A Self-Triggered Agentic Push Recommendation System

Zhao—Yu Zhang*
zhaoyu.1@bytedance.com
ByteDance
Beijing, China

Qingying Chen*
chenqingying@bytedance.com
ByteDance
Beijing, China

Chunyuan Zheng*
zhengchunyuan@bytedance.com
ByteDance, Peking University
Beijing, China

Jing Zhou
zhoujing.23@bytedance.com
ByteDance
Beijing, China

Jian Sun
sunjian.2022@bytedance.com
ByteDance
Beijing, China

Siqi Chen
chensiqi.2025@bytedance.com
ByteDance
Beijing, China

Leiying Chen
chenleiying@bytedance.com
ByteDance
Beijing, China

Chuan Zhou
chuanzhou@bytedance.com
ByteDance
Beijing, China

Huiyou Jiang†
jianghuiyou.jhy@bytedance.com
ByteDance
Beijing, China

Xin Tao
taoxin.xt@bytedance.com
ByteDance
Beijing, China

Haoxuan Li
hxli@stu.pku.edu.cn
Peking University
Beijing, China

Zhouchen Lin
zlin@pku.edu.cn
Peking University
Beijing, China

Abstract

Push notification is a critical recommendation scenario on large-scale platforms, allowing the system to proactively reach users outside the application to improve long-term re-engagement. However, designing an optimal push system requires handling a complex action space for the "whether and when" delivery problem under strict system resource constraints, i.e., systems cannot naively evaluate every user at every second to find the optimal delivery moment. Existing solutions typically fall into two paradigms: the first is a two-stage approach that relies on offline user-level uplift modeling combined with integer programming solvers to allocate optimal delivery times and frequencies. The second periodically activates the push system to decide whether to send a push. However, the former cannot effectively utilize real-time information, and the latter results in resource inefficiency. In addition, such a multi-stage solution suffers from local optima. In this paper, we propose a proactive, self-triggered end-to-end agentic push recommendation system, which is already fully deployed at Douyin with over 1 billion users, **allowing the system to generate push time and determine whether to send in a closed loop with both real-time effectiveness and efficiency.** Specifically, the agentic system consists of two decision transformer-based agents: a planning agent that determines when to schedule the next push time with a gated ordinal regression method, and an action-execution agent that decides whether to send a push based on a trajectory reward. In addition, we further introduce a lightweight filtering agent to both control the computational overhead and act as a crucial safeguard against unreasonable planning behaviors. Extensive experiments demonstrate that this approach can both increase user active days by **0.2843%** and reduce push permission disablement by **1.9089%**. In addition, adding the filtering agent can reduce the computational overhead by **79.42%**.

CCS Concepts

• **Information systems** → **Recommender systems**.

Keywords

Agentic Recommender Systems, Push Notification, Generative Recommendation

ACM Reference Format:
Zhao—Yu Zhang, Qingying Chen, Chunyuan Zheng, Jing Zhou, Jian Sun, Siqi Chen, Leiying Chen, Chuan Zhou, Huiyou Jiang, Xin Tao, Haoxuan Li, and Zhouchen Lin. 2026. A Self-Triggered Agentic Push Recommendation System. In *Proceedings of RecSys '26.* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

*Zhao—Yu Zhang, Qingying Chen, and Chunyuan Zheng contributed equally.
†Huiyou Jiang is the corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

RecSys '26, Minneapolis, MN

© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Push notification is an important recommendation channel because it enables the system to proactively reach users outside the application, rather than waiting for them to launch the application on their own [19, 23, 26]. As shown in Figure 1, when users enable the push permission (left), they will receive the notification (right). At the same time, push notification is also a challenging problem: the system must decide whether and when to deliver a notification [18, 27]. Well-designed pushes can improve re-engagement

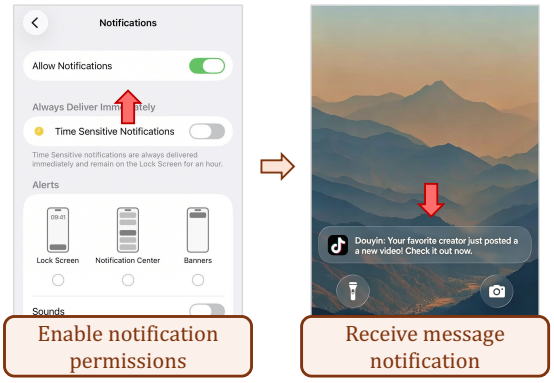


Figure 1: Push notification examples.

and user activity, but poorly timed ones lead to push notification disablement because of user fatigue [3, 22, 25].

Solving the "whether and when" problem is highly non-trivial, since push delivery is a continuous and sequenced optimization problem [1, 15]. To find the optimal delivery moment, a naive approach would be to evaluate every user every second. However, in large-scale industrial systems, this per-second real-time ranking is impractical due to system resource constraints. Existing solutions can be divided into two main paradigms. The first is a two-stage approach that relies on offline user-level uplift modeling to calculate the positive and negative value of each possible timing and frequency, and then uses integer programming solvers to solve optimal times and frequencies [9, 29]. The second periodically activates push systems based on a pre-defined short time interval to decide whether to send a push [2, 13, 33].

However, these methods have several limitations. First, making a decision based on a pre-defined time points and frequencies without considering timely information, such as recent user activity and push content, will lead to unreasonable pushes. For example, a push may not be sent even when the trajectory suggests that a current push could be valuable. In addition, activating the system periodically will result in too many action-execution processes, consuming a lot of resources since the item sorting involves heavy computation. Finally, the multi-stage framework may find the local optima of each stage, leading to an overall suboptimal performance.

