications company. This achieved a 4.95% reduction in customer churn and outperformed competitive baselines. To the best of our knowledge, this is one of the first studies to demonstrate measurable success in customer upselling and reduction in long-term churn events, providing valuable insights for real-world applications.

The rest of the paper is organized as follows: Section 2 reviews related work, Section 3 outlines the theoretical background, Section 4 details the methods and the model, and Section 5 presents the dataset, baselines, and evaluation metrics, and the choice of hyperparameters. Finally, Section 6 concludes the paper and discusses possible future work. Section A details the hyperparameter tuning search space for all baselines, and the best values that we found.

2 Related work

2.1 Offline RLRS

RL has become a popular approach for RLRS because it can model the sequential nature of user interactions [13, 63]. Unlike simpler models that only optimize for immediate feedback, RLRS aim to learn a recommendation policy π that maximizes long-term user engagement and satisfaction [61, 68]. RLRS can be divided into three different categories. The first one is *on-policy* RL methods that require the learning agent to interact directly with live users to collect data for policy updates. This is often not practical for large systems because it is expensive and can lead to a poor user experience [52].

The second one is *off-policy* RL methods [27], which buffer all their data in a replay buffer and then replay it from there. However, classic off-policy RL methods require additional online collection, as after we update the policy, we usually use it to obtain additional data [71]. The third is *offline RL* [43], where the agent learns from a fixed dataset of pre-collected experiences without collecting additional data during training [37, 60].

However, offline RL faces the fundamental challenge of distributional shift [42]. Since the dataset is generated by a fixed behaviour policy, it does not cover all possible actions [65]. Consequently, standard off-policy algorithms can wrongly assign high values to out-of-distribution (OOD) actions [21, 46], creating policies that appear successful during training, but fail upon deployment [37]. Several approaches have been proposed to mitigate this issue. One approach involves constraining the learned policy to stay close to the policy that generated the data, thereby limiting divergence [50, 51]. While effective, these constraints may be too limiting and may prevent the discovery of more effective strategies. An alternative approach is to change the learning objective to be cautious about OOD actions. This is the main idea behind Conservative Q-learning (CQL) [37, 58]. CQL uniformly pushes down all Q-values to guarantee a lower bound on performance.

Building on CQL, several variants have been developed to address specific limitations. Calibrated Q-Learning (Cal-QL) enables efficient online fine-tuning after offline pre-training [47]. Cal-QL learns conservative lower bounds on the learned policy's true value and reasonable upper bounds on reference policies such as the behavior policy. This calibration prevents the "unlearning" phenomenon where offline-trained policies deteriorate during online fine-tuning, enabling rapid improvement with minimal additional interaction.

Exclusively Penalized Q-Learning (EPQ) tackles the problem of unnecessary conservatism in well-covered regions of the state-action space [64]. EPQ introduces an adaptive threshold mechanism that selectively applies penalties only to state-action pairs that are genuinely prone to overestimation, thereby avoiding the excessive underestimation that can occur in dense data regions. This selective approach maintains the safety benefits of conservative estimation while reducing bias in areas where the behavioral data provides adequate coverage.

Counterfactual Conservative Q-Learning (CFCQL) adapts CQL principles to multi-agent settings by computing conservative regularization for each agent separately in a counterfactual manner [56]. This approach maintains the theoretical guarantees of CQL while ensuring that the induced regularization bounds remain independent of the number of agents, making it particularly suitable for large-scale multi-agent systems. State-corrected algorithms, such as SCAS [46], go beyond penalizing actions, offering more holistic robustness.

Value Penalized Q-Learning (VPQ) [23] is an offline RL algorithm that addresses value overestimation differently. It captures uncertainty via an ensemble of Q-functions and uses it to penalize OOD actions. A key advantage

of VPQ is that it obviates the need for explicit behavior policy estimation, a challenging task in recommender systems, thereby improving estimates of long-term rewards [66].

In light of this, we selected Conservative Q-Learning (CQL) as the foundation for our proposed method. Our primary objective is to develop a robust and reliable policy learned from a fixed, static dataset. CQL's core mechanism of penalizing OOD actions provides a safe and effective foundation against value overestimation, making it a suitable and strong choice for our high-stakes industrial application, where unreliable recommendations can directly lead to customer churn.

2.2 Delayed Feedbacks

A fundamental challenge in applying RL to real-world recommender systems is the significant delay between issuing a recommendation and observing its true outcome [34]. While immediate signals, such as clicks or ratings, are readily available, critical long-term outcomes, such as user retention, subscription renewal, or churn, may only become evident weeks or months later. This temporal gap complicates credit assignment, as the RL agent must learn to associate a delayed reward to the correct sequence of prior actions that caused it. This challenge is further exacerbated in the offline RL setting, where the agent cannot perform online exploration and must infer delayed outcomes only from historical data where such connections are not explicit [3, 26]. In recommender systems, the real reward depends on the user's interaction with the recommended item [29], an event that may require weeks to be determined.

Several strategies have been proposed to mitigate the effects of delayed feedbacks. Mann et al. [45] exploit the intermediate reward signals to determine future delayed feedback using a factored learning approach. Hung et al. [31] solve delayed reward tasks by reassigning the rewards that appear after a causative event. To do this, they employ a memory-based agent and utilize the memory retrieval system to attribute credit to events from the distant past appropriately. These events are valuable for predicting the value function at the current timestep. Arjona-Medina et al. [3] present the RUDDER algorithm that adopts a backward-view approach for a task, creating a return-equivalent Markov Decision Process where delayed rewards are more evenly redistributed across time. More recently, Han et al. [26] focus on establishing a sample-efficient off-policy RL algorithm that is adaptive to delayed environment signals.

Memory-augmented models also play a decisive role in tackling this challenge. Architectures incorporating explicit memory retrieval modules have shown strong performance in linking key past events to delayed rewards [5], facilitating more effective value estimation. At the architectural level, deep RL methods (e.g., DDPG, SAC, TD3) have been extended to support long-horizon or delayed feedback scenarios. These adaptations often involve state augmentation [38], attention mechanisms, and sophisticated sequence modeling [10] or counterfactual reasoning [4], further enhancing robustness and interpretability [22].

While these approaches offer sophisticated mechanisms, they often assume complex temporal dependencies or require substantial architectural modifications. In the context of telco churn, the problem is both simpler and more definitive: churn is a binary, terminal event occurring after a certain delay. To address this directly, we propose a more practical methodology. By structuring each episode with a fixed look-ahead window (see Section 3.2.4) and designing a reward function that heavily penalizes this terminal churn event (see Section 3.2.5), we align the agent's learning objective with the long-term business goal of churn reduction in a simple yet effective way, avoiding the need for complex model modifications.

2.3 Auxiliary Tasks

In deep RL, an agent's success depends on the quality of its internal representations, yet in environments characterized by high-dimensional observations and sparse rewards, learning these representations from the primary objective alone often proves inefficient. Auxiliary tasks, secondary objectives optimized alongside the

main RL objective, have been introduced to provide richer, more frequent learning signals, accelerating both representation learning and policy optimization [20, 33, 44]. These extra tasks provide additional signals that help the agent develop improved representations and policies, particularly in situations where data is sparse or rewards are delayed.

Auxiliary tasks fall into several paradigms. One category is predictive learning, in which the agent learns to predict future aspects of its environment, such as next state, immediate reward, or latent dynamics. This paradigm is central to model-based RL methods, such as Dreamer [8, 25]. By learning to predict what will happen next, it integrates reconstruction and prediction losses within a latent world model, allowing the agent to predict trajectories before acting in the real environment. This driven approach dramatically improves sample efficiency by guiding exploration toward the most informative transitions.

Another key paradigm is self-supervised learning, where training signals are generated from the input data itself, without requiring external labels. Contrastive learning approaches like CURL [39] train the agent to produce similar representations for different augmented views of the same state, promoting invariance to task-irrelevant features. These techniques have been shown to improve sample efficiency in vision-based RL [18]. Other self-supervised tasks include observation reconstruction, where an autoencoder architecture is used to reconstruct observations, and inverse dynamics modeling, where the agent learns to predict the action taken between two consecutive states [49].

Auxiliary tasks can also be framed as control objectives. Diversity-maximization methods like DIAYN [17] train the agent to learn a set of skills by maximizing the mutual information between a latent skill variable and the resulting state visitation distribution. This promotes broad exploration and the acquisition of disentangled features that can benefit the main RL objective. This encourages the exploration of the state space.

In recommender systems, auxiliary tasks are typically designed to address the unique challenges of user modeling. One application is to stabilize the learning process and handle stochasticity. For instance, the Stochastic Reward Stabilization (SRS) framework [7] replaces the unreliable environment's stochastic user feedback (e.g. clicks, or likes) with a learned reward expectation, improving training stability and convergence. Similarly, augmenting a recommendation agent with a click-prediction head forces the agent to encode short-term user preferences, accelerating learning when real rewards are sparse.

