OneRec Technical Report

OneRec Team

Recommender systems have been widely used in various large-scale user-oriented platforms for many years. Over the past decade, recommendation technology has evolved from traditional heuristic-based rules to deep learning models, significantly improving recommendation accuracy. However, compared to the rapid changes and developments in the AI community, recommendation systems have not achieved a breakthrough in recent years. For instance, they still rely on a multi-stage cascaded architecture rather than an end-to-end approach, leading to computational fragmentation and optimization inconsistencies. Additionally, the cascading structure has hindered the effective application of key breakthrough technologies from the AI community in recommendation scenarios.

To address these issues, we propose OneRec, which reshapes the recommendation system through an end-to-end generative approach. Under this new architecture, we have achieved promising results. Firstly, we have enhanced the computational FLOPs of the current recommendation model by 10 $\times$ and have identified the scaling laws for recommendations within certain boundaries. Secondly, reinforcement learning (RL) techniques, previously difficult to apply for optimizing recommendations, show significant potential in this framework. Lastly, through infrastructure optimizations, we have achieved 23.7% and 28.8% Model FLOPs Utilization (MFU) on flagship GPUs during training and inference, respectively, aligning closely with the LLM community. This architecture significantly reduces communication and storage overhead, resulting in operating expense (OPEX) that is only 10.6% of traditional recommendation pipelines. Deployed in Kuaishou/Kuaishou Lite APP, it handles 25% of total queries per second (QPS), enhancing overall App Stay Time by 0.54% and 1.24%, respectively. Additionally, we have observed significant increases in metrics such as 7-day Lifetime (LT7), which is a crucial indicator of recommendation experience. We also provide practical lessons and insights derived from developing, optimizing, and maintaining a production-scale recommendation system with significant real-world impact.

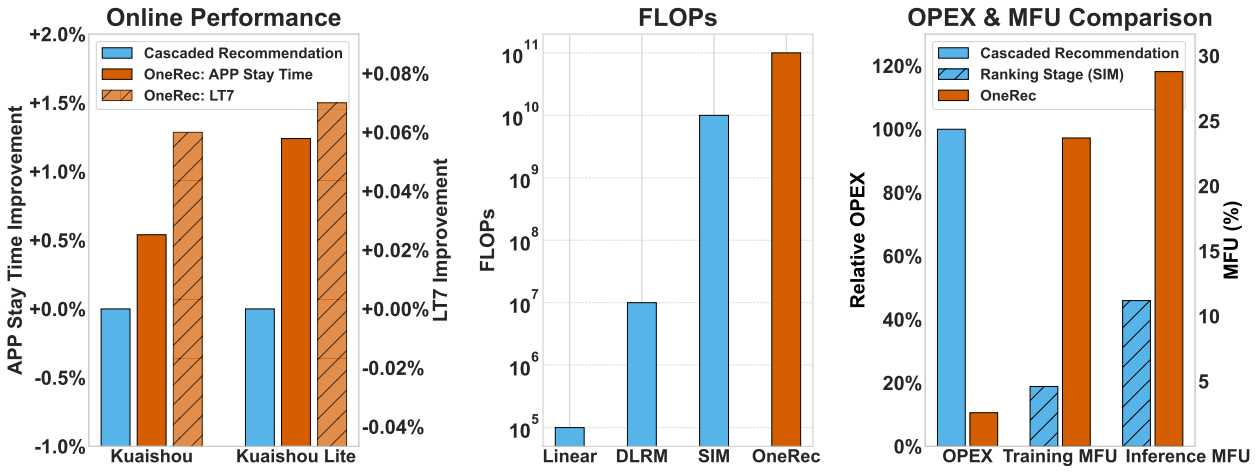

Figure 1 | Online performance, FLOPs, OPEX, and MFU comparison.

Contents

1. Introduction

With the rapid advancement of online services, recommender systems (RS) have become essential infrastructure for mitigating information overload and delivering personalized content at scale (Ricci et al., 2010). During the past decades, recommender systems have achieved several breakthrough advancements - from early Factorization Machines (Rendle, 2010) to modern deep learning architectures (Cheng et al., 2016; Guo et al., 2017; Pi et al., 2020; Zhou et al., 2018). Despite the substantial progress made by the RS research community, traditional recommendation models still rely on multi-stage cascaded architectures (see the top part of Figure 2) rather than end-to-end approaches, which face several limitations that hinder their optimal performance:

Fragmented Compute. The cascaded architecture suffers from low computational efficiency. Our comprehensive analysis of resource distribution, using Kuaishou as a case study, reveals that over **50%** of resources during serving are allocated to communication and storage rather than high-precision computation. This significant allocation to non-computational tasks highlights a fundamental inefficiency in the current architecture. Moreover, the resources dedicated to computation, particularly for the most computation-intensive ranking models, demonstrate markedly low utilization. **Specifically, the model's training and inference MFU is only 4.6% and 11.2% on flagship GPUs, respectively, which is substantially lower than the efficiency observed in large language models (LLMs), where the MFU is approximately 40% on H100** (Grattafiori et al., 2024; Shoeybi et al., 2019). This discrepancy underscores the inefficiency in resource utilization for computational tasks in recommender systems. Additionally, due to the high QPS requirements (*greater than 400k*) and low latency demands (*less than 500ms*), recommender models are often constrained to operate at a low scale and are not computation-intensive. This operational constraint further limits the potential for high-precision computation, thereby affecting the overall performance and scalability of the recommender system.

Objective Collision. What optimization objectives correspond to "good" recommendation results are not well-defined, which leads to the following conflict:

1) *Conflicts from Diverse Objectives*: Beyond common optimization goals like click-through rate and watch time, there are competing goals (hundreds of goals in Kuaishou) from users, creators, and platform ecosystems. These objectives intervene at various stages of the system, gradually undermining system consistency and increasing complexity and operational inefficiency.

2) *Cross-Stage Modeling Conflicts*: Even when modeling similar objectives, conflicts can arise due to different structures and sizes of models at various stages. For instance, the effectiveness of the retrieval stage might be constrained by the limitations of the ranking model, which, in turn, could be affected by suboptimal upstream results. This highlights the need for a more unified optimization goal and model structure across the recommendation system to ensure coherence and efficiency.

Lag Behind AI Evolution. While remarkable progress has been made in LLM and visual language model (VLM) domains (e.g., scaling laws (Henighan et al., 2020; Hoffmann et al., 2022; Kaplan et al., 2020), reinforcement learning (Ouyang et al., 2022; Rafailov et al., 2023; Shao et al., 2024; Ziegler et al., 2019)), the existing cascaded recommendation framework presents fundamental architectural barriers to adopting these proven techniques. This structural misalignment creates a widening gap between recommendation systems and mainstream AI advancements, limiting potential performance gains from state-of-the-art approaches.

To address the challenges faced by traditional cascaded recommendation architectures, we propose **OneRec** (See the bottom part of Figure 2), a novel recommendation system designed to overcome the limitations of cascade ranking systems by integrating retrieval and ranking processes into a

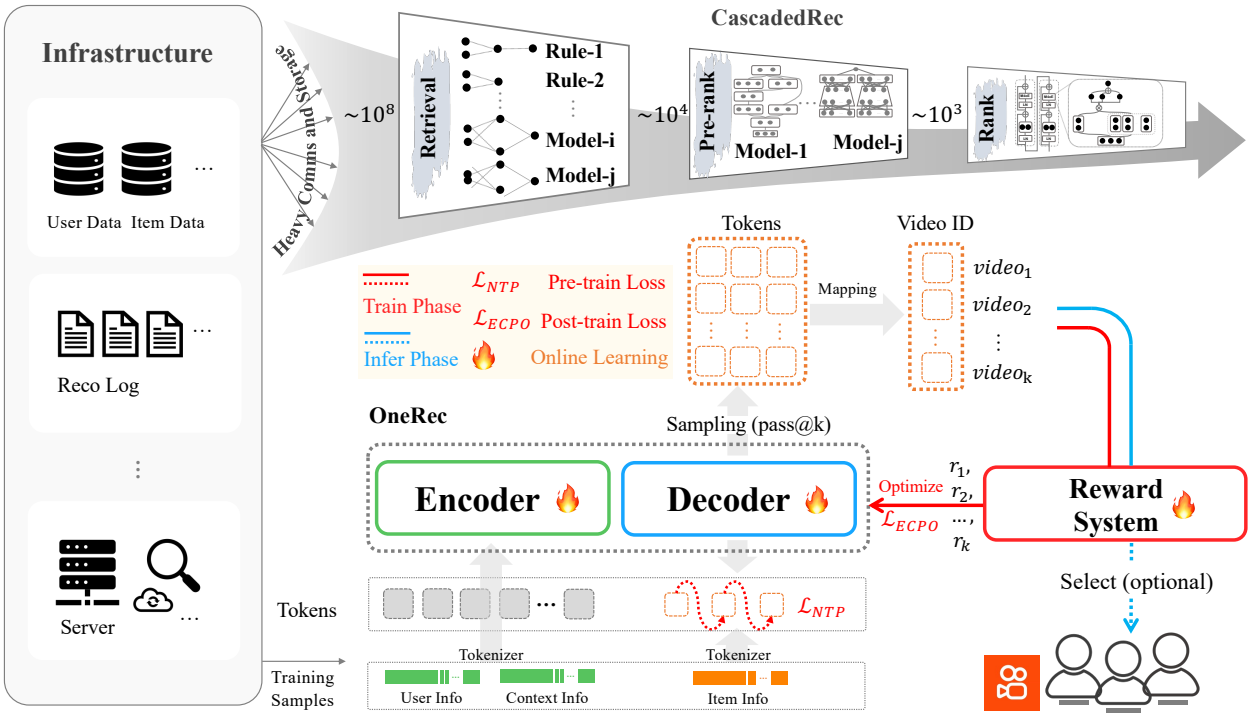

Figure 2 | Comparison between a cascaded recommender system and the OneRec. The cascaded approach system typically involves stages such as retrieval, pre-ranking, and ranking, each potentially employing multiple strategies or models. In contrast, OneRec adopts an encoder-decoder architecture to generate user-preferred videos in an end-to-end manner under the guidance of a reward model.

single-stage encoder-decoder based generative framework. This approach exhibits the following characteristics:

⊦ **End-to-End Optimization**: The system is designed to be both end-to-end and sufficiently simple to enable direct optimization for the final objective.