To address these issues, we propose a proactive, self-triggered end-to-end agentic push recommendation system, which has the ability to plan and execute action, in a closed loop with both real-time effectiveness and efficiency. Specifically, the agentic system is composed of three agents: a planning agent, an action-execution agent, and a filtering agent. The overall pipeline works as follows. First, the planning agent provides the predicted next push time. Second, when the time arrives, the action-execution agent decides whether to send such a push to users. At the end of this loop, the planning agent updates a new push time. Finally, we introduce a lightweight filtering agent that screens out both low-value requests before they consume heavy computation and unreasonable planning behaviors to keep the push system safe.

We summarize our main contributions as follows:

- We reformulate industrial push recommendation as a self-triggered closed-loop decision problem, and propose a novel agentic end-to-end push recommendation system, which proactively generates the next push time (planning agent) and determine whether to push (action-execution agent), with a lightweight filtering agent to both control the computational overhead and ensure system safety.
- For push time generation, we propose a novel gated ordinal regression method to effectively inject target returns. For action-execution, we propose a value-guided learning method to correct suboptimal behaviors in offline logs.
- The proposed pipeline has been fully deployed on the Douyin app with over 1 billion users, showing that such a pipeline can both improve daily active users and reduce push permission disablement rate. In addition, we show the filtering agent can significantly reduce the computational overhead.

2 Related Work

2.1 Push Notification Systems

Push notification systems are multi-dimensional decision making systems that involve the content to send, and more importantly, *whether* and *when* to send. The problem of determining push content has become standard in industrial systems [14, 32, 34], and in this paper we focus on the "whether and when" problem. To solve this problem, the first group of approaches focuses more on the timing aspect, usually conducting offline user-level uplift modeling and planning by integer programming solvers in advance for *when* the system should send the push notification [9, 12, 17, 29, 31]. However, these methods of scheduling push notifications in advance result in reduced timeliness and diminish their effectiveness. Meanwhile, another group of approaches bypass the timing issue by implementing a fixed time interval triggering mechanism to determine whether to send a notification [4, 6, 21, 24]. However, time slices that are too broad result in lower-value push notifications, while slices that are too granular consume excessive resources. The limitations of these previous push systems call for a multidimensional, collaborative action-execution mechanism that ensures both real-time performance and push efficiency.

2.2 Generative Reinforcement Learning

Reinforcement learning has been widely studied for optimizing long-term objectives in recommendation, including slate recommendation, feed ranking, and notification delivery [10, 20, 30, 35]. However, deploying conventional value-based or policy-gradient methods in large-scale industrial systems can sometimes present challenges, as these approaches may be susceptible to unstable bootstrapping and sensitivity to distribution shifts in offline logs. To address this, recent generative RL methods offer an alternative by formulating sequential decision-making as a conditional sequence modeling problem. Decision transformer (DT) [7] predicts actions conditioned on return-to-go, historical states and past actions through a supervised learning objective. Trajectory Transformer [11] further treats states, actions, and rewards as a unified trajectory sequence for long-horizon planning, and Q-learning decision transformer [28] further improves trajectory stitching by relabeling returns with dynamic programming, thereby combining

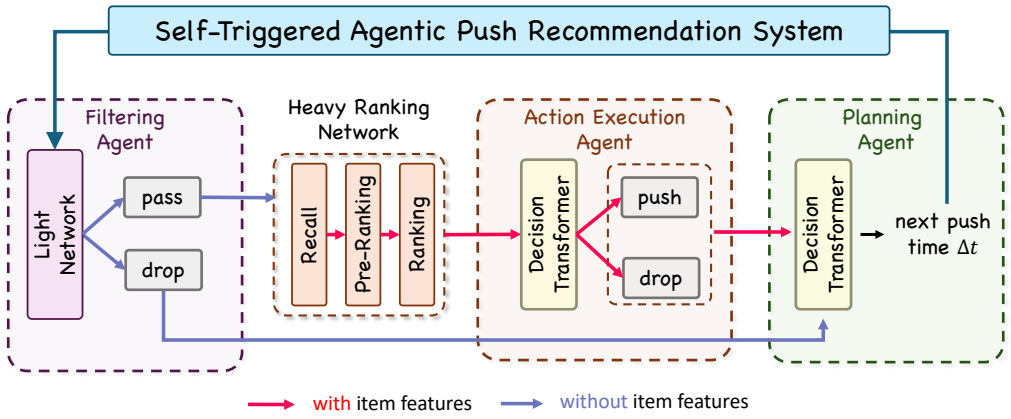


Figure 2: Overall framework.

the stitching ability of Q-learning with the training stability of DT. This paradigm has also shown promise in industrial multi-objective systems, such as notification optimization [16] and auto-bidding [8]. Our work follows this direction but targets proactive push recommendation, where the system must jointly generate *when* to activate the next request and decide *whether* to send after activation.

3 Method

3.1 Problem Setup

Generally, optimizing a push recommendation system for long-term engagement can be formulated as a sequential action-execution problem over a continuous time horizon. Suppose there are T timestamps to push within a time period (we use one day as an example), the goal is to maximize the positive value (e.g., daily active) while controlling the daily negative value (e.g., notification disabled) over the T candidate push timestamps, where T can be varied for different users and methods. For a specific user at each time $t \in \{1, 2, \ldots, T\}$, unlike standard RL that optimizes a policy purely via reward maximization, recent advances in decision transformers (DT) formulate this as a conditional sequence modeling problem. Let R_t denote the return-to-go (RTG) at step t, defined as the desired future return:

$$R_t = \sum_{j=t}^{T} \left(r_j^+ - \lambda_n r_j^- \right),$$

where r^+ and r^- are positive and negative rewards respectively, and λ_n is a hyperparameter balancing the trade-off.