Another prevalent strategy is to model immediate user feedback to create denser training signals. Chen et al. [12] augment a model-free RL agent with a task that predicts immediate user reactions (e.g., click or no-click), encouraging representations aligned with the user's short-term reaction. In the context of news recommendation, models such as DRN [69] treat different types of user feedback (e.g., clicks, shares, read time) as separate tasks within a multi-task learning framework. This encourages the agent to learn a more holistic representation of user engagement. These innovations in predictive, self-supervised, and control-based auxiliary learning have advanced deep RL, producing agents that learn faster, generalize better, and tackle sparse-reward challenges in both synthetic and real-world applications.

Despite their success, these methods typically involve joint-training of the primary RL agent with auxiliary objectives, which can be computationally expensive and impractical in large-scale (i.e., millions or billions of users) enterprise environments where specialized models are already developed and maintained. A key contribution of our work is moving away from this paradigm. We demonstrate that by strategically decoupling these tasks, i.e. using the outputs of pre-existing, independently trained models as direct inputs to the agent's state representation (see Section 3.2.1), we can significantly enrich the state and improve performance without the overhead and complexity of multi-task learning. This makes our approach highly practical and scalable for real-world deployment.

3 Methodology

3.1 Markov Decision Process

The recommendation task can be effectively modeled as a Markov Decision Process. In this setting, a recommender agent interacts with the environment over a series of time steps. At each time step, the agent selects an action a from a set of possible actions A, based on the current user state s, which belongs to the set of possible states S. After performing this action, the agent receives a reward $R(s, a)$, and the user state transitions to a new state s' according to the transition probability distribution $p(s'|s, a)$. The goal of the recommender agent is to learn an optimal policy $\pi_\theta : S \times A \rightarrow [0, 1]$ that maximizes the expected cumulative reward over time. The Markov Decision Process is defined by the tuple $M = (S, A, P, R, \rho)$, where:

- **States (S)**: a state $s \in S$ encapsulates the user's features and interaction history.
- **Actions (A)**: an action $a \in A$ corresponds to recommending a specific item from the dataset to the user.
- **Transition probability ($p(s'|s, a)$)**: it describes the probability of transition from state s to s' after action a.
- **Reward function ($R(s, a)$)**: this function quantifies the user's response to action a in state s.
- **Discount Rate (ρ)**: the discount rate $\rho \in [0, 1]$ measures how the value of future rewards is perceived in the present; a higher value places more emphasis on long-term rewards, encouraging the agent to consider the long-term consequences of his actions.
- **Policy π_θ**: the policy defines the agent's strategy for selecting actions given a particular state, mapping each state s to a probability distribution over actions A. The optimal policy aims to maximize the expected sum of discounted rewards, guiding the agent to make recommendations that not only meet immediate user preferences, but also promote long-term engagement.

3.2 Proposed Framework

Our goal is to create an RLRS that provides personalized recommendations to customers, maximizing the acceptance of recommended items while minimizing customer churn. Our proposed framework, named CRAB, is illustrated in Fig. 1. It consists of two main components: the environment and the agent. Raw user interaction data is pre-processed to create an appropriate RL environment. Within this environment, the RL model generates personalized recommendations for each customer, aiming to reduce churn by providing optimal action suggestions. The implementation of CRAB starts with the creation of the RL environment, which is described in detail in the following section.

3.2.1 States. States of our RL environment encompass the daily characteristics of each customer. We collect approximately 350 features per customer, encompassing previous interactions, demographic attributes, billing history, call records, and navigation data.

More specifically, each state includes fundamental demographic attributes such as age, gender, region of birth, and current region of residence. It also captures the client's relationship history with the company, including client tenure, i.e the total duration the user has been a customer, and line age, i.e. the operational age of their current service line. The dataset details user service subscriptions and financial interactions, including the billing amount, which reflects the total monthly cost of all services, i.e. the composite amount of their core service plan and charges for any supplementary add-on packages. These add-ons provide insight into customer preferences for value-added services, encompassing offerings such as premium television packages, integrated smart home or domotic solutions, and traditional voice services. In addition to their commercial and demographic profiles, the dataset captures the client's direct interactions with the customer care. These records serve as a valuable proxy for customer satisfaction, allowing for the systematic analysis of prevalent issues, including technical support needs such as connectivity problems and slow speeds, as well as administrative queries concerning billing issues,

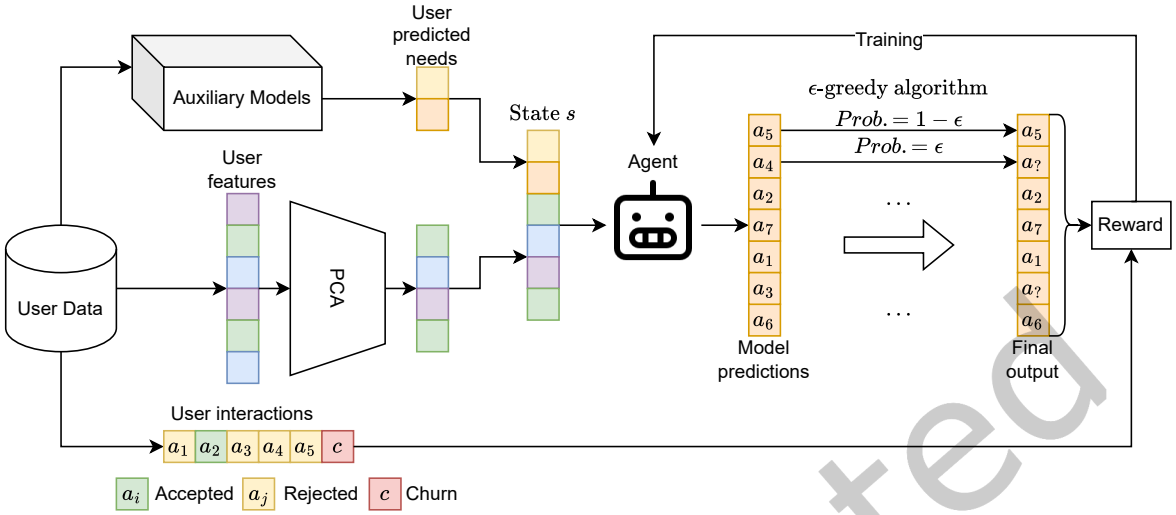

Fig. 1. The schematic structure of the proposed CRAB framework. States are composed of the concatenation of user features and company model outputs. The agent is trained using previous user interactions, taking into account its responses and churn events. ϵ-greedy strategy is implemented during the training procedure to deal with the exploration-exploitation problem.

and commercial inquiries about upgrading services or activating new offers. Another set of features quantifies the client's network usage and online behavior, such as data traffic, specifically the total volume of data downloaded and uploaded by the client over specified intervals.

To reduce the computational cost by approximately three times, we use Principal Component Analysis (PCA) to project the original feature space into a 128-dimensional space, retaining over 80% of the explained variance.

3.2.2 Auxiliary Models. Inspired by the works presented in Section 2.3, we incorporate auxiliary tasks into the customer state representations to enhance agent recommendations without the need for retraining the auxiliary models, training them jointly, or training the primary model to perform multiple tasks. Specifically, we use eight pre-existing auxiliary models developed for related business tasks, including: a gradient-boosted tree (GBT) model that predicts the 60-day churn probability, a regression model that estimates the 12-month Customer Lifetime Value, and multiple GBT-based propensity models for offer acceptance across categories (e.g, entertainment, sport, fiber-to-the-home, assurance, etc.). These models are automatically retrained in the event of a performance drop, and their outputs are generated daily. Their scalar outputs are concatenated with the PCA-reduced base user feature vector.

This design choice offers two key advantages: computational efficiency and operational viability. From a computational perspective, we bypass the overhead of joint-training pipelines, which often demand massive, unified datasets and extensive hyperparameter tuning to balance the loss functions. Secondly, by treating the auxiliary models purely as feature extractors, we avoid potential optimization challenges that could affect the performance of the primary task. Furthermore, our framework aligns with the operational realities of large companies, where a set of specialized machine learning models is developed, deployed, and managed by different teams to solve specific business problems. CRAB is designed to integrate their valuable, pre-existing outcomes without requiring costly and complex changes to current systems.

3.2.3 Actions. In our RL setup, the action space defines the set of all possible recommendations the agent can make. These correspond to commercial offers and service upgrades designed to improve the customer experience and increase lifetime value. The catalog is highly diverse, spanning multiple product categories. It includes foundational telecommunication services (e.g., specific voice and Wi-Fi plans), a wide array of entertainment subscriptions for TV and streaming platforms (such as DAZN, Netflix, Amazon Prime, Disney+, and TIMVISION), various promotional discounts, smart home devices (e.g., Google Nest), extra services (e.g., assurance), and essential network equipment like routers. For the scope of this study, this cumulative catalog results in a discrete action space comprising over three hundred unique actions, presenting a substantial and complex decision-making challenge for the recommendation agent.