⊦ **Computational Efficiency**: With a focus on computational intensity, the method rigorously optimizes computational utilization efficiency during both training and inference phases, thereby fully leveraging the benefits brought by computing power advancements.

Our new framework yields several significant findings:

- Through extensive infrastructure optimizations, we have achieved **23.7% and 28.8% MFU on flagship GPUs during training and inference, respectively** — representing 5.2× and 2.6× improvements over the original ranking model — significantly narrowing the gap with the LLM community. More importantly, this end-to-end architecture dramatically reduces unnecessary communication and storage overhead, **resulting in OPEX that is merely 10.6% of that associated with traditional complex recommendation pipelines**. Currently, its deployment in the main scenarios of the Kuaishou/Kuaishou Lite APP manages approximately 25% of total QPS, **delivering improvements of 0.54% and 1.24% in App Stay Time**, while simultaneously improving all core metrics—including user engagement, video cold start, and distribution balance — demonstrating comprehensive performance gains.
- We have enhanced the computational FLOPs of the current recommendation model by 10×. Through this process, we have identified the scaling laws for recommendation systems. This discovery provides valuable insights into how recommendation system performance can be

optimized as model size and computational resources are scaled, ensuring efficient and effective deployment in various operational contexts.

- Reinforcement learning (RL) techniques, which previously had shown limited impact in traditional architectures, now demonstrate substantial potential within our framework. We have conducted extensive experiments with both offline and online performance comparisons and have developed specific application practices tailored to meet real-world industrial iteration requirements. These implementations enable the system to leverage RL, resulting in improved adaptability and performance.

In the remainder of this paper, we first elaborate on the OneRec architecture (Section 2), detailing our tokenization pipeline for short videos, the encoder's design for user interest modeling and compression, and scalable decoder optimization for precise output generation; we also introduce our reinforcement learning framework for recommendation optimization, discussing the impact of sampling space design, policy, and reward function on recommendation outcomes, along with empirical insights from production deployment. Next, we present the pre-training and post-training pipeline (Section 3), covering training data construction, hyperparameter configurations, and critical implementation discussions, followed by a description of the evaluation framework (Section 4), including offline metric systems and online performance/efficiency optimizations. Lastly, we conclude this work, discuss the existing limitations of OneRec, and propose potential directions for future research (Section 5).

2. Architecture

In this section, we present the OneRec architecture (as illustrated in the bottom part of Figure 2). The architecture first employs a tokenizer (Section 2.1) to convert videos into semantic IDs which serve as the prediction targets for the model. During the **training phase**, the encoder-decoder structure (Section 2.2 and Section 2.3) performs next token prediction to forecast target items, while simultaneously undergoing reinforcement learning alignment through the reward system (Section 2.4). In the **inference phase**, the model first generates semantic IDs and then maps these tokens back to video recommendations, with an optional reward-based selection step for further refinement.

2.1. Tokenizer

OneRec is a generative recommendation system at Kuaishou, while its billion-scale, ever-growing item space prevents generating atomic identifiers due to computational and architectural constraints. To resolve these, OneRec tokenizes items into coarse-to-fine semantic IDs using a reduced and fixed vocabulary, enabling knowledge transfer among similar items and better generalization to new items (Rajput et al., 2024). However, prior solutions (Rajput et al., 2024; Zheng et al., 2024) generate semantic IDs exclusively from context features, neglecting collaborative signals and yielding suboptimal reconstruction quality, as demonstrated in Section 4.4. Consequently, our solution integrates collaborative signals with multimodal features and then leverages RQ-Kmeans (Luo et al., 2024) to generate higher-quality hierarchical semantic IDs.

2.1.1. Aligned Collaborative-Aware Multimodal Representation

We integrate multimodal content with collaborative signals by aligning multimodal representations of collaboratively similar item pairs, as shown in Figure 3 (left). Therefore, we require the preparation of multimodal representations, item pairs, and an alignment strategy:

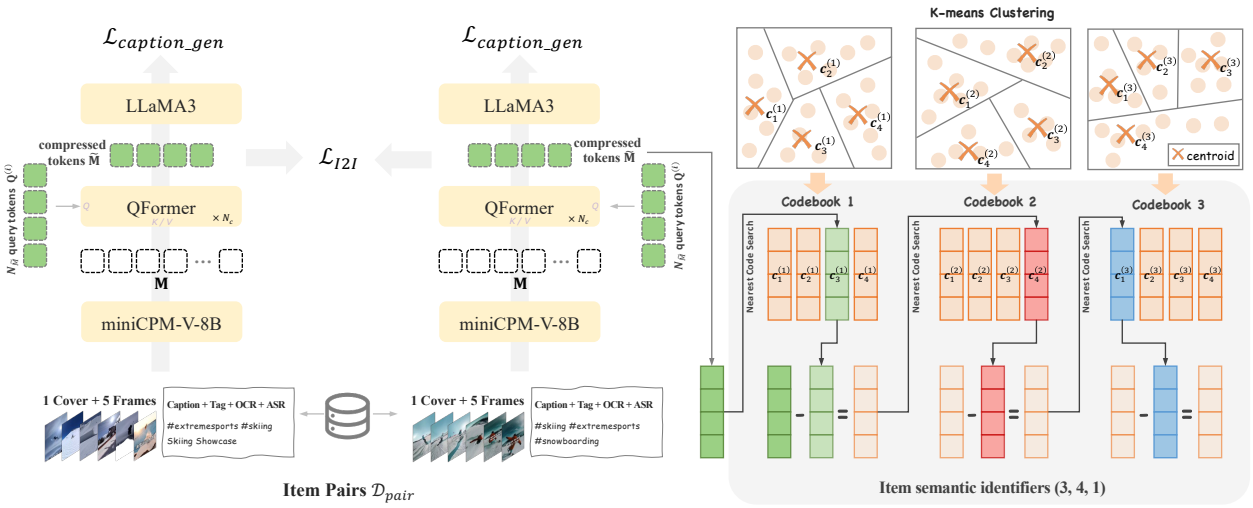

Figure 3 | Illustration of our tokenizer implementation. We first align multimodal representations of item pairs with high collaborative similarity to obtain collaborative multimodal representations, then tokenize these representations into discrete semantic IDs using RQ-Kmeans.

- **Multimodal Representations.** We incorporate multimodal inputs for each video: the caption, tag, ASR (speech-to-text), OCR (image-to-text), the cover image, and 5 uniformly sampled frames. These inputs are processed using miniCPM-V-8B (Hu et al., 2024), generating $N_M = 1280$ token vectors $\mathbf{M} \in \mathbb{R}^{N_M \times d_t}$ ($d_t = 512$). A Querying Transformer (QFormer) (Li et al., 2023) then compresses these tokens with $N_{\tilde{M}} = 4$ learnable query tokens $\mathbf{Q}^{(1)} \in \mathbb{R}^{N_{\tilde{M}} \times d_t}$:

$$\mathbf{Q}^{(i+1)} = \text{CrossAttn}\left(\mathbf{Q}^{(i)}, \mathbf{M}, \mathbf{M}\right), \tag{1}$$

$$\mathbf{Q}^{(i+1)} = \text{FFN}(\text{RMSNorm}(\mathbf{Q}^{(i+1)})), \quad \text{for } i \in \{1, 2, \ldots, N_c\}, \tag{2}$$

  where $\tilde{\mathbf{M}} = \mathbf{Q}^{(N_c+1)} \in \mathbb{R}^{N_{\tilde{M}} \times d_t}$ denotes the compressed version of $\mathbf{M}$, and $N_c = 4$ denotes the number of QFormer layers.
- **Item Pairs.** We construct high-quality item-pair dataset $\mathcal{D}_{pair}$ via: 1) User-to-Item Retrieval: For each user, we take a positively clicked target item and pair it with the most collaboratively similar item from the user's latest historical positive clicks, and 2) Item-to-Item Retrieval: We pair items exhibiting high similarity scores (e.g., the Swing similarity) (Yang et al., 2020).
- **Item-to-Item Loss and Caption Loss.** We introduce dual training objectives: 1) An item-to-item contrastive loss aligns representations of collaboratively similar video pairs $(i, j) \in \mathcal{D}_{pair}$, capturing behavioral patterns, and 2) a caption loss prevents hallucination by performing next-token prediction on video captions using LLaMA3 (Dubey et al., 2024) as the decoder, thereby preserving content understanding capabilities.

$$\mathcal{L}_{I2I} = -\frac{1}{|\mathcal{D}_{pair}|} \sum_{(i,j) \in \mathcal{D}_{pair}} \log \frac{\exp\left(\text{sim}\left(\tilde{\mathbf{M}}_i, \tilde{\mathbf{M}}_j\right)/\tau\right)}{\sum_{(i',j') \in \mathcal{D}_{pair}} \exp\left(\text{sim}\left(\tilde{\mathbf{M}}_{i'}, \tilde{\mathbf{M}}_{j'}\right)/\tau\right)}, \tag{3}$$

$$\mathcal{L}_{caption\_gen} = -\sum_k \log P\left(t^{k+1} \mid \left[t^1, t^2, \cdots, t^k\right]\right), \tag{4}$$

  where τ denotes the temperature coefficient, $\text{sim}(\cdot, \cdot)$ denotes the similarity function, t^k denotes the k-th caption token.