In previous DT-based recommendation frameworks, the model takes the current state s_t, including user feature, item feature, and environment information (such as current time and previous push time), and the target RTG R_t as input, and conditionally generates the corresponding binary action $a_t \in \{0, 1\}$:

$$a_t = \pi(s_t, R_t).$$

The first type of previous multi-stage methods pre-calculate T offline before the day based on uplift modeling and integer optimization, and send a push at time $t \in \{1, 2, \ldots, T\}$, which does not consider the timely environment information and may find local optima. The second type methods periodically activate the system based on a pre-defined short time interval, which consumes resources and involves many low-value push timings.

To bridge this gap, we formulate push recommendation as a sequential generate-and-decide problem, in which the system sequentially generates the next push time and decides whether to send a push based on the timely environment information, discarding the pre-defined T. Specifically, if there is a push at time point t, the planning agent will output $\Delta_t > 0$ based on s_t, which defines the time gap from now. Then the system will be activated at $t + \Delta_t$. Next, the system executes a binary action $a_{t+\Delta_t} \in \{0, 1\}$ based on $s_{t+\Delta_t}$ and RTG $R_{t+\Delta_t}$ to determine whether to send the push. Finally, the planning agent outputs a next push time, which defines the time gap between now $(t + \Delta_t)$ and the next push. Formally, under our formulation, we need to determine the action tuple (a, Δ) simultaneously at time t in an end-to-end manner:

$$(\Delta_t, a_t) = (\pi_P(s_t, R_t), \pi_A(s_t, R_t)),$$

where π_P and π_A are the planning agent and action-execution agent, respectively. The overall structure of our framework is shown in Figure 2. To ensure an end-to-end learning, we share the bottom user, environment, and item embedding parameters for planning and action tasks.

3.2 Planning Agent

The planning agent acts as the proactive engine of our agentic framework. As illustrated in Figure 3, the DT model structure requires both the state s_t and a target RTG R_t as inputs, and the generated Δ_t as outputs. Note that the item side information is optional, since it depends on whether we do a item sorting. To effectively achieve this generative capability in a highly dynamic push scenario, we introduce two core methodologies:

Gated-RTG Mechanism for Condition Injection. The standard architectures that simply concatenate RTG with dense, high-dimensional state representations, will cause the RTG to be easily overwhelmed by the dominant state features, which makes the model difficult to learn stable correlations. To address this, we propose a Gated-RTG structure. Instead of naive concatenation, the RTG R_t is first passed through a Multi-Layer Perceptron (MLP) tower to upscale its dimensionality. Then, the RTG is injected into

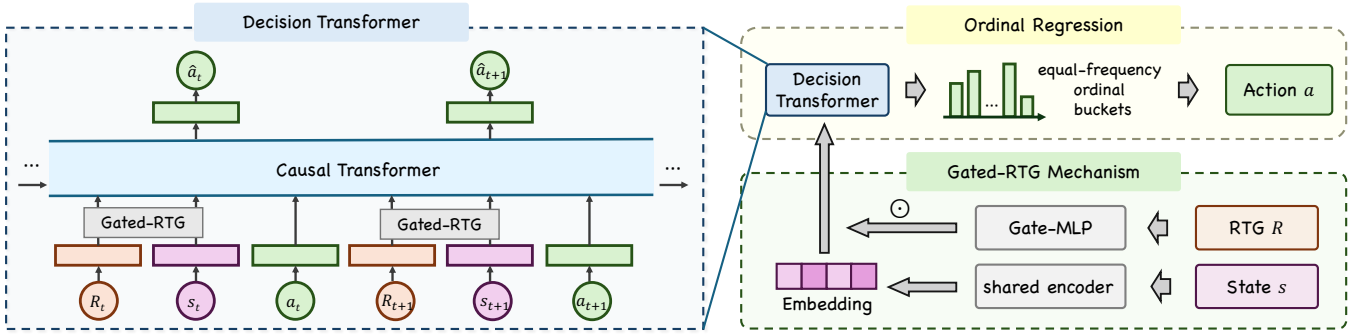


Figure 3: Architecture of planning agent.

the state via an element-wise dot product (gating mechanism):

$$e_{gated} = e_s \odot \text{MLP}(R_t),$$

where e_s is the state embedding. This gating mechanism ensures that the RTG is not overwhelmed by state features. The gated representation, e_{gated}, is then fed into the decoder to generate the push time.

Ordinal Regression for Time Generation. Predicting an exact continuous time gap Δ_t (ranging from 0 up to 24 hours) via standard regression is unstable, since the label distribution is non-Gaussian, heavily long-tailed, and the tolerance to errors is non-linear. For example, the difference between 10 minutes and 40 minutes is far more critical than that between 10 hours and 10.5 hours. To overcome this, we reformulate the time generation step as an ordinal regression problem [5], since there is an overall monotonic relationship between action and reward. For example, as the push gap increases, the active rate decreases. Specifically, we divide the continuous action space into $K = 100$ equal-frequency buckets with boundaries $\{\tau_k\}_{k=1}^{K}$. The decoder outputs ordinal logits z_k, where each bucket represents the probability that the next push time Δ_t is strictly greater than the boundary τ_k. Therefore, for a specific user, the label in k-th bucket is denoted as $y_k = \mathbf{1}[\Delta_t > \tau_k]$, the model is optimized using a Binary Cross-Entropy (BCE) loss across all buckets:

$$\mathcal{L}_{next} = \sum_{k=1}^{K} BCE(y_k, z_k).$$

Online Inference. During online inference, we decode the ordinal logits into a precise continuous time gap by smoothly aggregating the probabilities:

$$\Delta_t = \sum_{k=1}^{K} \frac{\tau_k + \tau_{k-1}}{2} \cdot \text{Sigmoid}(z_k).$$

In addition, we can adjust the RTG to achieve different desired business values. We expect the predicted Δ_t to work well under different input RTG. Therefore, we need to make $\Delta_t = \pi_P(s_t, R_t)$ capture the effect of λ_n in R_t. To achieve this, instead of fixing λ_n, we sample λ_n from a uniform distribution $U(0, 1)$ in the training stage. As a result, we can adjust the λ_n to get the desired business effect when inferring online.