At inference time, recommendations are deployed in real-time customer service interactions. Whenever a customer contacts the call center, for example, to cancel their service, report technical malfunctions, or request information, the RL framework estimates Q-values for all possible actions and ranks them. The top five actions are presented to the human operator, who then presents one or more to the customer. Crucially, the feedback used for training corresponds to the specific offer the human operator suggested, and the reward is based on the customer's response to that chosen offer. This process ensures our agent learns to generate rankings that are effective in a real-world, operator-assisted environment.

As only the top five items are shown in practice, our offline analysis presented in Section 5.1 places a significant emphasis on evaluation metrics calculated with a cutoff of $K = 5$. This ensures alignment between offline evaluation and its intended operational environment.

3.2.4 Episodes. In our RL formulation, an episode represents the complete interaction history of a single customer: sequence of states, actions (or recommendations), and rewards, starting from their first observed interaction and terminating either when they churn or when the study's observation period ends.

A significant challenge in modeling customer behavior, particularly churn, is the inherent latency between a customer's decision to leave and the actual service cancellation. The negative experiences or unsatisfactory offers that trigger churn may occur weeks or even months before the event is formally recorded in the system. A naive approach that only considers the churn status at the immediate end of an episode would fail to capture these delayed effects, potentially misattributing the churn event to more recent, unrelated interactions and failing to penalize the true causal actions. To address this temporal misalignment, we establish a two-month look-ahead window beyond the episode's terminal state. If churn occurs within this window, the episode's terminal state is retrospectively labeled as a churn state, and the corresponding reward is applied.

Crucially, while this window is used for outcome labeling, any actions suggested to the customer or interactions that occur within this period are excluded from the episode's state-action-reward sequence. This ensures that rewards are attributed to the sequence of recommendations made during the episode and not to events that happened after the episode had concluded. This strategy allows the agent to learn the long-term consequences of its actions more effectively, leading to a more robust and realistic recommendation policy.

3.2.5 Reward. The reward function is a crucial component in our framework, as it guides the agent to learn actions that not only maximize immediate acceptance rates, but also minimize long-term customer churn. The reward for each action depends on whether the action was accepted or refused by the customer, as well as the occurrence of a churn event.

To account for churn, we introduce a delayed reward component. This approach ensures that the agent prioritizes actions that both engage the customer and reduce the likelihood of churn. Specifically, the agent receives a reward or penalty in a terminal state not only based on the acceptance of the last offer, but also based on the churn outcome. The reward function $R(s, a)$, based on the customer's response r to the suggested action

and the churn event c, is defined as follows:

$$R(s,a) = \begin{cases} \alpha, & \text{if } r(s,a) = 1, c(s,a) = 0 \quad \textit{Accept} \\ \beta, & \text{if } r(s,a) = 0, c(s,a) = 0 \quad \textit{Refuse} \\ \gamma, & \text{if } r(s,a) = 1, c(s,a) = 1 \quad \textit{Accept but churn} \\ \delta, & \text{otherwise} \quad \textit{Refuse but churn} \end{cases} \tag{1}$$

where $r(s,a)$ is the customer's response (1 for accepted, 0 for refusal), and $c(s,a)$ indicates the occurrence of a churn event (1 for churn, 0 for no churn). α, β, γ and δ are constants chosen according to the following criteria:

- $\alpha, \beta > 0, \alpha > \beta$ representing positive rewards for non-churning users.
- $\gamma, \delta < 0$ to reflect a penalty for churning.
- $\alpha > \beta, \gamma > \delta$ to prioritize action acceptance over rejection.
- $\alpha + \delta < 0$ to discourage the event of a user accepting a recommendation and churning after the first refusal.
- $\alpha + \gamma > 0$ to accept the event of a user accepting two recommendations before churning

The churn-related rewards apply only in the terminal state, while for the rest of the episode, only cases with $c = 0$ are considered. The reward is constructed this way to emphasize the importance of preventing the churn of customers. Even if the reward considers whether the action is accepted or refused, it gives more weight to whether this action leads to the churn of the customer or not. In an ideal scenario, with this reward, the agent, by interaction with the environment, should learn the long-term consequences of the actions to understand which action leads to churn.

3.2.6 Recommender System. To optimize the policy, we employ a discrete *Conservative Q-Learning* (CQL) model [37], a state-of-the-art approach in offline reinforcement learning. CQL is designed to learn conservative, lower-bound estimates of the value function by incorporating regularization into the Q-values during training, thereby mitigating the risks associated with overestimation and distributional shift. Formally, the model is trained to minimize the loss:

$$L(\theta) = \phi\, \mathbb{E}_{s_t \sim D}[\log \sum_a expQ_\theta(s_t, a) - \mathbb{E}_{a \sim D}[Q_\theta(s,a)]] + L_{DoubleDQN(\theta)} \tag{2}$$

In the CQL algorithm, multiple Q-functions can be updated, instead of one, with independently sampled experience replay memory, and then the action selected by the ensemble is taken. The number of these Q-functions is referred to as the number of critics. Previous studies, such as [62], have shown that this ensemble approach surpasses non-ensemble methods in both performance and stability. Utilizing an ensemble of Q-functions effectively reduces decision variance, leading to more efficient training in specific tasks. However, this approach also introduces a potential drawback: increased computational time for each reinforcement learning step.

To address the exploration-exploitation dilemma, we adopted the ϵ-greedy approach during the training phase. Specifically, with some probability ϵ, the agent selects a random action, while with probability $(1 - \epsilon)$, it chooses the best action according to the Q-function. This method allows the agent to explore, during training, different actions with probability ϵ and exploit the action that maximizes the Q-value with probability $(1 - \epsilon)$. This balance between exploration and exploitation is crucial for effectively learning an optimal policy. During inference and deployment, ϵ is *set* to 0, and the agent recommends the actions with the highest learned Q-values.

3.2.7 Cold start campaigns. When launching a new campaign, the company may introduce a new offer set that has not been previously tested or recommended to users. This situation leads to the cold start problem, because our reinforcement learning model has no historical performance data from which to derive Q-values for these new items. To address this, we leverage the similarities between the new offers and the existing ones, along with the rankings provided by the trained model, to generate a ranking for the campaign offers. This ranking

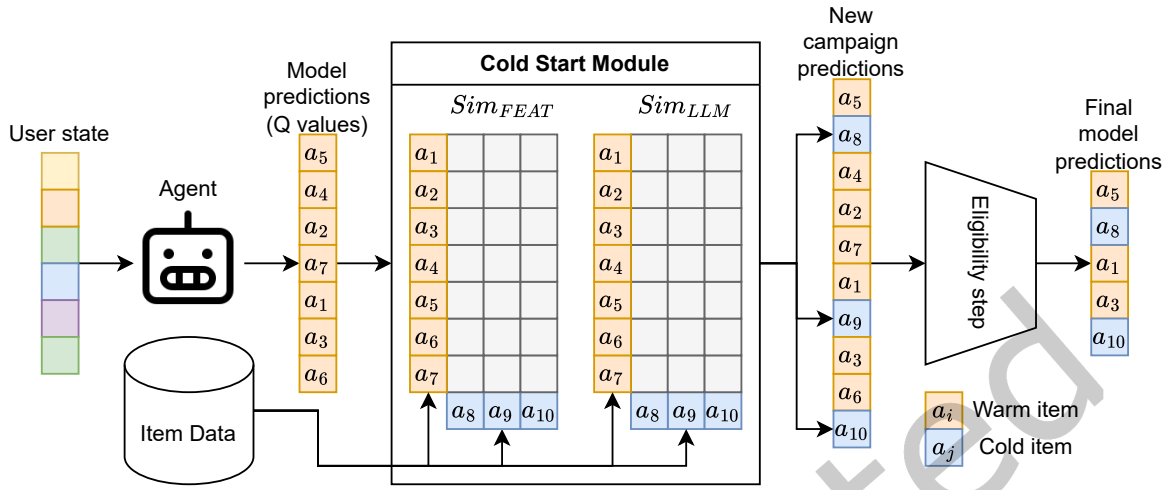

Fig. 2. CRAB functioning during inference. The cold-start module uses the agent predictions and similarities of cold start items to obtain the new campaign ranks. The eligibility step filters out items that are not eligible for selection.

takes into account several aspects, such as price, offer composition by using the various components (including components like data, voice, and extra packages), and the textual description of the offer.