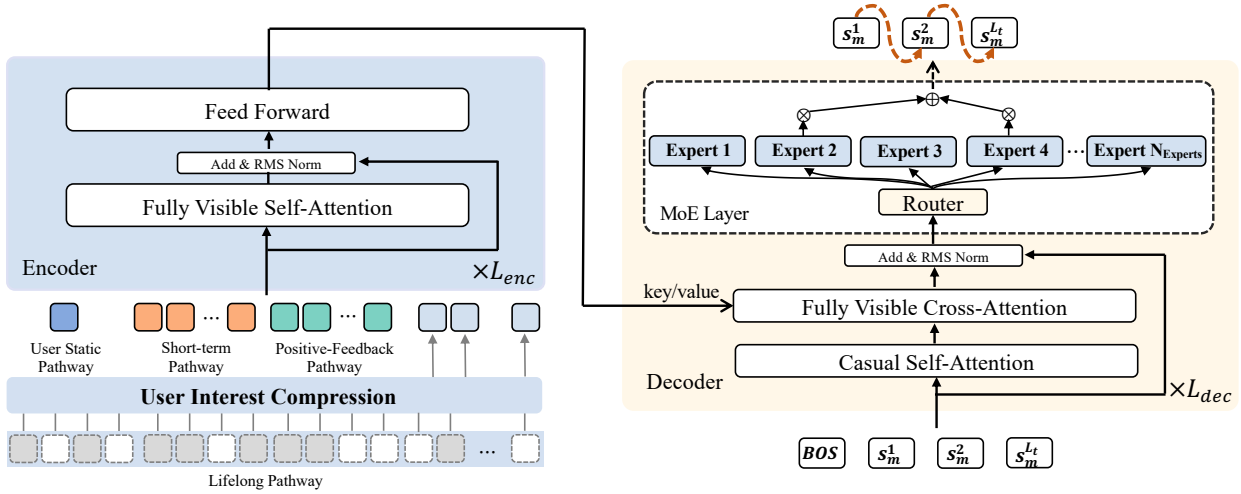

Figure 4 | Illustration of our encoder-decoder architecture.

2.1.2. Tokenization

We utilize RQ-Kmeans (Luo et al., 2024) for tokenization, which employs residual quantization to generate semantic IDs in a coarse-to-fine manner. This method constructs codebooks by applying K-means clustering directly on the residuals. An illustration of the RQ-Kmeans process is provided in Figure 3 (right).

Formally, the initial residual at layer $l = 1$ is defined as:

$$\mathcal{R}^{(1)} = \left\{\tilde{\mathbf{M}}_i \in \mathbb{R}^{N_{\tilde{M}} \times d_t} \mid \forall \text{ video } i\right\}. \tag{5}$$

For each layer l, the codebook $C^{(l)}$ is derived from K-means centroids of $\mathcal{R}^{(l)}$:

$$C^{(l)} = \text{K-means}\left(\mathcal{R}^{(l)}, N_t\right), \tag{6}$$

where $C^{(l)} = \left\{\boldsymbol{c}_k^{(l)} \in \mathbb{R}^{N_{\tilde{M}} \times d_t} \mid k = 1, \ldots, N_t\right\}$ and N_t is the codebook size. The nearest centroid index for item i is computed as:

$$s_i^l = \arg\min_k \left\|\mathcal{R}_i^{(l)} - \boldsymbol{c}_k^{(l)}\right\|, \tag{7}$$

where $\|\cdot\|$ denotes the Euclidean norm. The residual of video i for layer $l+1$ is then updated:

$$\mathcal{R}_i^{(l+1)} = \mathcal{R}_i^{(l)} - \boldsymbol{c}_{s_i^l}^{(l)}. \tag{8}$$

This quantization iterates across $L_t = 3$ layers.

As demonstrated in Section 4.4, RQ-Kmeans offers enhanced reconstruction quality, better codebook utilization, and improved balance compared to the widely used RQ-VAE (Lee et al., 2022; Rajput et al., 2024). At this stage, each video m can be represented by L_t coarse-to-fine semantic identifiers: $\{s_m^1, s_m^2, \ldots, s_m^{L_t}\}$, which will serve as the output of the OneRec recommendation system, enabling progressive item generation.

2.2. Encoder

2.2.1. Multi-Scale Feature Engineering

This section presents the feature engineering component of OneRec. We process user behavior data through four specialized embedding pathways, each designed to capture distinct scales of user

interaction patterns: **user static pathway, short-term pathway, positive-feedback pathway, and lifelong pathway**.

User Static Pathway The user static pathway generates a compact representation of core user characteristics, incorporating user identifier (`uid`), age (`age`), gender (`gender`), etc., which is then transformed into the model's hidden dimension:

$$\mathbf{f}_u = [\mathbf{e}_{\text{uid}}; \mathbf{e}_{\text{gender}}; \mathbf{e}_{\text{age}}; \cdots], \tag{9}$$

$$\mathbf{h}_u = \text{Dense}(\text{LeakyReLU}(\text{Dense}(\mathbf{f}_u))). \tag{10}$$

where $\mathbf{e}_{\text{uid}}, \mathbf{e}_{\text{gender}}, \mathbf{e}_{\text{age}} \in \mathbb{R}^{64}$ and $\mathbf{h}_u \in \mathbb{R}^{1\times d_{\text{model}}}$.

Short-term Pathway The short-term behavior pathway processes the most recent ($L_s = 20$) user interactions, incorporating video identifier (which can be represented as video identifiers `vid` or semantic identifiers `sid` as described in Section 2.1.2, we will discuss these two representation approaches in Section 4.2.2.), author identifiers (`aid`), tags (`tag`), timestamps (`ts`), playtime (`playtime`), duration (`dur`), labels (`label`, user interactions with each video, including like, follow, forward, dislike, comment, profile entry, etc.) This pathway produces representations that capture immediate user preferences and contextual factors influencing current behavior patterns:

$$\mathbf{f}_s = [\mathbf{e}^s_{\text{vid}}; \mathbf{e}^s_{\text{aid}}; \mathbf{e}^s_{\text{tag}}; \mathbf{e}^s_{\text{ts}}; \mathbf{e}^s_{\text{playtime}}; \mathbf{e}^s_{\text{dur}}; \mathbf{e}^s_{\text{label}}], \tag{11}$$

$$\mathbf{h}_s = \text{Dense}(\text{LeakyReLU}(\text{Dense}(\mathbf{f}_s))), \tag{12}$$

The feature dimensions are organized as follows: video embeddings $\mathbf{e}^s_{\text{vid}}$ match the model dimension d_{model}, author embeddings $\mathbf{e}^s_{\text{aid}}$ use 512 dimensions, while all remaining features employ 128 dimensions. All features span L_s sequence positions, yielding the final representation $\mathbf{h}_s \in \mathbb{R}^{L_s\times d_{\text{model}}}$.

Positive-feedback Pathway The positive-feedback behavior pathway operates on a sequence of high-engagement interactions ($L_p = 256$). The pathway maintains the established dimensional structure:

$$\mathbf{f}_p = [\mathbf{e}^p_{\text{vid}}; \mathbf{e}^p_{\text{aid}}; \mathbf{e}^p_{\text{tag}}; \mathbf{e}^p_{\text{ts}}; \mathbf{e}^p_{\text{playtime}}; \mathbf{e}^p_{\text{dur}}; \mathbf{e}^p_{\text{label}}], \tag{13}$$

$$\mathbf{h}_p = \text{Dense}(\text{LeakyReLU}(\text{Dense}(\mathbf{f}_p))). \tag{14}$$

All features span L_p sequence positions, yielding the final representation $\mathbf{h}_p \in \mathbb{R}^{L_p\times d_{\text{model}}}$.

Lifelong Pathway The lifelong behavior pathway is designed to process ultra-long user interaction histories with sequences of up to 100,000 videos. Directly applying attention mechanisms to such sequences is computationally prohibitive. This pathway employs a two-stage hierarchical compression strategy inspired by our previous work (Si et al., 2024).

Behavior Compression Using the multimodal content representations described in Section 2.1.1, we perform hierarchical K-means clustering on each user's interaction sequence. To balance computational efficiency and model effectiveness, we dynamically adjust the number of clusters by setting the cluster count for each step to $\lfloor \sqrt[3]{|D|} \rfloor$, where $|D|$ is the number of items in the current data. This is an empirically determined setting. The clustering process terminates when the number of items in the current cluster does not exceed a preset threshold M. Upon termination, we select the item closest to each cluster center as the representative of that cluster.

item序列使用了聚类，cate特征聚类中心，数值特征平均化

Feature Aggregation For each cluster, we construct representative features by handling discrete and continuous attributes differently. For sparse categorical features such as `vid`, `aid`, and `label`, we directly inherit the features from the representative video (i.e., the video closest to the cluster center). For continuous features such as `tag`, `ts`, `playtime`, and `duration`, we compute the average values across all videos within the cluster to capture collective behavioral patterns.

For the user's long-term historical sequence ($L_l = 2000$), each video is replaced by the features of its corresponding cluster representative:

$$\mathbf{f}_l = [\mathbf{e}^l_{\text{vid}}; \mathbf{e}^l_{\text{aid}}; \mathbf{e}^l_{\text{tag}}; \mathbf{e}^l_{\text{ts}}; \mathbf{e}^l_{\text{playtime}}; \mathbf{e}^l_{\text{dur}}; \mathbf{e}^l_{\text{label}}], \tag{15}$$

$$\mathbf{v}_l = \text{Dense}(\text{LeakyReLU}(\text{Dense}(\mathbf{f}_l))). \tag{16}$$

The final representation $\mathbf{v}_l \in \mathbb{R}^{L_l \times d_{\text{model}}}$. The lifelong pathway compresses historical sequences through QFormer, where learnable query vectors $\mathbf{h}^{(0)}_l \in \mathbb{R}^{N_q \times d_{\text{model}}}$ ($N_q = 128$) attend to the processed historical features:

$$\mathbf{h}^{(i+1)}_l = \text{CrossAttn}(\mathbf{h}^{(i)}_l, \mathbf{v}_l, \mathbf{v}_l), \tag{17}$$

$$\mathbf{h}^{(i+1)}_l = \text{FFN}(\text{RMSNorm}(\mathbf{h}^{(i+1)}_l)). \tag{18}$$

Followed by $N_l = 2$ blocks, we obtain the compressed lifelong feature representation $\mathbf{h}_l = \mathbf{h}^{(N_l)}_l \in \mathbb{R}^{N_q \times d_{\text{model}}}$.