3.3 Action-Execution Agent

The action-execution agent is responsible for determining whether to send a push at the time generated by the planning agent. Formally, based on s_t, inspired by GAVE [8], we propose a Generative Value-Guided Decision architecture, with the same gated RTG and λ_n sampling to ensure the online performance tuning ability.

Generative Value-Guided Decision. Specifically, we need to estimate an action-value function $Q(s_t, a_t|\lambda_n)$ to learn R_{t+1} and an action generation policy $\pi(a_t|s_t, R_t)$. Unlike traditional GAVE, including action sampling in continuous space, we exclude this in our binary push scenario. In addition, the original GAVE paper directly regress $Q(s_t, a_t|\lambda_n)$ on observed RTG R_t as the loss function. However, since the actions in training data may be suboptimal, the observed RTG fails to represent the future reward under the given state s_t [28]. Therefore, directly regressing on the observed RTG cannot yield superior performance. Therefore, we propose using the Bellman equation to learn $Q(s_t, a_t|\lambda_n)$. Taking r_t^+ as an example, the meaning of Q value is the active probability. Therefore, we define the following reward:

$$r_t^+ = \begin{cases} 0, & t < T \\ 0, & t = T, \text{and user remains inactive finally} \\ 1, & t = T, \text{and user becomes active finally} \end{cases}$$

Therefore, the positive RTG is

$$R_t^+ = \begin{cases} 1, & \text{if the user is already active} \\ 1, & \text{if } a_t = 1 \text{ \& user clicks push} \\ r_t^+ + \gamma \max_a Q^+(s_t, a|\lambda_n), & \text{via the Bellman equation} \end{cases}$$

The same technique of defining the RTG can also hold for R_t^-. Then we learn $Q(s_t, a_t|\lambda_n)$ using MSE loss.

$$\mathcal{L}_Q = \sum_{t=1}^{T-1} (R_{t+1} - Q(s_t, a_t|\lambda_n))^2,$$

where $R_t = R_t^+ - \lambda_n R_t^-$ and $Q(s_t, a_t|\lambda_n) = Q^+(s_t, a_t) - \lambda_n Q^-(s_t, a_t|\lambda_n)$. Our goal is to learn a better action, which will correspond to a larger $Q(s_t, \lambda_n)$. Therefore, following GAVE [8], we calculate the weight $w_t = \text{Sigmoid}(\alpha_r \cdot (Q(s_t, a_t|\lambda_n) - Q(s_t, 1 - a_t|\lambda_n)))$ to reweight the action loss below, where α_r is a hyperparameter:

$$\mathcal{L}_\pi = -\sum_{t=1}^{T} \left[w_t' \log \pi(a_t|s_t, R_t) + (1 - w_t') \log(1 - \pi(a_t|s_t, R_t)) \right],$$

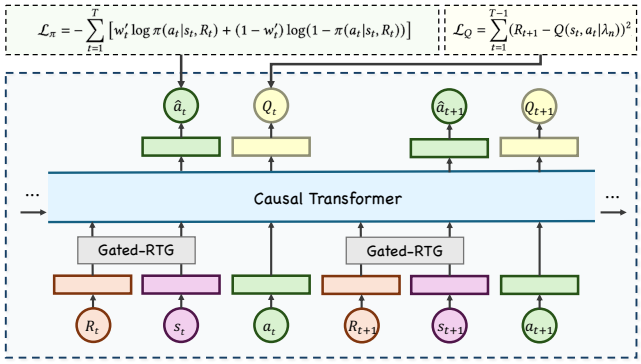


Figure 4: Architecture of the action-execution agent.

where w'_t represents the weight with gradients frozen. The intuition is, we add more weight to ensure the action-execution agent can learn actions with larger Q value.

To summarize, the final loss function for model training is

$$\mathcal{L} = \mathcal{L}_{next} + \lambda_1 \mathcal{L}_Q + \lambda_2 \mathcal{L}_\pi.$$

Discussion of Model Structure. In this subsection, we discuss the model structure difference between the planning and action-execution agent. First, since predicting the exact next delivery time is a high-noise task with a continuous action space, we use the traditional decision transformer structure based on imitation learning to ensure a lower bound of performance. Specifically, this high level of noise arises because the planning agent sits upstream in the push system. The effect of each single decision can be diluted by multiple factors. In addition, the push time decisions do not directly determine delivery, leading to a relatively weak connection between RTG and generated push time. Meanwhile, we find that introducing a critic can produce a higher performance upper bound empirically, but it is more difficult to tune parameters and may easily suffer from reward hacking. Therefore, we use this structure in the binary action-execution scenario with less noise.

3.4 System Landing Challenge

First, given the large volume of business events we have, such as live streams and video posts, relying solely on the planning agent carries the risk of missing critical events. For example, if a star followed by a user goes live, this is a potential push time. Therefore, in our implementation, we integrate such business events with self-generated push time to better enhance user experience.

Second, our platform handles millions of queries per second (QPS). Since our system does not rely on fixed budget limits and predefined delivery time periods, there is a high risk of unreasonable planning, such as too short or too long. In addition, if the sending is too frequent, it will consume too many computing resources. Therefore, we deploy a lightweight filtering agent to ensure the safety and efficiency of the entire agentic push system.