To estimate the new Q-values, $Q(s, \hat{a})_{\pi_\theta}$, which represent the long-term expected reward of taking action $\hat{a}$ in state s under policy π_θ, we use the following formula:

$$Q(s, \hat{a})_{\pi_\theta} = \sum_{a}^{A'} Q(s, a)_{\pi_\theta} * \frac{(Sim_{FEAT}(a, \hat{a}) + Sim_{LLM}(a, \hat{a}))}{2} \tag{3}$$

where $\hat{a}$ represents one of the cold actions in the set $\hat{A}$ and A' are the warm actions with top-5 associated similarities. This equation combines three elements:

- **The old Q-values** $Q(s, a)_{\pi_\theta}$**:** these are provided by the trained model on the warm items.
- **Similarity of tabular features** Sim_{FEAT}**:** this represents the cosine similarity between the new and existing tabular features. These attributes include quantitative features like offer price and data allowance.
- **NLP similarity** Sim_{LLM}**:** this is the cosine similarity of offer description embeddings. The embeddings are generated using a pre-trained Italian sentence-BERT-based model [16]. It captures the similarity in the offers' textual descriptions.

By averaging these two similarity measures, we adjust the old Q scores to provide reasonable values for the new cold start items. This process, illustrated in Fig. 2, ensures that new campaigns can be launched with an effective initial ranking, mitigating the cold start problem.

3.2.8 Eligibility step. After the agent generates a complete, ranked list of all actions, including values for new items from the cold-start procedure, the framework proceeds to the eligibility step, a critical phase that allows to recommend real-world offers. Each recommended action is mapped to a specific set of offers from the catalog. This process involves filtering out offers that are ineligible due to various constraints, such as business price constraints, connection availability, customer accessibility reasons, or company rules. This step ensures that the recommendations presented to customers are practical and actionable.

4 Experimental Setup

In this section, we outline the setup of our offline and online experiments, designed to test the performance of the proposed CRAB framework against appropriate baselines. In addition, we conduct ablation studies to identify the key factors that contribute to its effectiveness.

4.1 Dataset

The company dataset regards upselling recommendations, and it has several unique characteristics. Unlike traditional recommendation datasets that track single-item interactions, our dataset captures multiple recommendations occurring on the same day. This is because recommendations are provided through human phone calls, and during a single call, customer care agents may suggest many items to the customer. Moreover, the company recommender system suggests the top five sorted offers to the customer care agent, but the items may not be suggested in order. It can happen because the customer explicitly requests a specific item during the call, or simply because the customer care agent decides not to recommend it in that order, based on how the call proceeds.

We use thirteen months of historical daily customer data provided by the company. The data is split temporally: the first eight months serve as training and validation sets, while the last five months are used for offline evaluation (test set). The validation set consists of a random 10% sample of episodes from the first eight months. The dataset contains daily customer information, including their status, actions proposed by the company, and customers' responses (accepted or rejected). Actions have been generated by the company's baseline recommender system, with varying frequency across customers.

The dataset used for training comprises records from over 7 million users, of whom 1.7 million exhibited activity within the specified time period. A total of 9.4 million actions were recorded during this period, with only 370 thousand actions meeting the acceptance criteria, yielding an acceptance rate below 4%. The mean episode length was 5.65 ± 9.09. Furthermore, 190 thousand customers churned during the considered timeframe.

We use this offline dataset to benchmark the CRAB framework against the company's recommender system over the last five months. Testing our method on a public dataset is currently not feasible because, to the best of our knowledge, no available dataset provides the essential combination of features. Specifically, our approach requires temporal data on users, detailed churn information, and a rich history of user interactions, including, in particular, user acceptance or rejection of proposed offers.

To provide deeper insights into the recommendation environment and the specific challenges our RL framework must overcome, we present a detailed analysis of the statistical and temporal characteristics of the upselling catalog. The action space consists of over 380 unique offers, ranging from standard data upgrades to streaming subscriptions and smart home devices. The distribution of item acceptances is highly skewed, exhibiting a pronounced long-tail pattern. While a small subset of foundational items receives moderate to high acceptance, the vast majority of items are either niche offers or items with minimal acceptances. This severe imbalance underscores the challenge of accurate value estimation in offline RL. It specifically motivates our use of CQL, which penalizes out-of-distribution actions and prevents the agent from overestimating the value of rarely accepted, long-tail items.

Moreover, the telecommunications upselling environment is highly dynamic. The catalog is not static; rather, offers are frequently introduced, bundled, or retired in response to seasonal trends, expiring promotions, and new marketing campaigns. Consequently, item interaction patterns often show irregular behavior, where specific items experience a spike in recommendations during a targeted campaign and then rapidly decline in interaction volume. This continuous temporal evolution and the frequent introduction of new offers necessitate the inclusion of our dedicated cold-start module (Section 3.2.7), ensuring that new campaigns can be effectively ranked even when they entirely lack historical RL training data.

Orthogonal to these temporal dynamics, the environment is further characterized by extreme feedback sparsity and multi-item interactions, where positive feedback is exceptionally rare. In this sparse and complex landscape, our framework captures the critical delayed churn signal to learn whether a given sequence of recommendations prevents or inadvertently triggers customer churn. Further details about the dataset can be found in Sbandi et al. [53].

4.2 Baselines

We evaluate our proposed CRAB framework against a diverse set of representative baseline models.

- **Multinomial Logistic Regression (MLR)** [19]: a linear model that classifies each possible offer and sorts the output odds to generate the list of recommended offers.
- **Alternating-Least-Squares with Weighted λ Regularization (ALS-WR)** [70]: a matrix factorization algorithm, known for its efficiency in handling sparse matrices, making it suitable for large-scale recommendation tasks.
- **LambdaMART** [6]: a learning-to-rank model that uses gradient boosting to produce personalized rankings.
- **LightGCN** [28]: a graph neural network performing link prediction on the bipartite graph made by users and items, where a link connects a user to an item if he bought it.
- **SASRec** [35]: a sequential model leveraging self-attention to capture users' sequential behavior patterns, enabling personalized next-item recommendations by modeling short and long term user preferences.
- **Company baseline**: the current non-deep learning recommender system used by the company.

All the baselines are trained on the first eight months of data and tested on the last five months. For MLR, ALS-WR, and LambdaMART, offers are rated as 1 (accepted), 0 (rejected), or 0.5 (not offered). All models are trained on the same data, enriched with the same auxiliary information used in CRAB, ensuring a fair evaluation. To ensure a fair and robust comparison, all baseline models were tuned using Optuna [2]. Search spaces and final hyperparameters are detailed Section A.

4.2.1 Company Baseline. The company's recommender system used in production is fundamentally different from our approach. Its primary objective is to optimize for immediate, short-term outcomes by predicting the likelihood of offer acceptance at each interaction. It does not employ a sequential decision-making framework and is not explicitly trained to minimize long-term churn, making it a suitable benchmark for a short-term focus recommendation strategy. Due to the company's privacy policy and business confidentiality, further information about the model cannot be made publicly available.

4.3 Evaluation Metrics

We evaluate the models using Recall@K, Coverage@K, Normalized Discounted Cumulative Gain@K (NDCG@K) and Precision@K (P@K). Metrics are computed using relevant actions generated by the original policy that were accepted. Moreover, we use specific RL metrics to compare models with each other, performing a grid search to find the best hyperparameter set. We decide to use Average TD error and Soft Off-Policy Classification (SOPC), evaluating the difference for each time-step t between the estimate for V_t and the discounted value estimate of V_{t+1} plus the actual reward gained from transitioning between states s_t and s_{t+1}, and gaps of action-value estimation between the successful episodes and all episodes respectively.

4.4 Hyperparameters Tuning

To decide the values for the parameters α,β,γ and δ of the reward Eq. (1), we focus on obtaining opposite distributions between churning and non-churning customers. In particular, the company requires that a churning customer who has accepted two offers is still regarded as a favorable outcome. This imposes the additional

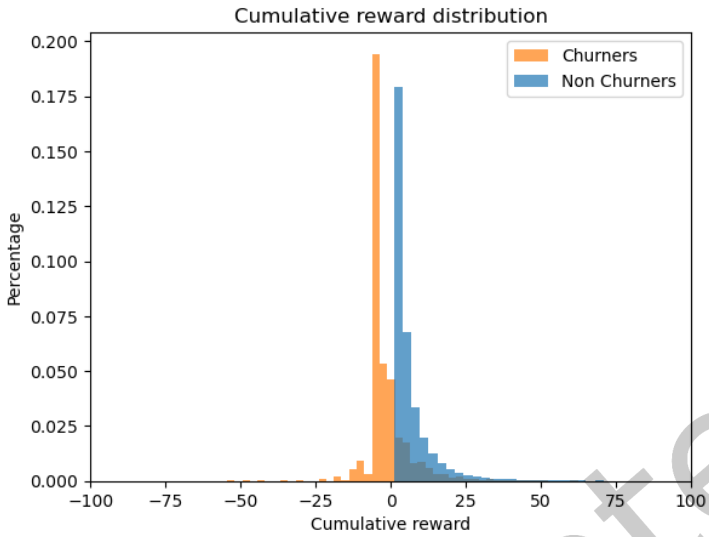

Fig. 3. Cumulative reward distribution for churners (left) and non-churners (right) customers.

constraint $\alpha + \gamma > 0 \wedge 2\alpha + \delta > 0$. To satisfy this and the other constraints introduced in 3.2.5, we select the following parameter values: $\alpha = 4, \beta = 1, \gamma = -3, \delta = -6$. Using these values, the cumulative reward distributions for clients who churn and clients who do not churn throughout an episode are shown in Fig. 3. The chosen values ensure that the cumulative reward is negative for most churning customers, while always being positive for non-churning customers, effectively aligning with the company's requirements.