2.2.2. Encoder Architecture

As illustrated in Figure 4, the encoder architecture of OneRec integrates multi-scale user behavior representations through a unified transformer-based framework. The encoder concatenates the outputs from the four multi-scale pathways to form a comprehensive input sequence:

$$\mathbf{z}^{(1)} = [\mathbf{h}_u; \mathbf{h}_s; \mathbf{h}_p; \mathbf{h}_l] + \mathbf{e}_{\text{pos}} \tag{19}$$

where $\mathbf{e}_{\text{pos}} \in \mathbb{R}^{(1+L_s+L_p+N_q) \times d_{\text{model}}}$ represents learnable positional embeddings. The integrated representation is processed through L_{enc} transformer encoder layers, each consisting of fully visible self-attention mechanisms followed by feed-forward networks with RMS normalization:

$$\mathbf{z}^{(i+1)} = \mathbf{z}^{(i)} + \text{SelfAttn}(\text{RMSNorm}(\mathbf{z}^{(i)})), \tag{20}$$

$$\mathbf{z}^{(i+1)} = \mathbf{z}^{(i+1)} + \text{FFN}(\text{RMSNorm}(\mathbf{z}^{(i+1)})). \tag{21}$$

The final encoder output $\mathbf{z}_{\text{enc}} = \mathbf{z}^{(L_{enc}+1)} \in \mathbb{R}^{(1+L_s+L_p+N_q) \times d_{\text{model}}}$ provides a holistic multi-scale user behavior representation, serving as the foundation for subsequent recommendation generation.

2.3. Decoder

OneRec adopts a point-wise generation paradigm during the decoding phase. For each target video m, the decoder input sequence is constructed by concatenating a learnable beginning-of-sequence token with the video's semantic identifiers:

$$\mathcal{S}_m = \left\{ s_{[\text{BOS}]}, s^1_m, s^2_m, \cdots, s^{L_t}_m \right\}, \tag{22}$$

$$\mathbf{d}^{(0)}_m = \text{Emb\_lookup}(\mathcal{S}_m). \tag{23}$$

The decoder processes this sequence through L_{dec} transformer layers. Each layer performs sequential operations:

$$\mathbf{d}_m^{(i+1)} = \mathbf{d}_m^{(i)} + \text{CausalSelfAttn}(\mathbf{d}_m^{(i)}), \tag{24}$$

$$\mathbf{d}_m^{(i+1)} = \mathbf{d}_m^{(i+1)} + \text{CrossAttn}(\mathbf{d}_m^{(i+1)}, \mathbf{Z}_{\text{enc}}, \mathbf{Z}_{\text{enc}}), \tag{25}$$

$$\mathbf{d}_m^{(i+1)} = \mathbf{d}_m^{(i+1)} + \text{MoE}(\text{RMSNorm}(\mathbf{d}_m^{(i+1)})). \tag{26}$$

Each decoder layer incorporates a Mixture of Experts (MoE) feed-forward network to enhance model capacity while maintaining computational efficiency. The MoE layer employs N_{experts} expert networks with a top-k routing strategy:

$$\text{MoE}(\mathbf{x}) = \sum_{j=1}^{k} \text{Gate}_j(\mathbf{x}) \cdot \text{Expert}_j(\mathbf{x}), \tag{27}$$

where $\text{Gate}_j(\mathbf{x})$ represents the gating weights determined by the routing mechanism, and $\text{Expert}_j(\mathbf{x})$ denotes the output of the j-th selected expert network. To ensure balanced expert utilization without introducing interference gradients, we implement a loss-free load balancing strategy following (Liu et al., 2024).

The model is trained using cross-entropy loss for next-token prediction on the semantic identifiers of target video m:

$$\mathcal{L}_{\text{NTP}} = -\sum_{j=1}^{L_t-1} \log P\left(s_m^{j+1} \mid \left[s_{[\text{BOS}]}, s_m^1, s_m^2, \cdots, s_m^j\right]\right) \tag{28}$$

2.4. Reward System

The pre-trained model only fits the distribution of the exposed item space through next token prediction, and the exposed items are obtained from the past traditional recommendation system. This results in the model being unable to break through the ceiling of traditional recommendations. To address this issue, we introduce preference alignment based on a reward system, using on-policy reinforcement learning to train the model in the generated item space. Through rewards, the model perceives more fine-grained preference information. We introduce the preference reward to align user preferences, the format reward to ensure the generation format is as legal as possible, and the specific industrial reward to align with some special industrial scenario needs.

2.4.1. User Preference Alignment

In recommendation systems, defining a "good recommendation" is much more challenging than determining the correctness of a mathematical solution. Traditional approaches (Chang et al., 2023; Wang et al., 2024) often define multiple objectives, such as clicks, likes, comments, and watch time, which are then combined into a score through a weighted fusion of the predicted values (xtr) for each objective. However, manually tuning these fusion weights is challenging, not only lacking accuracy but also lacking personalization, and often results in optimization conflicts between objectives.

To address these limitations, we propose using a neural network to learn a personalized fusion score, referred to as P-Score (Preference Score) (Cao et al., 2025). The overall framework of this model is illustrated in Figure 5 (middle). The model's underlying architecture is based on the Search-based Interest Model (SIM) (Pi et al., 2020). It includes multiple towers, each dedicated to learning specific objectives. During training, these towers compute binary cross-entropy (BCE) loss using the corresponding objective labels as auxiliary tasks. The hidden states of each tower, along with user

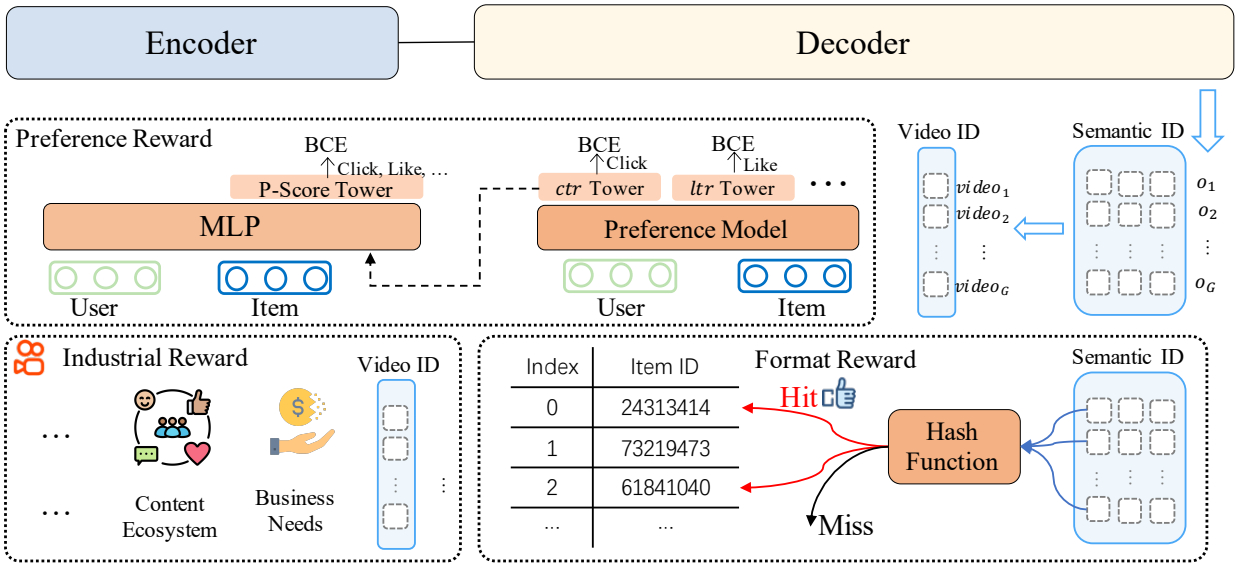

Figure 5 | Overall Framework of the Reward System. The Reward System is composed of three parts. They assign Preference Reward (P-Score), Format Reward, and specific Industrial Reward to the videos generated by the model, respectively.

and item representations, are fed into the final layer's Multi-Layer Perceptron (MLP). This MLP is followed by a single tower outputting the P-Score, which computes binary cross-entropy loss using the labels of all objectives.

This method allows the model to receive specific user information and adjust the Preference Score for that user appropriately, without compromising the experience of other users. Compared to the previous approach of indiscriminate weighted summation, this method is more likely to achieve Pareto optimization. Therefore, we use the P-Score obtained by this method as the reward for preference alignment.

Early Clipped GRPO In this section, we introduce how to use the Preference Score to align user preferences. We use ECPO (Early Clipped GRPO) for optimization. Specifically, for a user u, we generate G items using the old policy model. Each item, along with the user, is input into the Preference Reward Model to obtain the P-Score as reward r_i. The optimization objective is as follows:

$$\mathcal{J}_{ECPO}(\theta) = \mathbb{E}_{u\sim P(U),\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{old}}}\left[\frac{1}{G}\sum_{i=1}^{G}\min\left(\frac{\pi_\theta(o_i|u)}{\pi'_{\theta_{old}}(o_i|u)}A_i, \text{clip}\left(\frac{\pi_\theta(o_i|u)}{\pi'_{\theta_{old}}(o_i|u)}, 1-\epsilon, 1+\epsilon\right)A_i\right)\right], \tag{29}$$

$$A_i = \frac{r_i - \text{mean}(\{r_1, r_2, ..., r_G\})}{\text{std}(\{r_1, r_2, ..., r_G\})}, \tag{30}$$

$$\pi'_{\theta_{old}}(o_i|u) = \max\left(\frac{\text{sg}(\pi_\theta(o_i|u))}{1+\epsilon+\delta}, \pi_{\theta_{old}}(o_i|u)\right), \quad \delta > 0, \tag{31}$$

where sg represents the stop gradient operation and δ is a hyperparameter greater than 0.