To avoid extreme user experiences, we introduce two primary boundary controls:

- System-side control: This rule suppresses the activation of the downstream system for push time within a fixed window after the most recent successful send.
- User-side control: This rule suppresses the activation for push time within a fixed window after the user becomes active.

In practice, these boundary controls are set to highly conservative, hardcoded values on the scale of minutes rather than hours simply to prevent extreme user experiences. Beyond these basic safeguards, the primary filtering decision relies on a lightweight model based solely on user and environment information. Precisely, we distill the knowledge from the heavy action execution agent into a 3-layer MLP without item-side features. This exclusion is crucial because incorporating item-side features would require invoking the computationally expensive downstream item sorting module. By distilling the model decisions and dropping these item-side features, we find a slight degradation in model metrics, but a massive improvement in resource efficiency.

Computational Resource Efficiency. In industrial push recommendation pipelines, the downstream action-execution stage is computationally heavy, since it requires complex item recall, pre-ranking, and ranking using massive item-side features. In our scenario, including item sorting consumes 10 times more resources than only using user features and environment. The proposed filtering agent can also prevent the system from wasting valuable compute on low-potential push timing. We will discuss the specific resource savings in the experimental section.

4 Experiment

To comprehensively evaluate the proposed method, we conduct experiments on the Douyin offline and online datasets. The offline training and evaluation are conducted on a massive dataset comprising over six months of logs from more than one billion users, ensuring the model's robustness and generalizability across diverse user segments. All online experiments are conducted as A/B tests on the Douyin push notification platform.

4.1 Baseline and Experiment Protocol

All experiments are conducted as online A/B tests on the **Douyin** push notification platform. We report relative improvements or reductions. We measure the business value on three dimensions: **User Active Days (UAD)** (the higher the better), **Negative Experience (NE)** (such as notification disabled, the lower the better), and **Resource Consumption** (computational cost, the lower the better). We use a widely deployed multi-stage approach as our baseline. Specifically, before the day begins, the system relies on user-level uplift models to predict the benefit of push times and frequencies. It then applies integer programming solvers to allocate optimal delivery time and frequencies for each user under a total budget limitation.

4.2 Overall Performance & Ablation Study

To quantify the overall and respective contribution of the three core components (**planning**, **action-execution**, and **Filtering**), we report results in Table 1. The results indicate that all three agents effectively and synergistically drive the system's performance, collectively achieving a 0.2843% increase in UAD, 1.9089% reduction in Negative Experience (NE), and 79.42% reduction in computational resource. Among the three agents, the planning agent serves as the

Table 1: Performance results of each agent.

Agent	UAD↑	NE↓	Resource
Planning	+ 0.1808%	- 0.9781%	- 4.539%
Action-Execution	+ 0.1035%	- 0.6843%	-
Filtering	-	- 0.2465%	-74.88%
All	+ 0.2843%	- 1.9089%	-79.42%

Table 2: Relative change of pass rate after applying the planning agent.

Filtering Pass Rate	Action-Execution Pass Rate
+12.3%	+2.3%

Table 3: Relative change of inter-push gap.

Inter-push Gap	0~20min	20min~1h	1h~3h	3h~6h
Change	-35.93%	+20.37%	+91.93%	+179.84%
Quantile	**0.01**	**0.05**	**0.25**	**0.5**
Change	+0.1%	+13.1%	+20.1%	+25.6%

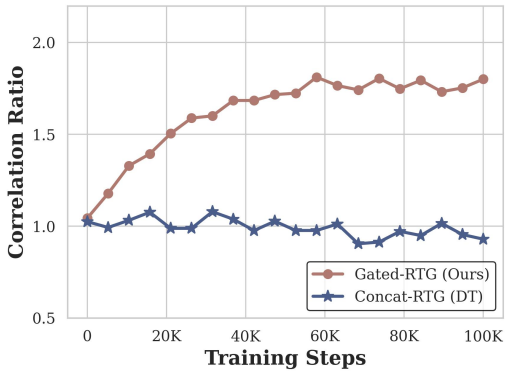


Figure 5: Effectiveness of learning correlation between RTG and action.

dominant contributor to both the positive and negative metrics. It accounts for the largest share of the overall positive gain (+0.1808% UAD) and over half of the total negative-metric reduction (-0.9781% NE). This demonstrates that better planning not only ensures valuable opportunities are properly captured but also proactively avoid unreasonable sending times that cause user disturbance.

4.3 Further Analysis on Planning

Downstream Pass Rate. Ideally, an optimal generated push time should successfully pass through all downstream agents to trigger a delivery, thereby maximizing both business value and computational resource utilization. As reported in Table 2, the pass rate of Filtering and Action-Execution improved significantly. This is mainly because our RTG is collected end-to-end from the Planning Agent's decision through Filtering, Action-Execution, and the final user response. Consequently, any block contributes zero to the RTG, making the Filtering outcome an implicit supervisory signal embedded in the reward. Training the Planning Agent on this RTG therefore biases its outputs toward time gaps that survive downstream filtering. The same compute resources now sustain a higher volume of high-quality requests. We identify this as the primary driver behind the observed gain in user active days.

Sending Density. Table 3 reports the inter-push gap distribution. The Planning Agent substantially suppresses the dense-push tail—the fraction of short-interval pushes drops sharply, while that of longer-interval pushes rises—and consequently shifts the 5th–50th percentiles of the gap distribution toward larger values. By reducing dense pushes, the Planning Agent contributes to mitigating user negative experience.