We conduct a grid search to optimize the following hyperparameters:

- batch size: $\{256, 512, 1024\}$
- the number of critics: $\{2, 3\}$
- Activation function: $\{Swish, ReLU\}$
- the tradeoff factor in the iterative update for training the Q-function in CQL algorithm ϕ: $\{0.01, 0.1, 1\}$
- probability of ϵ-greedy ϵ: $\{0, 0.1, 0.2, 0.4\}$
- discount factor ρ: $\{0.1, 0.5, 0.7, 0.99\}$.
- learning rate: $\{6.25e^{-6}, 0.01\}$.

Fitted Q Evaluation (FQE) [40] is employed for offline policy evaluation. We use it to retrain a critic for each policy generated by the offline RL algorithms. From all the obtained models, we select the one that maximizes Recall@K on the validation set.

The best model has the following hyperparameters:

- batch size: 1024
- the number of critics: 3
- activation function: *swish*
- tradeoff factor ϕ: 0.01
- probability of ϵ-greedy ϵ: 0.2
- discount factor ρ: 0.99.
- learning rate: $6.25e^{-6}$

All subsequent results are based on this model configuration.

4.5 Code and privacy restrictions

For the implementation of the discrete CQL model, we use the Python *d3rlpy* library[55]. The original implementation only returns the best action for each state according to the Q-values. However, to use the ϵ-greedy algorithm, our RLRS model must provide a list of all possible actions, sorted and ranked by their Q-values. To achieve this, we modify the library to output the top k actions for a given user in a specific state. The updated version of the d3rlpy library can be found on our private Git repository[1] and installed for further development and experimentation. Due to the company's privacy policy and business confidentiality, neither the trained model nor the dataset can be made publicly available.

5 Results

In this section, we present the results of our extensive experiments, which demonstrate the superiority of the proposed CRAB framework. Additionally, we conduct ablation studies to identify the key factors contributing to its effectiveness.

5.1 Offline Comparison

Table 1. Performance comparison between baseline models and CRAB for NDCG@K, Precision@K and Mean Average Precision (MAP). **Bold** indicates the best result, while second-best is underlined. †indicates a statistically significant result according to a Friedman test with a significance level of 0.05.

Model	NDCG@5	NDCG@10	Precision@5	Precision@10	MAP
MLR	0.4041	0.4041	0.0844	0.0422	0.3990
ALS-WR	0.3015	0.3283	0.0802	0.0511	0.3146
LambdaMART	0.4073	0.4079	0.0923	0.0469	0.3353
SASRec	0.4191	**0.4691**	0.0990	0.0527	0.4242
LightGCN	0.4183	0.4218	0.0904	0.0559	0.4219
Company Baseline	0.3154	0.4163	0.0836	0.0544	0.3996
CRAB	**0.4233**†	0.4479†	**0.0994**†	**0.0582**†	**0.4246**†

As discussed in Section 3.2.3, we focus on $K = 5$ because customer service agents suggest the top 5 recommendations returned by the recommendation model. Our primary goal is to provide a desirable item in the top-5 recommendations. We also include results for $K = 10$ to assess broader ranking effects.

In Table 1, we compare all tested models by analyzing recommended offers for all customers. The results show how CRAB can consistently outperform all competitors on metrics calculated with $K = 5$, with the sequential model SASRec achieving the second best results. At $K = 10$, SASRec achieves better performance in NDCG, while CRAB is the second-best. This indicates that while other models might be tuned for optimizing longer lists, CRAB is suited for short-list recommendations. Moreover, CRAB's goal is multi-objective and not only focused on upselling.

Table 2. CRAB offline performance against the company baseline. The average rank of accepted action increases.

	Accepted actions average rank ↓
Company baseline	4.57
CRAB	**3.73**

[1]https://github.com/ashba93/crab

Table 2 shows that CRAB also has a higher average rank of accepted actions (**3.73**) compared to the company baseline model **4.57**. This difference indicates that CRAB's accepted actions are, on average, roughly one rank higher, providing a better customer experience.

5.2 Online Performance

Table 3. CRAB online performance in A/B test. During the month experimenting A/B test, average acceptance increased and churn ratio decreased.

Offer Acceptance ↑	*Customer Churn* ↓
+3.51%	**-4.95%**

Experimental Setup. To validate our approach, we conduct an A/B test on a live system serving millions of users. The experiment lasts one month. Our focus is centered on two key metrics: offer acceptance and churn. Incoming customer interactions are randomly assigned to one of two experimental groups. The control group (90%) receive recommendations generated by the existing company baseline model, while the treatment group (10%) receive recommendations from CRAB, reflecting a conservative deployment strategy to limit exposure risk. Over the testing period, the experiment recorded approximately 30,000 interactions.

Results. We evaluate the overall acceptance rate as a binary outcome per user: whether the user accepts any proposed offer during the call. As shown in Table 3, our method improves the offer acceptance rate by **+3.51%** over the baseline. Statistical significance is confirmed via Welch's T-test ($p < 0.05$). Regarding the customer churn, we track the event within a fixed observation window following the call. We achieve a -4.95% relative reduction in the customer churn rate compared to the baseline. This reduction is statistically significant ($p < 0.05$), as confirmed by Welch's T-test.

5.3 Model Analysis and Hyperparameter Tuning

5.3.1 Impacts of the Number of Critics. In our experiments, we tested models with two and three critics. As explained in Section 3.2.6, we refer to critics as the number of Q functions for the ensemble. The results, as illustrated in Fig. 4, indicate that using three critics yields significantly higher performance compared to employing only two. However, due to constraints in computational resources, we could not test configurations with more than three critics, as it would considerably increase the algorithm's time. In particular, training with three critics necessitates approximately three times the time per epoch compared to training with two critics.

5.3.2 Impacts of the ϵ-Greedy Probability. In order to optimize the balance between exploration and exploitation, the efficacy of the model was evaluated using TD error and SOPC metrics, with values of $\epsilon \in \{0, 0.1, 0.2, 0.4\}$. The results are shown in Fig. 5. As illustrated, the model exhibits optimal performance with $\epsilon = 0.2$, indicating a good exploration rate while not preventing exploitation as much as higher values do. Action values in successful episodes are higher than in the others. The plot also illustrates that the lowest performance is attained when $\epsilon = 0$ values are not equal to zero, proving the ϵ-greedy approach to be beneficial.

5.3.3 Impacts of the Discount Factor - Delayed Feedback Analysis. We investigated the influence of delayed feedback of customer churn on model performance by varying the discounted factor ρ. Preventing churn is one of the main company's objectives, and churn events typically mark the conclusion of a customer episode, so it is crucial to consider high values of ρ. In Fig. 6, we present the NDCG obtained by different models trained with ρ equal to $\{0.1, 0.5, 0.7, 0.99\}$. The outcomes demonstrate that a high discount factor ($\rho = 0.99$) markedly enhances the model's efficacy, suggesting that prioritizing long-term rewards facilitates the model's capacity not only to more accurately account for churn, but also to provide better recommendations.

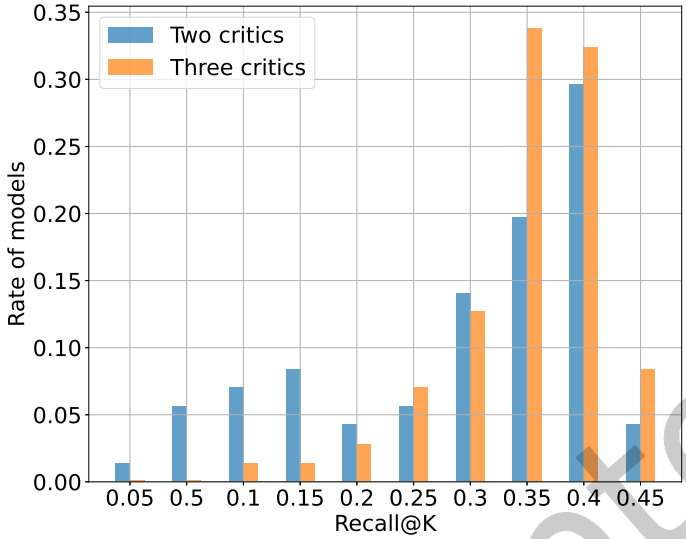

Fig. 4. Distribution of Recall@K within all models in grid search with two critics and three critics.