We make a modification to GRPO (Group Policy Relative Optimization) (Liu et al., 2024) to make its training process more stable. The illustration is presented in Figure 6. In the original GRPO, a large policy ratio ($\pi_\theta/\pi_{\theta_{old}}$) is allowed for negative advantages, which can easily lead to gradient

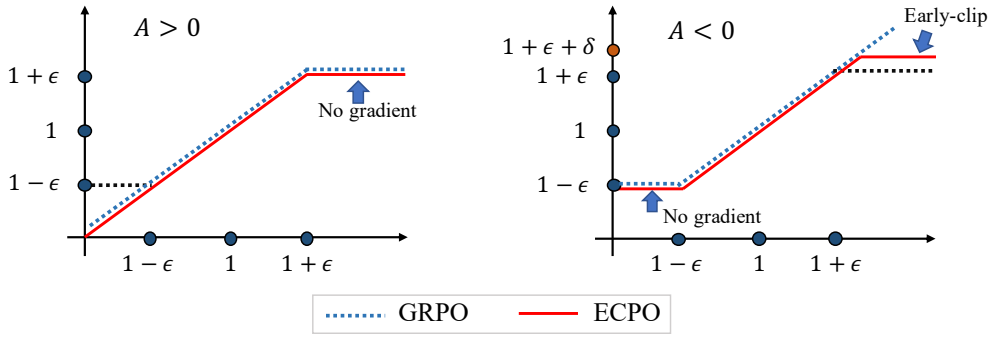

Figure 6 | Illustration of ECPO. The x-axis is $\pi_\theta/\pi_{\theta_{old}}$ and the y-axis is the clipped $\pi_\theta/\pi_{\theta_{old}}$. Items with $A > 0$ are processed in the same way as the original GRPO, while items with $A < 0$ are constrained by early-clipping to limit the maximum ratio.

explosion. Therefore, we preemptively clip policies with large ratios to ensure training stability while still allowing corresponding negative advantages to take effect. The larger the δ, the larger the tolerable policy ratio, which means the larger the tolerable gradient. This can be determined based on actual needs. In OneRec, we set δ to 0.1, which indicates that the ratio of policies with negative advantages is allowed to slightly exceed $1 + \epsilon$. We remove the KL divergence loss because the Reinforcement Learning (RL) and Supervised Fine-Tuning (SFT) are trained together in OneRec, and the SFT loss ensures the model remains stable.

2.4.2. Generation Format Regularization

In generative recommendation, the legality ratio refers to the proportion of generated semantic ID sequences that can be mapped to actual item IDs. This metric is crucial for assessing the stability of generation. In practice, the cardinality of semantic ID sequences $N_t^{L_t}$ is much larger than that of videos. This ensures that all items are covered, and a larger vocabulary introduces more parameters, leading to better performance. However, this may also result in generating semantic ID sequences without corresponding item IDs during inference, i.e., illegal generation.

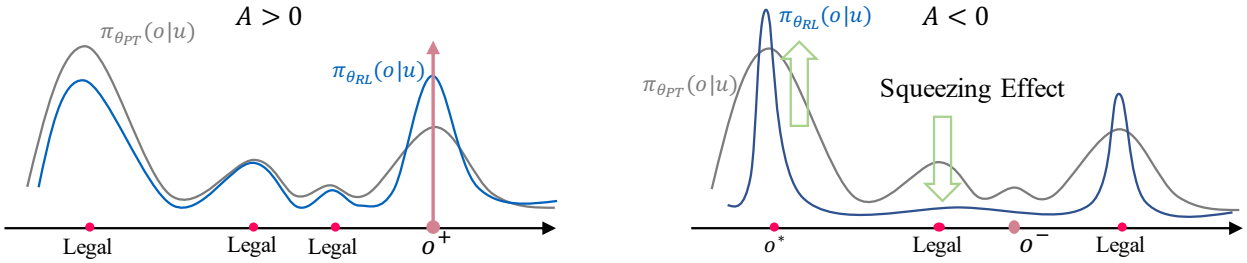

Figure 7 | Illustration of squeezing effect. $\pi_{\theta_{PT}}$ represents the pre-trained model, while $\pi_{\theta_{RL}}$ represents the model trained with ECPO. o^+ refers to videos with positive advantages, while o^- refers to those with negative advantages.

Introducing reinforcement learning with ECPO significantly increases the generation of illegal outputs. Recent work (Ren and Sutherland, 2024) suggests that this is due to the squeezing effect caused by negative advantages. As shown in Figure 7, the pre-trained model has learned to generate most of the legal tokens. After incorporating RL, items with $A > 0$ only slightly adjust the distribution. When an item with $A < 0$ is applied, the model's probability distribution compresses most of the probability mass into what it currently considers the optimal output o^*. This results in the probabilities

of some legal tokens being squeezed to levels comparable to those of illegal tokens, making it difficult for the model to distinguish legal tokens.

To address this issue, we propose incorporating a format reward in reinforcement learning to encourage the model's legal generation. Specifically, we randomly select K samples from the G samples for legality reinforcement learning. For legal samples, we set the advantage to 1, and for illegal samples, we discard them directly to avoid the squeezing effect.

$$A_i = \begin{cases} 1 & \text{if } o_i \in I_{\text{legal}} \\ 0 & \text{if } o_i \notin I_{\text{legal}} \end{cases} \tag{32}$$

The optimization objective formulation is the same as the ECPO (Equation 29) and we directly use A_i as advantages.

2.4.3. Industrial Scenario Alignment

In industrial scenarios, the recommendation system needs to consider not only user preferences but also various other aspects. For example, at Kuaishou, the ecosystem of the video community, commercialization needs, and the delivery of cold-start and long-tail videos. Traditional recommendation systems attempt to address these issues by applying algorithms or strategies at one stage of the recommendation pipeline. Due to inconsistencies across different stages, this can easily lead to a recurring cycle of unexpected problems emerging alternately. Engineers are forced to constantly make adjustments through patching, resulting in a bloated system over time that hinders iteration. In OneRec, we only need to incorporate optimization objectives into the reward system and adopt reinforcement learning to perform targeted optimization. This approach is not only convenient but also allows for end-to-end implementation, maintaining system consistency. We will provide an example of optimization practice in Section 4.3.3.

3. Training Framework

3.1. Training Infrastructure

In this section, we describe our hardware and infrastructure that facilitated the large-scale pre-training of OneRec and introduce several optimizations that enhance training efficiency.

Compute. We utilize 90 servers for training, each equipped with 8 flagship GPUs and 2 CPUs interconnected via 400Gbps NVLink to ensure high-speed intra-node bandwidth.

Networking. Intra-node communication is managed by the efficient NVLink network, while inter-node communication is supported by 400Gbps RDMA for training traffic and 100Gbps TCP for training data and embedding prefetching operations.

Storage. Each server is equipped with 4 NVMe SSDs to expedite checkpoint writes, allowing for the storage of large-scale embedding parameters and dense parameters in HDFS with minimal downtime for fault tolerance.

Training Acceleration. For training acceleration, several core optimizations are implemented:

1) Embedding Acceleration: To manage the extensive embedding workload beyond CPU capacity, we utilize Kuaishou's SKAI framework for GPU-based parameter servers. This framework leverages cross-GPU unified embedding tables, GPU caching paradigms, and prefetching pipelines to enhance training efficiency and reduce management overhead.

2) Training Parallelism: A combination of data parallelism, ZERO1 (Rajbhandari et al., 2020), and gradient accumulation is employed for model training. ZERO1 is selected because the current model's dense parameters can be loaded on a single GPU, minimizing synchronization overhead in data parallel groups when interleaving multiple macro batches.

3) Mixed Precision Training: BFloat16 is used for computations in certain MLP networks to optimize performance.

4) Compilation Optimization: For attention networks, compilation optimizations are applied to reduce computational overhead.

Thanks to advancements in highly optimized training infrastructure, the model's training MFU has improved to 23.7%, significantly narrowing the gap with the LLM training efficiency.

3.2. Pre-training

Pre-Training Data As illustrated in Section 2.2.1, our model takes multi-scale user behavior representations as input. The pre-training objective involves predicting sequences of target items for users. Each training sample comprises a target item which is tokenized into 3 semantic identifiers. This tokenization scheme results in 3 target tokens per training sample for the generative model's next-token prediction task. Our training pipeline processes approximately 18 billion samples daily, yielding a throughput of 54 billion tokens per day. The OneRec-0.935B model (detailed in Table 1) achieves convergence after training on approximately 100 billion samples, corresponding to a total exposure of 300 billion tokens during pre-training.

Key Hyperparameters The OneRec series comprises four models (two dense and two MoE variants) designed for recommendation tasks. Key architectural hyperparameters such as layer counts, hidden dimensions, and attention head numbers are detailed in Table 1. In these models, encoders and decoders have the same number of layers. For dense variants, the standard Feed-Forward Networks (FFNs) typically expand the hidden dimension d_{ff} to $2 \times d_{\text{model}}$. For the MoE variants, we replace standard FFNs with MoE layers in designated blocks, and employ SwiGLU FFNs (Shazeer, 2020; Thoppilan et al., 2022) as experts. Consistent with open-source MoE LLM settings (Fedus et al., 2022; Jiang et al., 2024), the hidden dimension for each SwiGLU expert is calculated as $\frac{2}{3} \times 4 \times d_{\text{model}}$, ensuring it is a multiple of 128.

The convergence curves for each model can be found in Section 4.2.1.

Table 1 | OneRec model architectures. "Layers" = #Encoder + #Decoder. "FFN Hid. Dim" is FFNs' intermediate size or MoEs' intermediate expert size.