Gated RTG Structure. In our industrial scenario, the observed RTG signal is inherently highly noisy, making it extremely difficult for the model to learn a stable relationship between the RTG condition and the generated action. To quantitatively evaluate this, we define a tracking metric:

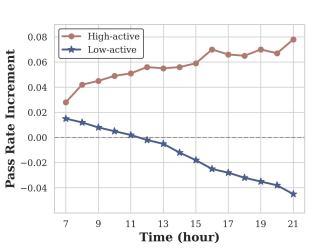
$$\text{Correlation Ratio} = \frac{\mathbb{E}_x\left[\hat{\Delta}_t(\text{RTG} = r_{\text{low}})\right]}{\mathbb{E}_x\left[\hat{\Delta}_t(\text{RTG} = r_{\text{high}})\right]}, \tag{1}$$

with $r_{\text{low}} = 0.0$ and $r_{\text{high}} = 1.0$. A ratio near 1.0 indicates that the RTG condition has no effect on the generated action, whereas a ratio significantly deviating from 1.0 indicates that the model has learned a stable correlation.

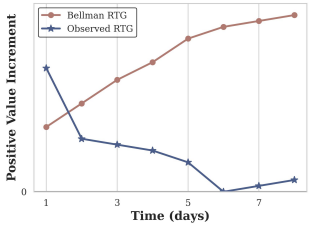
As shown in Figure 5, the Decision-Transformer (Concat-RTG) fails to capture this correlation, with the correlation ratio fluctuating randomly around 1.0 throughout the entire training process. This is primarily because the high noise in our environment introduces severe variance into the RTG signal. When directly concatenated, the neural network tends to bypass this unreliable condition, causing the fragile RTG to be completely overwhelmed by the rich, deterministic, high-dimensional state features. In contrast, our proposed Gated-RTG mechanism forces the state embeddings to be gated by the RTG, successfully learning the correlation with its ratio steadily converging to approximately 1.8—a value much closer to the true empirical statistics of our training samples. This strongly suggests that the Gated-RTG design effectively overcomes the noise and feature-overwhelming problems, successfully capturing the underlying data distribution.

4.4 Further Analysis on Action-Execution

Increment of Pass Rate. To further demonstrate that the action-execution agent captures real-time dynamics, we analyze the increment of the pass rate across different hours. As shown in Figure 6a, for currently inactive high-active users, the pass rate increment steadily increases and remains positive, peaking in the evening. This indicates that if a high-active user remains inactive by night, sending a push yields substantial positive value increase. Conversely, for currently inactive low-active users, the pass rate increment drops below zero into negative territory in the evening. This stark contrast reflects the model's trajectory awareness: if a low-active



(a) Hourly action-execution agent pass rate.

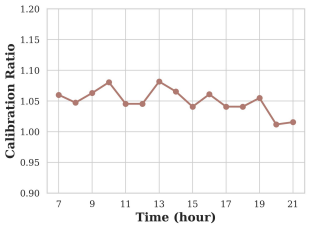


(b) Comparison of positive value increase between Bellman RTG and observed RTG.

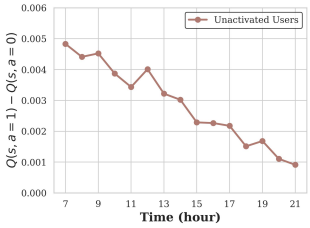
Figure 6: Analysis across push time and RTG calculation.

Table 4: User-level sending distribution.

Metric	Relative Change (%)
Zero sends (0)	−4.06%
Moderate sends (1∼20)	+20.31%
Frequent sends (21∼30)	−6.87%
Extreme sends (>30)	−9.30%



(a) Calibration ratio of predicted Q values.



(b) Action value difference for unactivated users.

Figure 7: Analysis of the learned Q value.

user remains unengaged by night, they are highly unlikely to become active, meaning the positive value of a push is minimal. These phenomena show the push action rationality of the agent.

Long-term Value of Bellman RTG Modeling. To validate whether the action-execution agent effectively captures the cumulative effects of push notifications, we monitor the long-term positive value increment on online A/B testing. As illustrated in Figure 6b, the traditional methods that directly learn $Q(s, a)$ on observed RTG experiences an initial spike but rapidly degrades, with its positive value increment converging around zero over time. One possible reason is that the observed RTG may be suboptimal due to the wrong action in training data. In contrast, our proposed Bellman RTG modeling exhibits a continuous and steady rise in positive value increment, stabilizing consistently above the baseline.

Alleviation of Extreme Sending Behaviors. In addition to optimizing long-term retention, trajectory-level modeling can also significantly improve the immediate user experience by naturally restricting irrational sending decisions. As detailed in Table 4, the distribution of daily sends per user becomes substantially more balanced under our proposed method. The system effectively curtails

Table 5: Relative computational resources for different agents. We let filtering agent as the unit, and present the relative computational overhead of other agents.

	Resource	
Agent	Overall	Per 1k request
Filtering	1.00	1.00
Planning	1.27	1.27
Action-Execution	9.62	57.72

Table 6: Request drop compared with no filter and 20% random filter. We present overall request and the results under different activity levels.

	Activity Level		
Compared Scheme	**Inactive**	**High-Active**	**Fully-Active**
No Filter	-78.86%	-66.63%	-85.65%
Random Filter	-5.600%	-5.131%	-63.70%

extreme over-sending, reducing the number of users receiving more than 20 and 30 pushes per day by 6.87% and 9.30%, respectively. Simultaneously, the volume of zero-send cases (users receiving no pushes at all) decreases by 4.06%. This confirms that modeling trajectory-level rewards penalizes sending too frequently or too infrequently and smoothly shifting the overall population towards a more reasonable and moderate daily push volume (1∼20 sends).

Correctness of the Learned Q Value. Taking Q^+ as an example, we conduct further analysis to verify the accuracy of Q value in the action-execution agent, as shown in Figure 7a and Figure 7b. First, the ratio of predicted Q value to actual ground truth RTG remains highly stable between 1.0 and 1.1 across all hours of the day, demonstrating accurate value predictions. Second, we investigate the marginal value of sending a push by analyzing $Q^+(s, a = 1) - Q^+(s, a = 0)$. For unactivated users, this difference exhibits a monotonicity trend, matching the business intuition that users are much easier to become active earlier in the day, which also consistent with results in Figure 6a.