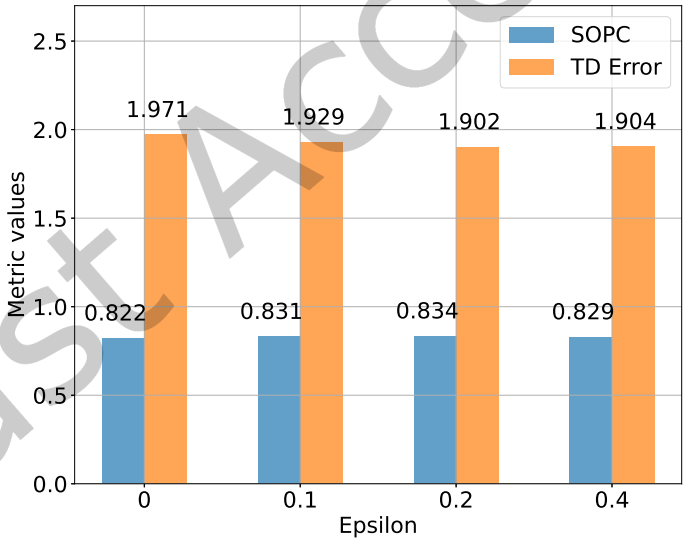

Fig. 5. RL metrics results obtained for different ϵ.

5.4 Ablation Study: Impact of Auxiliary Model Outputs

To assess the impact of auxiliary models, we compared the NDCG of CRAB variants: the first variant, CRAB-1, incorporates auxiliary customer predictions into the state representation. In contrast, the second variant,

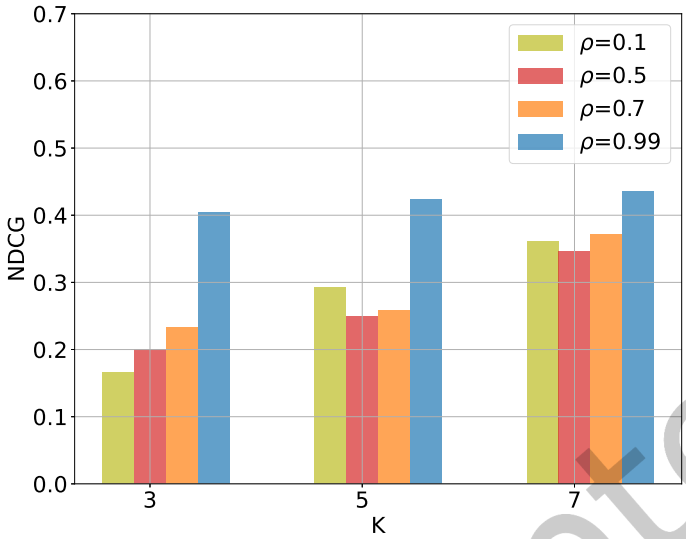

Fig. 6. Results obtained with different ρ values.

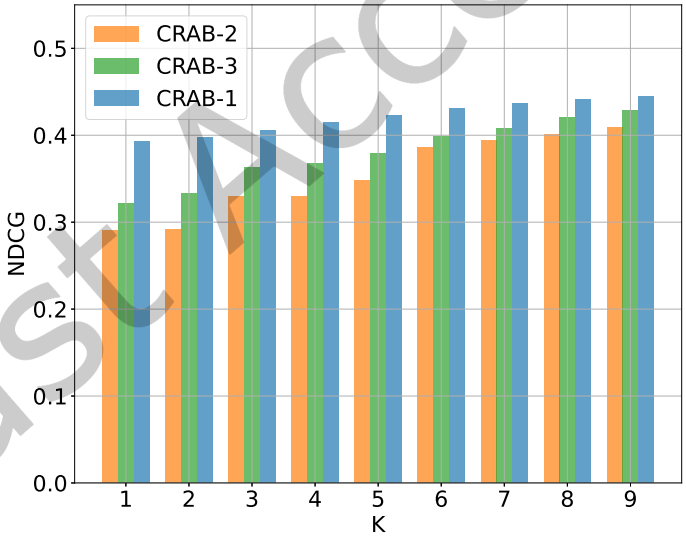

Fig. 7. Results comparing CRAB with (CRAB-1), without (CRAB-2), and using 50% random (CRAB-3) auxiliary model outputs.

CRAB-2, does not. Finally, to investigate whether performance scales with the amount of information provided, a third variant, CRAB-3, includes a random 50% subset of the auxiliary predictions. As shown in Fig. 7, CRAB-2 exhibits performance comparable to that of the company baseline model at K=5. CRAB-1 demonstrates superior

performance compared to both CRAB-2 and CRAB-3, thereby underscoring the added value of incorporating auxiliary model outputs into the state representation. This comparison not only highlights the benefit of this enrichment but also suggests that performance scales with the amount of auxiliary information provided, as CRAB-3 consistently places between the other variants.

The critical importance of this step is illustrated when contextualizing these results with our main Table 1. The performance of the CRAB-2 variant, which lacks auxiliary data, drops significantly. This means that without this information, our model would perform worse than top baseline competitors. Therefore, this is the key reason for CRAB's better performance.

6 Limitations

While our framework demonstrates strong performance and provides significant business value, we acknowledge several limitations in our study.

First, the performance improvements observed in the offline experiments over the baselines may appear modest. However, as reported by Deffayet et al. [15], there is an inherent misalignment in evaluating RL-based recommender systems using standard offline metrics. In our case, the RL model is explicitly optimized to maximize long-term cumulative rewards. Conversely, standard offline ranking metrics measure short-term, immediate relevance. Consequently, offline evaluation protocols inherently struggle to fully capture the primary long-term strengths of RL agents, which may explain why RL approaches can show only marginal gains in short-term offline settings despite learning superior long-term policies. Nevertheless, in a large-scale industrial environment, even small offline improvements translate to substantial real-world impact, as evidenced by the reduction in customer churn during A/B test.

Second, regarding the baseline comparisons, our offline evaluation focuses on comparing CRAB against the company's production baseline and state-of-the-art traditional and sequential recommendation models. We did not include other prior offline RL algorithms in our benchmark. This decision was driven by the strict constraints of our industrial application: adapting standard offline RL algorithms to our delayed-reward churn formulation and integrating pre-existing auxiliary business models would both require non-trivial architectural modifications. Consequently, we focused on CQL as our foundational RL method due to its proven safety against out-of-distribution overestimation in static datasets.

7 Conclusions and Future Work

In this paper, we addressed the critical challenge of customer churn in the telecommunications industry, a problem compounded by the delayed nature of customer feedback. We introduced CRAB (Churn Reduction Agent-Based) , a recommender system designed to mitigate delayed feedback churn cases through the utilization of offline RL. This approach employs historical customer interactions, product usage patterns, and user feedback to train the model, without requiring real-time user interactions. Unlike traditional recommender systems that optimize short-term metrics such as immediate clicks or offer acceptances, CRAB is explicitly designed to learn a policy that balances the immediate goal of upselling with the crucial long-term objective of reducing churn.

Our primary contribution is a practical and effective methodology for integrating delayed feedback into the RL training process. By defining episodes that extend into a future look-ahead window, and by designing a reward function that heavily penalizes future churn events, CRAB learns to avoid recommendations that seem effective in the short term, but carry a high risk of leading to customer dissatisfaction and churn. Our offline results show that the CRAB framework generates more diverse and relevant recommendations than the original company policy. Findings demonstrate the consistent superiority of CRAB over other baseline models. In particular, we demonstrated a novel and highly effective method for enriching the agent's state representation. By incorporating the predictive outputs of pre-existing, independently trained auxiliary models, we provide the agent with a richer,

more holistic view of the customer. Our ablation studies confirmed that this enrichment is fundamental to CRAB's superior performance.

The empirical validation of our framework, conducted on a large-scale proprietary dataset from a major telecommunications company, yielded compelling results. In extensive offline experiments, CRAB consistently outperformed a wide range of state-of-the-art baselines and the existing company policy. More importantly, subsequent live A/B testing in a real-world production environment confirmed these findings, demonstrating a 4.95% reduction in customer churn alongside a 3.51% increase in offer acceptance.

In addition to improving performance, the ability of offline RL to operate without direct interaction with the customer can reduce the risks associated with online RL, such as privacy concerns and ethical issues. Our findings provide a strong case for adopting offline RL in the recommendation systems domain. Following the successful online testing, CRAB has been deployed to all users.

Nonetheless, there are various directions for future work.

- Currently, CRAB handles the cold start problem for new items using similarity-based Q-value estimation. However, recommendations for new users are missing. A method to address the cold-start issue for users needs to be designed.
- The current framework is applied only to upsell offers. We plan to extend its application beyond upselling offers and also to include retention strategies.
- Exploring more advanced offline RL algorithms beyond CQL could yield further performance gains.