Model	Layers	Hid. Dim	FFN Hid. Dim	Attn. Heads	Experts (Tot/Act)	MoE Loc.
OneRec-0.015B (Dense)	4	128	256	4	N/A	N/A
OneRec-0.121B (Dense)	8	1024	2048	8	N/A	N/A
OneRec-0.935B (MoE)	8	1024	2048	8	24 / 2	Decoder
OneRec-2.633B (MoE)	24	1024	2048	8	24 / 4	Enc & Dec

3.3. Post-training

In the post-training phase, we perform online training using real-time data streams. We simultaneously perform Reject Sampling Fine-Tuning (RSFT) and Reinforcement Learning (RL). For RSFT, we filter out the bottom 50% of exposure sessions based on play duration. The training loss is the same as the

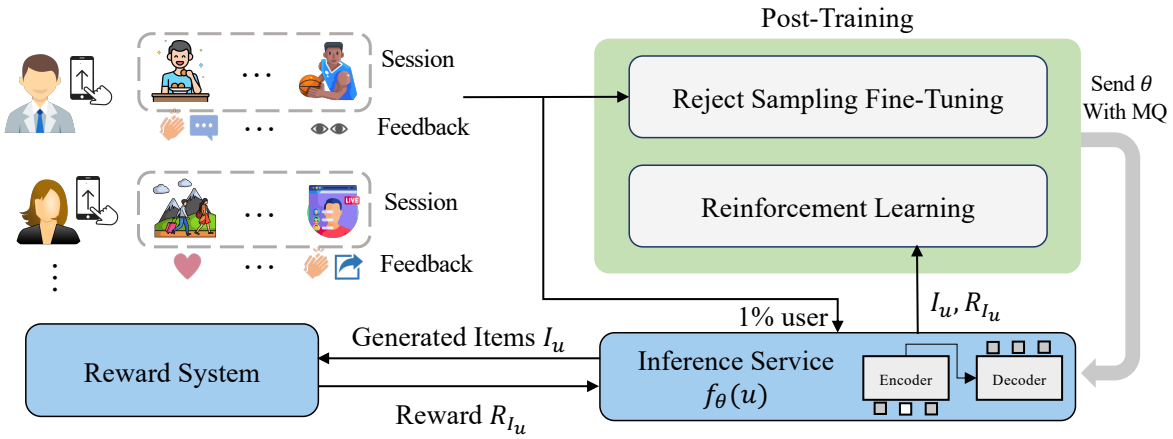

Figure 8 | The overall process of OneRec's post-training, including continual pre-training and reinforcement learning.

$\mathcal{L}_{\text{NTP}}$ loss in the pre-training process, but we apply annealing by reducing the learning rate of sparse parameters to 1×10^{-4} and dense parameters to 8×10^{-5}. For RL, we randomly select 1% of users from the RSFT data to generate RL samples.

To maximize computational resource utilization, we decouple the generation of RL samples from the training process by using an external inference service. During training, 1% of users access the external service to generate 512 items, request rewards for each item from the reward model, and then return the data to the training task. The training task sends updated parameters to the external inference service via a Message Queue (MQ) every 1000 steps. The overall post-training process is summarized in Figure 8.

4. Evaluation

4.1. Evaluation Metric

We assess model performance through the following metrics:

- **Cross-entropy loss**: Next-token prediction loss $\mathcal{L}_{\text{NTP}}$ curves.
- **P (preference)-Score**: Learned comprehensive evaluation metric, as detailed in Section 2.4.1.
- **xtr metrics**: A set of user engagement indicators derived from a pre-trained ranking model (Chang et al., 2023; Wang et al., 2024) currently deployed in our system, including:
  - **lvtr** (Long View Through Rate): Predicted probability of significant video viewing
  - **vtr** (View Through Rate): Predicted probability of video viewing
  - **ltr** (Like Through Rate): Predicted probability of video liking
  - **wtr** (Follow Through Rate): Predicted probability of the creator following
  - **cmtr** (Comment Through Rate): Predicted probability of video commenting

For P-Score and xtr reward metrics, our evaluation system operates on streaming data where values may vary across different periods. Consequently, identical metrics may show different absolute values across experiments due to temporal variations in the data stream. However, we ensure reliable evaluation by conducting comparative experiments within the same periods and averaging results over sufficiently long observation windows, making our findings statistically confident.

4.2. Scaling

4.2.1. Training Scaling

Parameters Scaling The OneRec series includes models of varying sizes: OneRec-0.015B, OneRec-0.121B, OneRec-0.935B, and OneRec-2.633B, as detailed in Table 1. We investigated the impact of model parameter count on performance. Figure 9 illustrates the loss curves for these models, demonstrating a clear scaling trend where larger models achieve lower loss as training progresses. This indicates a strong capability for performance improvement with increased model size.

Regarding the influence of training data size, our experiments show that performance converges rapidly within the initial approximately 10 billion samples. While the rate of improvement diminishes significantly beyond this point, performance does not completely plateau and continues to benefit, albeit more slowly, from additional data (i.e., beyond 100 billion samples). This suggests that while substantial gains are achieved early in training, further, more gradual improvements are possible with larger datasets.

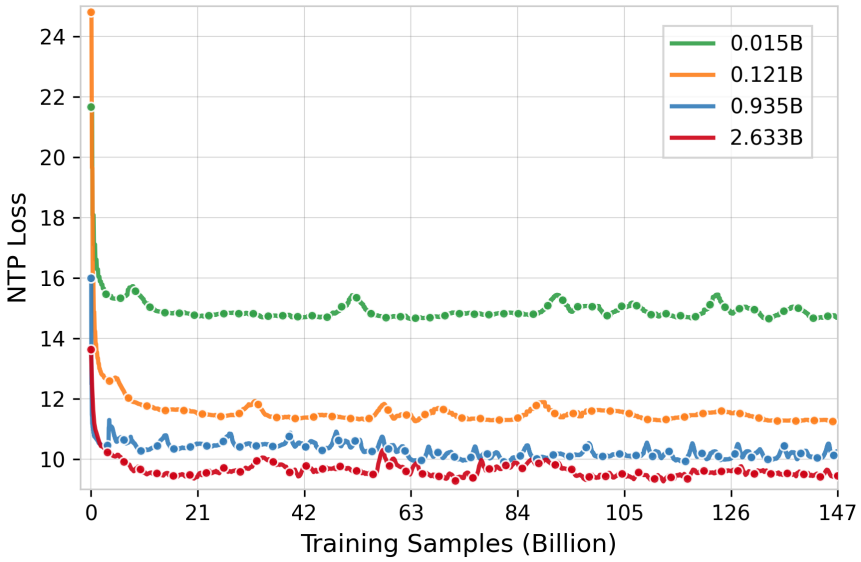

Figure 9 | Comparison of loss curves for different OneRec model sizes, showing loss scaling with training samples.

As model parameters scale up, load balancing among experts becomes a critical issue. Uneven expert utilization can lead to training inefficiency and suboptimal performance. We adopt DeepSeek's loss-free load balancing strategy (Liu et al., 2024), which maintains expert utilization balance without introducing additional loss terms. With this strategy, we observe a loss reduction of 0.2, demonstrating its effectiveness in improving convergence for scaled OneRec models.

Beyond parameter scaling, we conduct additional experiments to validate the effectiveness of scaling across other key dimensions using our 0.935B model. These experiments encompass feature scaling (examining the impact of comprehensive feature engineering), codebook scaling (investigating the effect of vocabulary size expansion), and inference scaling (analyzing the influence of beam search parameters). Each dimension demonstrates distinct scaling behaviors and provides valuable insights for future model optimization.

Feature Scaling To investigate the impact of feature engineering on model performance, we compare the model with two input configurations: a baseline using only item ID `vid` embeddings from 256 positive-feedback items, and an enhanced version incorporating the comprehensive feature set described in our methodology. As shown in Figure 10 and Table 2, the enhanced model with additional features achieves lower training loss and substantial improvements across multiple dimensions of recommendation quality.

Figure 10 | Training loss comparison with and without additional features.

Metric	w/o. feature	w/. feature	Impr.
lvtr	0.4940	0.5500	11.34%
vtr	0.8730	0.8901	1.96%
ltr	0.0391	0.0441	12.79%
wtr	0.0190	0.0224	17.89%
cmtr	0.0919	0.1010	9.90%
P-score	0.0749	0.0966	28.88%

Table 2 | Performance comparison with and without additional features.

Codebook Scaling To investigate the impact of codebook size on model performance, we experiment by expanding the codebook from 8,192 to 32,768. It is important to note that NTP loss, as defined in our parameter scaling experiments, cannot be directly used for comparison here. This is because an increase in codebook size inherently expands the candidate set for the cross-entropy loss calculation, rendering direct loss comparisons misleading. Consequently, we evaluate performance using reward-based metrics. The performance improvements across various metrics are presented in Table 3. As shown in the result, increasing the codebook size yields significant improvements in playtime metrics and a slight gain in interaction metrics.

Infer Scaling We investigate the impact of different numbers of generated items in inference (Pass@K) on model performance. As detailed in Table 4, increasing K of Pass@K from 8 to 512 results in consistent performance improvements across all evaluated metrics. However, further increasing K from 512 to 1,024 yields only marginal gains. Considering the trade-off between performance improvements and the associated computational resource consumption, we select K=512 for deployment in our production environment.

Metric	Size=8K	Size=32K	Impr.
lvtr	0.5118	0.5245	2.48%
vtr	0.9384	0.9491	1.14%
ltr	0.0298	0.0299	0.34%
wtr	0.0153	0.0154	0.65%
cmtr	0.0650	0.0664	2.15%
P-score	0.2516	0.2635	4.75%

Table 3 | Codebook Scaling.

Metric	Pass@8	Pass@64	Pass@512	Pass@1024	Impr.
lvtr	0.3675	0.4927	0.5351	0.5443	48.11%
vtr	0.9444	0.9462	0.9513	0.9530	0.91%
ltr	0.0278	0.0346	0.0425	0.0452	62.59%
wtr	0.0114	0.0138	0.0182	0.0197	72.81%
cmtr	0.0350	0.0566	0.0809	0.0891	154.57%
P-score	0.0811	0.2051	0.3375	0.3859	376.10%

Table 4 | Inference Pass@K Scaling.

4.2.2. Semantic Identifier Input Representation

As model sizes scale to billions of parameters, we explore an alternative input representation strategy that leverages video semantic identifiers for user interaction histories instead of constructing separate sparse embeddings for video identifiers (`vid`). This semantic identifier input achieves performance comparable to traditional sparse embedding methods, while offering significant advantages in parameter efficiency, communication overhead, and sequence processing capacity that make it particularly promising for further scaling exploration.

Scaling Performance Analysis As shown in Figure 11, our empirical analysis reveals that at scale (2.6B parameters), the semantic identifier input approach achieves performance comparable to or exceeding traditional sparse embedding methods.

Figure 11 | Training loss comparison between OneRec-2.633B with semantic identifier input and sparse embedding input.