4.5 Further Analysis on Filtering

Necessity of Early-Stage Filtering. We first compare the resource consumption across different modules to demonstrate the necessity of early-stage filtering for real-world online deployment. As shown in Table 5, although the filtering and planning agents share a comparable per-request overhead, the action-execution agent is computationally far more expensive. Specifically, it consumes over 50 times the computing resources of the filtering stage per 1k requests, primarily due to the heavy value estimation required for candidate item sorting. Consequently, effectively reducing the volume of requests that reach this final execution stage translates into substantial computational savings, ensuring that the end-to-end agentic pipeline remains highly scalable and viable under massive online traffic.

Value-Aware Resource Reduction. Table 6 further demonstrates that the filtering agent provides substantial resource reduction by aggressively pruning low-value requests before they enter the more expensive downstream agents. Compared with the no-filter setting, the proposed filtering agent drops 74.88% of requests overall, with particularly large reductions for inactive and fully-active users. This is consistent with the design motivation: inactive users often have limited marginal response value. Conversely, highly active users naturally trigger an abundance of business events (such as followed creators going live or friends posting new videos), generating numerous potential push times. However, the marginal increment to UAD from these additional pushes is relatively low for such already-engaged users. Consequently, we can observe that the filter module predominantly chooses to filter out these redundant requests. Compared with a random filter at the same send level, our filtering agent removes more requests overall and especially avoids wasting computation on fully-active users, showing that the filtering decision is value-aware rather than merely budget-driven.

These results further reveal that the filtering stage plays two distinct roles. (i) By discarding a portion of the candidate push times before the downstream stages, the filter directly reduces the computational cost of the subsequent pipeline. (ii) By filtering out low-value push time, the filter improves the overall quality of the push passed to downstream agents, which contributes to the positive metric.

5 Conclusion

In this paper, we addressed the highly non-trivial "whether and when" delivery problem in push recommendation systems. While traditional multi-stage frameworks evaluate delivery decisions at pre-planned time slices, their performance is hindered by local optima and a lack of sensitivity to real-time user state feedback. To bridge this gap, we proposed a proactive, self-triggered end-to-end agentic push recommendation framework that allows the system to generate and consume signals in a closed loop, discarding predefined budgets and times. Our proposed system first utilizes a decision transformer-based planning agent with a gated ordinal regression method to dynamically determine when to schedule the next push. Then, a generative value-guided action-execution agent evaluates whether to send the push based on a trajectory reward. In addition, we introduced a lightweight filtering agent to both control the computational overhead and act as a crucial safeguard against unreasonable planning behaviors. Extensive online A/B testing on the Douyin platform demonstrates that this approach significantly improves user re-engagement while effectively reducing user push permission disabled rate. One potential limitation is that the signal filtering agent currently relies on fixed thresholds trained offline, which might limit its real-time adaptability to changes in the system. It is promising to investigate fully dynamic, end-to-end filtering mechanisms in future work.

References

[1] Sami Abboud, Eleanor Hanna, Olivier Jeunen, Vineesha Raheja, and Schaun Wheeler. 2025. Agentic personalisation of cross-channel marketing experiences. In *Proceedings of the Nineteenth ACM Conference on Recommender Systems*. 907–910.

[2] Michal Aharon, Yohay Kaplan, Rina Levy, Oren Somekh, Ayelet Blanc, Neetai Eshel, Avi Shahar, Assaf Singer, and Alex Zlotnik. 2019. Soft frequency capping for improved ad click prediction in yahoo gemini native. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2793–2801.

[3] Renee Barnes, Rory Mulcahy, and Aimee Riedel. 2025. Push notifications and news snacking: the impact of mobile news alert framing on reader engagement. *new media & society* 27, 3 (2025), 1486–1506.

[4] Stephen Bonner and Flavian Vasile. 2018. Causal embeddings for recommendation. In *Proceedings of the 12th ACM conference on recommender systems*. 104–112.

[5] Paul-Christian Bürkner and Matti Vuorre. 2019. Ordinal regression models in psychology: A tutorial. *Advances in Methods and Practices in Psychological Science* 2, 1 (2019), 77–101.

[6] Jiaju Chen, Wang Wenjie, Chongming Gao, Peng Wu, Jianxiong Wei, and Qingsong Hua. 2024. Treatment effect estimation for user interest exploration on recommender systems. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1861–1871.

[7] 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.

[8] Jingtong Gao, Yewen Li, Shuai Mao, Peng Jiang, Nan Jiang, Yejing Wang, Qingpeng Cai, Fei Pan, Peng Jiang, Kun Gai, et al. 2025. Generative auto-bidding with value-guided explorations. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 244–254.

[9] Rupesh Gupta, Guanfeng Liang, Hsiao-Ping Tseng, Ravi Kiran Holur Vijay, Xiaoyu Chen, and Rómer Rosales. 2016. Email volume optimization at LinkedIn. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 97–106.

[10] Eugene Ie, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Morgane Lustman, Vince Gatto, Paul Covington, et al. 2019. Reinforcement learning for slate-based recommender systems: A tractable decomposition and practical methodology. *arXiv preprint arXiv:1905.12767* (2019).

[11] Michael Janner, Qiyang Li, and Sergey Levine. 2021. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems* 34 (2021), 1273–1286.

[12] Huxiao Ji, Haitao Yang, Linchuan Li, Shunyu Zhang, Cunyi Zhang, Xuanping Li, and Wenwu Ou. 2024. TIM: Temporal Interaction Model in Notification System. In *Proceedings of the 2024 International Conference on Multimedia Retrieval*. 1120–1124.