In conclusion, CRAB shows that offline RL is an efficient way to solve difficult, real-world business problems. By effectively managing delayed rewards and leveraging auxiliary models for richer state representations, CRAB offers a scalable solution for improving upselling and reducing customer churn.

References

[1] M Mehdi Afsar, Trafford Crump, and Behrouz Far. 2022. Reinforcement learning based recommender systems: A survey. *Comput. Surveys* 55, 7 (2022), 1–38.

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Association for Computing Machinery, New York, NY, USA, 2623–2631.

[3] Jose A Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. 2019. RUDDER: Return decomposition for delayed rewards. *Advances in Neural Information Processing Systems* 32 (2019).

[4] Nasim Baharisangari, Yash Paliwal, and Zhe Xu. 2024. Counterfactually-Guided Causal Reinforcement Learning with Reward Machines. In *2024 American Control Conference (ACC)*. IEEE, 522–527.

[5] Debmalya Biswas. 2020. Delayed Rewards in the Context of Reinforcement Learning based Recommender Systems. In *AAI4H@ ECAI*. 49–53.

[6] Christopher Burges, Robert Ragno, and Quoc Le. 2006. Learning to rank with nonsmooth cost functions. *Advances in Neural Information Processing Systems* 19 (2006).

[7] Tianchi Cai, Shenliao Bao, Jiyan Jiang, Shiji Zhou, Wenpeng Zhang, Lihong Gu, Jinjie Gu, and Guannan Zhang. 2023. Model-free Reinforcement Learning with Stochastic Reward Stabilization for Recommender Systems. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Association for Computing Machinery, New York, NY, United States, 2179–2183.

[8] Chang Chen, Yi-Fu Wu, Jaesik Yoon, and Sungjin Ahn. 2022. Transdreamer: Reinforcement learning with transformer world models. *arXiv preprint arXiv:2202.09481* (2022).

[9] Hung-Chen Chen and Arbee L. P. Chen. 2001. A Music Recommendation System Based on Music Data Grouping and User Interests. In *Proceedings of the Tenth International Conference on Information and Knowledge Management* (Atlanta, Georgia, USA) *(CIKM '01)*. Association for Computing Machinery, New York, NY, USA, 231–238. doi:10.1145/502585.502625

[10] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. 2021. Decision transformer: Reinforcement learning via sequence modeling. *Advances in Neural Information Processing Systems* 34 (2021), 15084–15097.

[11] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H Chi. 2019. Top-k off-policy correction for a REINFORCE recommender system. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. Association for Computing Machinery, New York, NY, USA, 456–464.
[12] Minmin Chen, Bo Chang, Can Xu, and Ed H Chi. 2021. User response models to improve a reinforce recommender system. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 121–129.
[13] Xiaocong Chen, Siyu Wang, Julian McAuley, Dietmar Jannach, and Lina Yao. 2024. On the opportunities and challenges of offline reinforcement learning for recommender systems. *ACM Transactions on Information Systems* 42, 6 (2024), 1–26.
[14] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for YouTube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. Association for Computing Machinery, New York, NY, USA, 191–198.
[15] Romain Deffayet, Thibaut Thonet, Jean-Michel Renders, and Maarten De Rijke. 2023. Offline evaluation for reinforcement learning-based recommendation: a critical issue and some alternatives. In *ACM SIGIR Forum*, Vol. 56. ACM New York, NY, USA, Association for Computing Machinery, New York, NY, USA, 1–14.
[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, volume 1 (long and short papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186.
[17] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. 2019. Diversity is all you need: Learning skills without a reward function. *Proceedings of the International Conference on Learning Representations (ICLR)* (2019).
[18] Benjamin Eysenbach, Tianjun Zhang, Sergey Levine, and Russ R Salakhutdinov. 2022. Contrastive learning as goal-conditioned reinforcement learning. *Advances in Neural Information Processing Systems* 35 (2022), 35603–35620.
[19] Morten W Fagerland, David W Hosmer, and Anna M Bofin. 2008. Multinomial goodness-of-fit tests for logistic regression models. *Statistics in Medicine* 27, 21 (2008), 4238–4253.
[20] Ching Fang and Kimberly L Stachenfeld. 2024. Predictive auxiliary objectives in deep RL mimic learning in the brain. *Proceedings of the International Conference on Learning Representations (ICLR)* (2024).
[21] Scott Fujimoto, David Meger, and Doina Precup. 2019. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*. PMLR, 2052–2062.
[22] Jasmina Gajcin and Ivana Dusparic. 2024. Redefining counterfactual explanations for reinforcement learning: Overview, challenges and opportunities. *Comput. Surveys* 56, 9 (2024), 1–33.
[23] Chengqian Gao, Ke Xu, Kuangqi Zhou, Lanqing Li, Xueqian Wang, Bo Yuan, and Peilin Zhao. 2022. Value penalized Q-learning for recommender systems. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Association for Computing Machinery, New York, NY, USA, 2008–2012.
[24] Carlos A Gomez-Uribe and Neil Hunt. 2015. The Netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)* 6, 4 (2015), 1–19.
[25] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. 2020. Dream to control: Learning behaviors by latent imagination. *Proceedings of the International Conference on Learning Representations (ICLR)* (2020).
[26] Beining Han, Zhizhou Ren, Zuofan Wu, Yuan Zhou, and Jian Peng. 2022. Off-policy reinforcement learning with delayed rewards. In *International Conference on Machine Learning*. PMLR, 8280–8303.
[27] Matthew Hausknecht, Peter Stone, and Off-policy Mc. 2016. On-policy vs. off-policy updates for deep reinforcement learning. In *Deep Reinforcement Learning: Frontiers and Challenges, IJCAI 2016 Workshop*. AAAI Press New York, NY, USA.
[28] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 639–648.
[29] Yan-e Hou, Wenbo Gu, WeiChuan Dong, and Lanxue Dang. 2023. A Deep Reinforcement Learning Real-Time Recommendation Model Based on Long and Short-Term Preference. *International Journal of Computational Intelligence Systems* 16, 1 (Jan. 2023), 4. doi:10.1007/s44196-022-00179-1
[30] Bingquan Huang, Mohand Tahar Kechadi, and Brian Buckley. 2012. Customer churn prediction in telecommunications. *Expert Systems with Applications* 39, 1 (2012), 1414–1425.
[31] Chia-Chun Hung, Timothy Lillicrap, Josh Abramson, Yan Wu, Mehdi Mirza, Federico Carnevale, Arun Ahuja, and Greg Wayne. 2019. Optimizing agent behavior over long time scales by transporting value. *Nature Communications* 10, 1 (2019), 5223.
[32] Shin-Yuan Hung, David C Yen, and Hsiu-Yu Wang. 2006. Applying data mining to telecom churn management. *Expert Systems with Applications* 31, 3 (2006), 515–524.
[33] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. 2016. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397* (2016).
[34] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4 (1996), 237–285.