Metric	VID	Semantic ID	Impr.
lvtr	0.4447	0.4467	0.45%
vtr	0.8725	0.8726	0.01%
ltr	0.0336	0.0336	0.00%
wtr	0.0104	0.0105	0.96%
cmtr	0.0565	0.0573	1.42%
P-score	0.0371	0.0378	1.74%

Table 5 | Performance comparison between OneRec-2.633B with semantic identifier input and sparse embedding input.

Advantages and Future Scaling The semantic identifier approach provides several key advantages over traditional sparse embedding methods, making it particularly attractive for further scaling exploration:

- **Parameter Efficiency:** By sharing embeddings between input and output representations, the model eliminates the need for separate sparse embedding tables for `vid`. This dramatically reduces the total parameter count, particularly for Kuaishou with billions of items.
- **Communication Efficiency:** In distributed training environments, sparse embedding operations require extensive parameter server communication for embedding lookup and gradient updates. The semantic identifier approach reduces communication overhead by leveraging dense operations and shared vocabulary, leading to faster training throughput and reduced communication bottlenecks.
- **Extended Sequence Capacity:** The elimination of large sparse embedding tables enables the allocation of computational resources toward processing longer user interaction sequences. This allows the model to capture more comprehensive user preference evolution patterns, potentially extending sequence lengths from thousands to tens of thousands of interactions.
- **Representation Consistency:** Sharing the same semantic space between input and output ensures representational consistency and enables the model to learn more coherent item-to-item relationships. This unified representation has the potential to facilitate better generalization across different recommendation scenarios.

Given these compelling advantages and the competitive performance demonstrated at the 2.6B parameter scale, we are actively pursuing further scaling exploration based on semantic identifier input representation. This approach promises to unlock new possibilities for large-scale recommendation systems while maintaining computational efficiency and architectural elegance.

4.3. Reinforcement Learning

4.3.1. User Preference Alignment

Defining what constitutes a "good" recommendation has always been a challenging task. To rigorously verify RL's impact, we use the single-objective *vtr* (view-through rate) as the reward, which corresponds to online metrics such as Watch Time and App Stay Time. The reported online results are relative improvements compared to Kuaishou's traditional recommendation system, referred to as the overall baseline. *Relative Impr.* in the table indicates the relative enhancement of the latter group over the former group.

Notably, while using *vtr* as the reward can significantly improve duration metrics, it does not necessarily indicate a high-quality recommendation, as other metrics, such as Video View, which represent the number of videos viewed, may decrease significantly. We primarily focus on Watch Time and App Stay Time to find the optimal RL setting, and ultimately use it to validate the benefits of the P-Score reward.

Sampling Efficiency Reinforcement learning optimizes the probability distribution of sampled items to increase the likelihood of selecting high-reward items, thereby significantly enhancing sampling efficiency. To quantify this effect, we conduct multi-point sampling experiments at pass@32, pass@128, and pass@512, with results summarized in Table 6. Treating the model without RL as the baseline, we define the improvement in app stay time as the sampling efficiency gap. Notably, RL shows the most substantial improvement gap at pass@32, indicating that the accuracy of top-ranked items is significantly enhanced. This improvement is crucial for reducing sampling overhead, as it ensures high precision when sampling a small number of items. In recommendation systems, balancing cost and benefit is essential, and the enhanced accuracy at lower sample numbers K provides a solid foundation for achieving this balance.

	Method	vtr	Watch time	App Stay Time	Video View[1]
Pass@32	OneRec w/o RL	0.1978	+1.62%	-0.10%	-4.18%
	OneRec w/ RL	0.2138	+3.17%	+0.39%	-9.87%
	Relative Impr.	+8.08%	+1.55%	+0.49%↑↑↑	-3.69%
Pass@128	OneRec w/o RL	0.2239	+4.61%	+1.11%	-12.75%
	OneRec w/ RL	0.2387	+5.22%	+1.49%	-15.06%
	Relative Impr.	+6.61%	+1.53%	+0.38%↑↑	-2.65%
Pass@512	OneRec w/o RL	0.2444	+6.32%	+1.66%	-15.54%
	OneRec w/ RL	0.2494	+5.88%	+1.75%	-13.88%
	Relative Impr.	+2.05%	-0.41%	+0.09%↑	+1.97%

Table 6 | The impact of reinforcement learning under different numbers of generated items (Pass@K) during inference.

[1]Video View is provided for reference only, as our primary focus is on Watch Time and App Stay Time to determine the optimal RL setting.

Search Space In ECPO training, expanding the action search space increases the likelihood of discovering the optimal item with maximum reward, albeit at higher computational costs. To investigate this trade-off, we examine how the search space size (i.e., group size) affects performance. The results for pass@128 are summarized in Table 7. From Table 7, we observe a significant improvement in performance when the group size is increased from 128 to 512. This clearly demonstrates the positive impact of expanding the search space. It is somewhat disappointing that increasing the search space to 2048 does not yield much additional benefit, which might be due to the current reference model's diversity not being sufficient to discover more and better items. Nonetheless, this finding is promising, and we empirically suggest setting the ECPO training group size to approximately four times the inference output quantity for optimal results.

Group Size	vtr	Watch time	App Stay Time	Video View[1]
0(w/o RL)	0.2198	+4.61%	+1.11%	-12.75%
128	0.2303	+5.22%	+1.49%	-15.06%
512	0.2350	+5.73%	+1.82%	-15.49%
2048	0.2352	+5.84%	+1.78%	-15.49%

Table 7 | Performance of different group sizes when calculating ECPO loss on Pass@128.

Search Strategy Reinforcement learning for large language models typically employs top-k and top-p sampling for sample generation. In OneRec, we also explore beam search as an alternative strategy. Table 8 compares the results of these two approaches, revealing that beam search significantly outperforms top-k and top-p sampling in OneRec's reinforcement learning framework. This improvement stems from the inherent regularity of semantic ID structures, which follow a prefix tree encoding scheme and thus align well with the systematic exploration of beam search.

	vtr	Watch time	App Stay Time	Video View[1]
Top-k+Top-p	0.2131	+4.45%	+1.16%	-13.61%
Beam Search	0.2162	+5.35%	+1.76%	-13.30%
Relative Impr.	+1.45%	+0.87%	+0.60%	+0.36%

Table 8 | Performance of reinforcement learning with different search strategies.

Reference Model In this section, we compare two reference models for strategy generation in ECPO: (1) the pre-trained model (off-policy) and (2) the current policy model (on-policy). The experimental results are summarized in Table 9. From the table, it is evident that using the current policy model yields better results, especially in offline reward evaluation. This indicates that the on-policy approach allows the model to continuously teach itself, breaking through the limitations of the reference model and achieving a higher upper limit. However, in terms of online performance, the improvement with the on-policy approach is not very significant. This is due to the suboptimal definition of the reward, leading to slight reward hacking. We will focus on this aspect as a key direction for future work.

P-Score Reward In this section, we observe the comprehensive improvements achieved through reinforcement learning when using P-Score as the reward. Based on the conclusions from the above ablation experiments, we select the optimal RL setting, which involves using beam search for RL

Reference Model	vtr	Watch time	App Stay Time	Video View[1]
Pre-trained Model	0.2262	+5.35%	+1.51%	-13.51%
Current Policy Model	0.2389	+6.19%	+1.56%	-13.89%
Relative Impr.	+5.61%	+0.79%	+0.04%	-13.89%

Table 9 | Performance of reinforcement learning with different reference models.

sample generation and employing the current policy model as the reference model. We examine the impact of RL in two scenarios, including Kuaishou and Kuaishou Lite, with the results summarized in Table 1. From the table, we can conclude that in both scenarios, P-Score significantly improves App Stay Time and Watch Time while also increasing Video View, indicating an enhancement in the overall user recommendation experience.

Scenario	Watch time	App Stay Time	Video View
Kuaishou	+0.21%	+0.26%	+0.17%
Kuaishou Lite	+0.71%	+0.22%	+0.35%

Table 10 | The relative improvement of OneRec with P-Score Reward compared to without it in the Kuaishou and Kuaishou Lite scenarios.

4.3.2. Generation Format Regularization

In this section, we conduct experiments to verify the effectiveness of format reward. As mentioned in Section 2.4.2, after incorporating reinforcement learning into the pre-trained model, the legality of the model's output significantly drops to below 50% due to the squeezing effect. This means that more than half of the generated semantic IDs do not correspond to actual video IDs, which is detrimental to the stability of recommendations and the scalability of inference. We evaluate the impact of format reward by comparing two sample selection methods for computing format loss: (1) selecting the top-5 highest-probability samples from 128 generated candidates, and (2) randomly selecting 5 samples.

Figure 12 illustrates their effects on output legality. The left figure shows legality rates across all 128 generated samples, while the right panel focuses on the selected samples. Without format rewards, baseline legality remains below 50%. The Top-k Selection approach produces an interesting pattern: while overall legality initially rises then falls, the selected samples rapidly achieve 100% legality, suggesting the model learns to generate legal outputs only within the top-ranked subset. In contrast, Random Selection presents a more challenging learning objective, yet drives steady improvement - ultimately reaching 95% legality without showing a decline.

Notably, format reward integration yields benefits beyond legality alone. Online metrics demonstrate substantial gains: +0.13% in APP Stay Time and +0.30% in Watch Time. This experimental case not only validates the format reward mechanism but also highlights the critical role of careful reward design in reinforcement learning systems.