[13] Kathy Lee, Ashequl Qadir, Vivek V Datla, Sadid A Hasan, Joey Liu, Aaditya Prakash, and Oladimeji Farri. 2016. Assorted Textual Features and Dynamic Push Strategies for Real-time Tweet Notification.. In *TREC*.

[14] Babak Loni, Anne Schuth, Lucas van de Haas, Jeroen Jansze, Vasco Visser, and Marlies Van der Wees. 2019. Personalized push notifications for news recommendation. In *2nd Workshop on Online Recommender Systems and User Modeling*. PMLR, 36–45.

[15] Bayram Berkay Mumcu and Ayça Çebi. 2025. You have a notification: the role of push notifications in shaping students' engagement, self-regulation and academic procrastination. *International Journal of Educational Technology in Higher Education* 22, 1 (2025), 36.

[16] Borja Ocejo, Ruofan Wang, Ke Liu, Rohit K Patra, Haotian Shen, David Liu, Yiwen Yuan, Gokulraj Mohanasundaram, Fedor Borisyuk, and Prakruthi Prabhakar. 2025. Generative Sequential Notification Optimization via Multi-Objective Decision Transformers. *arXiv preprint arXiv:2509.02458* (2025).

[17] Tadashi Okoshi, Kota Tsubouchi, and Hideyuki Tokuda. 2019. Real-world product deployment of adaptive push notification scheduling on smartphones. In *Proceedings of the 25th acm sigkdd international conference on knowledge discovery & data mining*. 2792–2800.

[18] Xuan-Lam Pham, Thi-Huyen Nguyen, Wu-Yuin Hwang, and Gwo-Dong Chen. 2016. Effects of push notifications on learner engagement in a mobile learning app. In *2016 IEEE 16th International Conference on Advanced Learning Technologies (ICALT)*. IEEE, 90–94.

[19] Martin Pielot, Amalia Vradi, and Souneil Park. 2018. Dismissed! a detailed exploration of how mobile phone users handle push notifications. In *Proceedings of the 20th international conference on human-computer interaction with mobile devices and services*. 1–11.

[20] Prakruthi Prabhakar, Yiping Yuan, Guangyu Yang, Wensheng Sun, and Ajith Muralidharan. 2022. Multi-objective optimization of notifications using offline reinforcement learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3752–3760.

[21] Yuta Saito, Suguru Yaginuma, Yuta Nishino, Hayato Sakata, and Kazuhide Nakata. 2020. Unbiased recommender learning from missing-not-at-random implicit feedback. In *Proceedings of the 13th international conference on web search and data mining*. 501–509.

[22] Natalie Jomini Stroud, Cynthia Peacock, and Alexander L Curry. 2020. The effects of mobile push notifications on news consumption and learning. In *Mobile news*. Routledge, 32–48.

[23] Konglong Tang, Yong Wang, Hao Liu, Yanxiu Sheng, Xi Wang, and Zhiqiang Wei. 2013. Design and implementation of push notification system based on the MQTT protocol. In *2013 International Conference on Information Science and Computer Applications (ISCA 2013)*. Atlantis Press, 116–119.

[24] Wenjie Wang, Fuli Feng, Xiangnan He, Xiang Wang, and Tat-Seng Chua. 2021. Deconfounded recommendation for alleviating bias amplification. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*. 1717–1725.

[25] Dawn Wheatley and Raul Ferrer-Conill. 2021. The temporal nature of mobile push notification alerts: A study of European news outlets' dissemination patterns. *Digital journalism* 9, 6 (2021), 694–714.

[26] Atilla Wohllebe. 2020. Consumer acceptance of app push notifications: systematic review on the influence of frequency. (2020).

[27] Atilla Wohllebe, Dirk Siegfried Hübner, Uwe Radtke, and Szilárd Podruzsik. 2021. Mobile apps in retail: Effect of push notification frequency on app user behavior. *Innovative Marketing* 17, 2 (2021), 102–111.

[28] Taku Yamagata, Ahmed Khalil, and Raul Santos-Rodriguez. 2023. Q-learning decision transformer: Leveraging dynamic programming for conditional sequence modelling in offline rl. In *International Conference on Machine Learning*. PMLR, 38989–39007.

[29] Kevin P Yancey and Burr Settles. 2020. A sleeping, recovering bandit algorithm for optimizing recurring notifications. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3008–3016.

[30] Yiping Yuan, Ajith Muralidharan, Preetam Nandy, Miao Cheng, and Prakruthi Prabhakar. 2022. Offline reinforcement learning for mobile notifications. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 3614–3623.

[31] Yiping Yuan, Jing Zhang, Shaunak Chatterjee, Shipeng Yu, and Romer Rosales. 2019. A state transition model for mobile notifications via survival analysis. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. 123–131.

[32] Yuguang Yue, Yuanpu Xie, Huasen Wu, Haofeng Jia, Shaodan Zhai, Wenzhe Shi, and Jonathan J Hunt. 2022. Learning to Rank For Push Notifications Using Pairwise Expected Regret. *arXiv preprint arXiv:2201.07681* (2022).

[33] Bo Zhao, Koichiro Narita, Burkay Orten, and John Egan. 2018. Notification volume control and optimization system at Pinterest. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1012–1020.

[34] Huasha Zhao, Luo Si, Xiaogang Li, and Qiong Zhang. 2017. Recommending complementary products in e-commerce push notifications with a mixture model approach. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 909–912.

[35] Lixin Zou, Long Xia, Zhuoye Ding, Jiaxing Song, Weidong Liu, and Dawei Yin. 2019. Reinforcement learning to optimize long-term user engagement in recommender systems. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2810–2818.