[35] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 197–206.
[36] Mozhgan Karimi, Dietmar Jannach, and Michael Jugovac. 2018. News recommender systems – Survey and roads ahead. *Information Processing and Management* 54, 6 (2018), 1203–1227. doi:10.1016/j.ipm.2018.04.008
[37] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. 2020. Conservative Q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems* 33 (2020), 1179–1191.
[38] Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. 2020. Reinforcement learning with augmented data. *Advances in Neural Information Processing Systems* 33 (2020), 19884–19895.
[39] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. 2020. CURL: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*. PMLR, 5639–5650.
[40] Hoang Le, Cameron Voloshin, and Yisong Yue. 2019. Batch policy learning under constraints. In *International Conference on Machine Learning*. PMLR, 3703–3712.
[41] George Lekakos and Petros Caravelas. 2008. A Hybrid approach for movie recommendation. *Multimedia Tools Appl.* 36 (01 2008), 55–70. doi:10.1007/s11042-006-0082-7
[42] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. 2020. Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. *CoRR* abs/2005.01643 (2020). arXiv:2005.01643 https://arxiv.org/abs/2005.01643
[43] Timothy Paul Lillicrap, Jonathan James Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver, and Daniel Pieter Wierstra. 2020. Continuous control with deep reinforcement learning. US Patent 10,776,692.
[44] Clare Lyle, Mark Rowland, Georg Ostrovski, and Will Dabney. 2021. On the effect of auxiliary tasks on representation dynamics. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 1–9.
[45] Timothy A Mann, Sven Gowal, Ray Jiang, Huiyi Hu, Balaji Lakshminarayanan, and Andras Gyorgy. 2018. Learning from delayed outcomes with intermediate observations. *arXiv preprint arXiv:1807.09387* (2018).
[46] Yixiu Mao, Qi Wang, Chen Chen, Yun Qu, and Xiangyang Ji. 2024. Offline reinforcement learning with OOD state correction and OOD action suppression. *Advances in Neural Information Processing Systems* 37 (2024), 93568–93601.
[47] Mitsuhiko Nakamoto, Simon Zhai, Anikait Singh, Max Sobol Mark, Yi Ma, Chelsea Finn, Aviral Kumar, and Sergey Levine. 2023. Cal-QL: Calibrated offline RL pre-training for efficient online fine-tuning. *Advances in Neural Information Processing Systems* 36 (2023), 62244–62269.
[48] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).
[49] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. 2017. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*. PMLR, 2778–2787.
[50] Zhiyong Peng, Changlin Han, Yadong Liu, and Zongtan Zhou. 2023. Weighted policy constraints for offline reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 9435–9443.
[51] Yuhang Ran, Yi-Chen Li, Fuxiang Zhang, Zongzhang Zhang, and Yang Yu. 2023. Policy Regularization with Dataset Constraint for Offline Reinforcement Learning. In *ICML*.
[52] Marlesson RO Santana, Luckeciano C Melo, Fernando HF Camargo, Bruno Brandão, Anderson Soares, Renan M Oliveira, and Sandor Caetano. 2020. MARS-Gym: A Gym framework to model, train, and evaluate Recommender Systems for Marketplaces. In *2020 International Conference on Data Mining Workshops (ICDMW)*. IEEE, 189–197.
[53] Alessandro Sbandi, Federico Siciliano, and Fabrizio Silvestri. 2025. TIM-Rec: Explicit Sparse Feedback on Multi-Item Upselling Recommendations in an Industrial Dataset of Telco Calls. In *Proceedings of the Nineteenth ACM Conference on Recommender Systems*. 865–873.
[54] Ben Schafer, Joseph Konstan, and John Riedl. 1999. Recommender Systems in E-Commerce. *1st ACM Conference on Electronic Commerce, Denver, Colorado, United States* (10 1999). doi:10.1145/336992.337035
[55] Takuma Seno and Michita Imai. 2022. d3rlpy: An offline deep reinforcement learning library. *Journal of Machine Learning Research* 23, 315 (2022), 1–20.
[56] Jianzhun Shao, Yun Qu, Chen Chen, Hongchang Zhang, and Xiangyang Ji. 2023. Counterfactual conservative Q-learning for offline multi-agent reinforcement learning. *Advances in Neural Information Processing Systems* 36 (2023), 77290–77312.
[57] RR Sharma and S Rajan. 2017. Evaluating prediction of customer churn behavior based on artificial bee colony algorithm. *International Journal of Engineering and Computer Science* 6, 1 (2017), 20017–20021.
[58] Yutaka Shimizu, Joey Hong, Sergey Levine, and Masayoshi Tomizuka. 2024. Strategically Conservative Q-Learning. In *ICML*.
[59] Brent Smith and Greg Linden. 2017. Two decades of recommender systems at Amazon. com. *IEEE Internet Computing* 21, 3 (2017), 12–18.
[60] Quan Vuong, Aviral Kumar, Sergey Levine, and Yevgen Chebotar. 2022. Dasco: Dual-generator adversarial support constrained offline reinforcement learning. *Advances in Neural Information Processing Systems* 35 (2022), 38937–38949.

[61] Dingrong Wang, Krishna P. Neupane, and Qi Yu. 2024. Evidential Conservative Q-Learning for Dynamic Recommendations. In *ICLR*.
[62] Hang Wang, Sen Lin, and Junshan Zhang. 2021. Adaptive ensemble Q-learning: Minimizing estimation bias via error feedback. *Advances in Neural Information Processing Systems* 34 (2021), 24778–24790.
[63] Teng Xiao and Donglin Wang. 2021. A General Offline Reinforcement Learning Framework for Interactive Recommendation. In *AAAI Conference on Artificial Intelligence*.
[64] Junghyuk Yeom, Yonghyeon Jo, Jeongmo Kim, Sanghyeon Lee, and Seungyul Han. 2024. Exclusively penalized Q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems* 37 (2024), 113405–113435.
[65] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. 2020. MOPO: Model-based offline policy optimization. *Advances in Neural Information Processing Systems* 33 (2020), 14129–14142.
[66] Jing Zhang, Linjiajie Fang, Kexin Shi, Wenjia Wang, and Bing-Yi Jing. 2024. Q-distribution guided Q-learning for offline reinforcement learning: Uncertainty penalized Q-value via consistency model. *Advances in Neural Information Processing Systems* 37 (2024), 54421–54462.
[67] Xiao Zhang, Haonan Jia, Hanjing Su, Wenhan Wang, Jun Xu, and Ji-Rong Wen. 2021. Counterfactual Reward Modification for Streaming Recommendation with Delayed Feedback. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, Canada) *(SIGIR '21)*. Association for Computing Machinery, New York, NY, USA, 41–50. doi:10.1145/3404835.3462892
[68] Yi Zhang, Ruihong Qiu, Jiajun Liu, and Sen Wang. 2024. ROLeR: Effective Reward Shaping in Offline Reinforcement Learning for Recommender Systems. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management* (Boise, ID, USA) *(CIKM '24)*. Association for Computing Machinery, New York, NY, USA, 3269–3278. doi:10.1145/3627673.3679633
[69] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. 2018. DRN: A Deep Reinforcement Learning Framework for News Recommendation. In *Proceedings of the 2018 World Wide Web Conference* (Lyon, France) *(WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 167–176. doi:10.1145/3178876.3185994
[70] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. 2008. Large-scale parallel collaborative filtering for the Netflix prize. In *Algorithmic Aspects in Information and Management: 4th International Conference, AAIM 2008, Shanghai, China, June 23-25, 2008. Proceedings 4*. Springer, 337–348.
[71] Zhiyuan Zhou, Andy Peng, Qiyang Li, Sergey Levine, and Aviral Kumar. 2025. Efficient online reinforcement learning fine-tuning need not retain offline data. *Proceedings of the International Conference on Learning Representations (ICLR)* (2025).

A Baselines Hyperparameters Tuning

To ensure a fair comparison among all baseline methods, we performed hyperparameter optimization using Optuna [2]. All models were trained and validated using the same data splitting strategy, applied uniformly across all methods. Specifically, for each user, we sorted their interactions chronologically and split them into three disjoint sets, as described in Section 4.1: the last five months are used exclusively for test evaluation, while the first eight months were used for train and validation (split 90%-10%). Model selection and optimization decisions were based solely on performance on the validation set.

In the tuning process, we optimized NDCG@5 as the primary metric. This choice reflects our interest in accurately ranking the top few items, which is particularly important in our recommendation context. As discussed in Section 4.1, focusing on NDCG@5 aligns with our goal, where only top results are visible to the customer care agent during call recommendations.

In each Optuna study, we fixed a number of trials (up to 20), with pruning enabled to discard poorly performing configurations early. We report the hyperparameters tuned for each baseline with the final values selected through the optimization process.

MLR:

- L2 regularization : $\in [1e-4, 1e2]$
- Solver: $\{lbfgs, saga\}$
- Max iterations: $\in [100, 1000]$

ALS-WR:

- Latent factors: $\in [10, 200]$
- Regularization: $\in [1e-4, 10]$

- Number of iterations: $\in [10, 100]$

LambdaMART:

- Number of trees: $\in [100, 1000]$
- Max tree depth: $\in [3, 10]$
- Learning rate: $\in [1e-3, 0.1]$
- Subsample ratio $\in [0.6, 1]$

LightGCN:

- learning rate: $\in [1e-4, 0.1]$
- embedding dimension: $\in [16, 128]$
- number of layers: $\in [1, 3]$

SASRec:

- Attention blocks: $\in [1, 3]$
- Attention heads: $\in [1, 3]$
- Batch size: $\in [128, 2048]$
- Learning rate : $\in [1e-4, 1e-1]$
- Embedding size : $\in [16, 1024]$.

Following the optimization procedure, we identified the best-performing hyperparameter set for each baseline model on the validation set. The final values used for the test set evaluation are detailed in Table 4.

Table 4. Best hyperparameter configurations for baseline models.

Model	**Hyperparameter**	**Best Value**
MLR	L2 regularization	4e-3
	Solver	*lbfgs*
	Max iterations	346
ALS-WR	Latent factors	152
	Regularization	0.1
	Number of iterations	73
LambdaMART	Number of trees	497
	Max tree depth	7
	Learning rate	0.05
	Subsample ratio	0.8
LightGCN	Learning rate	3e-3
	Embedding dimension	54
	Number of layers	2
SASRec	Attention blocks	2
	Attention heads	2
	Batch size	985
	Learning rate	5e-3
	Embedding size	464

This unified tuning approach ensures that all models were evaluated under their best configurations according to the same protocol, making our baseline comparisons robust.

Received 29 November 2024; revised 17 April 2026; accepted 18 April 2026