4.3.3. Industrial Scenario Alignment

In this section, we present a practical example of using reinforcement learning to address industrial challenges. On the Kuaishou platform, viral content farms represent a significant portion of content creators, primarily producing repurposed and clipped videos with inconsistent quality. While OneRec

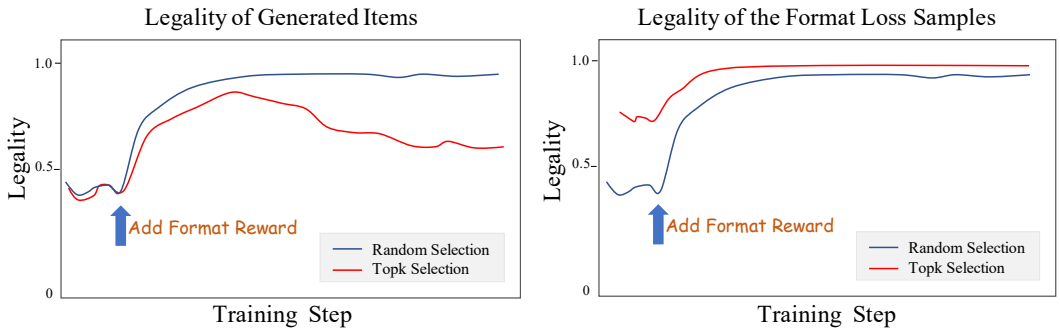

Figure 12 | The impact of training with format reward with samples obtained through different sampling strategies on the model's legality.

demonstrates superior performance over traditional recommendation systems across multiple business metrics, we observe that without proper post-filtering strategies, the exposure ratio of viral content increases significantly, which may negatively impact the platform's ecosystem.

The optimal proportion of viral content videos can be set to f. When the proportion exceeds f, we down-weight their P-score reward to suppress them while maintaining the system's perception of the quality of these contents.

$$r_i' = \begin{cases} r_i & \text{if } o_i \notin I_{\text{viral}} \\ \alpha r_i & \text{if } o_i \in I_{\text{viral}} \end{cases}, \tag{33}$$

where $\alpha \in (0, 1)$ is the suppression factor.

We term this approach Specific Industrial Reward (SIR). Experimental results show that SIR effectively reduces viral content exposure by 9.59% while maintaining stable performance on core metrics (Watch time and APP Stay Time). This experiment highlights OneRec's key advantage: the ability to achieve precise and consistent optimization through reinforcement learning's reward-shaping capability, a feature fundamentally unavailable in traditional recommendation systems.

4.4. Tokenizer

We employ three metrics to comprehensively evaluate our tokenization method, encompassing aspects of accuracy, resource utilization, and distribution uniformity:

- **Reconstruction Loss**: This metric assesses the accuracy with which discrete tokens reconstruct the original input, serving as an indicator of the model's fidelity in preserving the input data.
- **Codebook Utilization (Zhu et al., 2024)**: This metric evaluates the efficiency of vector usage within the codebook, reflecting how effectively the model leverages available resources to represent data.
- **Token Distribution Entropy (Bentz and Alikaniotis, 2016)**: Utilizing Shannon entropy, this metric quantifies the uniformity of token distribution, providing insight into the diversity and balance of token allocation across the model.

As shown in Table 11, compared to RQ-VAE, RQ-Kmeans's reconstruction loss is reduced by 25.18%, demonstrating superior accuracy in preserving input information. Simultaneously, RQ-Kmeans achieves perfect utilization (1.0000) in all three layers, indicating optimal resource efficiency in the codebook, while RQ-VAE shows slightly lower utilization rates in layers 2 and 3. Furthermore, RQ-Kmeans exhibits higher entropy values in all three layers compared to RQ-VAE, with significant improvements of 6.31%, 3.50%, and 1.44% in layers 1, 2, and 3, respectively, suggesting that RQ-Kmeans

Table 11 | Performance comparison of tokenization algorithms with a three-layer 8,192 codebook.

		RQ-VAE	RQ-Kmeans
Reconstruction Loss ↓		0.0548	**0.0410**
Codebook Utilization ↑	layer 1	**1.0000**	**1.0000**
	layer 2	0.9963	**1.0000**
	layer 3	0.9958	**1.0000**
Token Distribution Entropy ↑	layer 1	8.3892	**8.9191**
	layer 2	8.4805	**8.7770**
	layer 3	8.6037	**8.7276**

produces a more uniform and balanced token distribution, which is beneficial for model stability and generalization capability. These comprehensive results demonstrate that RQ-Kmeans outperforms RQ-VAE across all three evaluation metrics, making it a more effective choice for tokenization.

Further qualitative analyses of item representation and tokenization quality are provided in Appendix C.

4.5. Online A/B Test

We deployed OneRec in two major short-video scenarios on Kuaishou: the main Kuaishou feed and Kuaishou Lite feed - the platform's highest-traffic scenarios with daily active users of 400 million. Using a 5% traffic experimental group observed over one week, our primary metrics were APP Stay Time (reflecting total user engagement time) and LT7 (7-day Lifetime). Two experimental groups were established: one employing a pure generative model (OneRec) and another augmenting generative outputs with reward model based selection (OneRec with RM Selection). As shown in Table 12, the pure generative model with RL-based user preference alignment remarkably matched the performance of the entire complex recommendation system. Further applying reward model selection achieved statistically significant **improvements of +0.54% and +1.24% in APP Stay Time, and +0.05% and +0.08% in LT7** on these two scenarios, respectively. Notably, improvements of **0.1% in APP Stay Time and 0.01% in LT7** are already considered statistically significant on Kuaishou. Additionally, OneRec demonstrated significant gains across all interaction metrics (likes, follows, comments, etc.), indicating its ability to converge multi-task systems to a more balanced equilibrium without seesaw effects. After validation, we've expanded deployment to approximately 25% of total QPS, with implementation details available in Appendix B.

In addition to Kuaishou's short video recommendation scenarios, experiments have also been conducted in one of its significant business scenes — Local Life Service. The results demonstrate that OneRec achieves **a 21.01% growth in GMV, a 17.89% increase in order volume, an 18.58% rise in buyer numbers, and a 23.02% increase in new buyer acquisition**. Consequently, the system has now taken over **100% of QPS** for this business scenario. After full deployment, we observe even stronger growth across all metrics compared to the initial experimental phase. These results prove OneRec's generalizability across diverse business contexts for enhanced recommendation performance.

Infrastructure and Efficiency We utilize NVIDIA L20 GPUs for inference, and each server is equipped with 4 GPUs and 2 CPUs, connected via PCIe. We adopt Kuaishou's prediction platform - UniPredict to support online traffic. The inference service and embedding service are deployed in a 200Gb RDMA data center, leveraging RoCE networking. The maximum inter-machine communication bandwidth reaches 800Gb. In order to improve the efficiency, we employ TensorRT to compile and optimize the model's computation graph. Through custom plugins, we achieve high-performance implementations

Table 12 | The absolute improvement of OneRec compared to the current multi-stage system in the online A/B testing setting.

Scenarios	Online Metrics	OneRec	OneRec with RM Selection
Kuaishou	App Stay Time	+0.01%	+0.54%
	Watch Time	+0.07%	+1.98%
	Video View	+1.98%	+2.52%
	Like	-2.00%	+2.43%
	Follow	-2.88%	+3.24%
	Comment	-1.56%	+5.27%
	Collect	-0.61%	+2.93%
	Foward	+0.27%	+5.90%
Kuaishou Lite	App Stay Time	+0.06%	+1.24%
	Watch Time	+0.05%	+3.28%
	Video View	+2.40%	+3.39%
	Like	-2.64%	+1.49%
	Follow	-2.75%	+2.28%
	Comment	-2.23%	+3.20%
	Collect	-1.76%	+1.91%
	Foward	-1.86%	+3.48%

of cross-attention, MoE, and other operations. Combined with batching and MPS techniques, we achieve a 5 $\times$ throughput improvement, reaching an MFU of 28.8%.

5. Conclusion, Limitations, and Future Directions

In this paper, we introduce OneRec, a novel end-to-end generative recommendation architecture. Built as an encoder-decoder model, it compresses users' lifelong behavior sequences via its encoder to derive user interests, while leveraging Mixture-of-Experts (MoE) to massively scale decoder parameters for precise short-video recommendation decoding. During post-training, we develop a customized reinforcement learning (RL) framework to refine recommendations by aligning model outputs with the reward function. Thanks to meticulous engineering optimizations, OneRec achieves 23.7% and 28.6% Model FLOPs Utilization (MFU) in training and inference — a dramatic improvement from single-digit baselines — closing the gap with the mainstream AI community. Notably, this compute-intensive design operates at 10.6% the OPEX of conventional recommender systems. Comprehensive evaluations demonstrate that OneRec has surpassed existing recommendation systems in both effectiveness and efficiency. While acknowledging its powerful performance and high cost-effectiveness, we also recognize some limitations of OneRec and plan to strategically invest in the following areas:

- Inference Stage Scaling: The step scaling during the inference phase is not yet apparent, indicating

that OneRec currently lacks strong reasoning capabilities.

- Multimodal Integration: OneRec has not yet integrated with LLMs (Large Language Models) and VLMs (Vision Language Models). User behavior is also a modality, and in the future, we plan to design solutions that allow user behavior modality to become a native multimodal model, similar to vision and audio alignment.
- Reward System Design: The reward system design is still very rudimentary, which is an exciting aspect. Historically, recommendation systems were not end-to-end, making it difficult to define and iterate on what constitutes a good recommendation result. Under the OneRec architecture, the reward system impacts both online results and offline training. We believe that the structure will soon lead to technological breakthroughs in the reward system for recommendations.

OneRec establishes an entirely new architecture, introducing a transformative framework for technological evolution, business value optimization, and team collaboration. While currently not yet deployed across all traffic scenarios in Kuaishou, we have adopted this as our foundational approach to systematically push the boundaries of algorithmic innovation while refining team collaboration mechanisms, thereby building scalable infrastructure capable of supporting traffic growth at scale.

References

C. Bentz and D. Alikaniotis. The word entropy of natural languages. *arXiv preprint arXiv:1606.06996*, 2016.

J. Cao, P. Xu, Y. Cheng, K. Guo, J. Tang, S. Wang, D. Leng, S. Yang, Z. Liu, Y. Niu, et al. Pantheon: Personalized multi-objective ensemble sort via iterative pareto policy optimization. *arXiv preprint arXiv:2505.13894*, 2025.

J. Chang, C. Zhang, Z. Fu, X. Zang, L. Guan, J. Lu, Y. Hui, D. Leng, Y. Niu, Y. Song, et al. Twin: Two-stage interest network for lifelong user behavior modeling in ctr prediction at kuaishou. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3785–3794, 2023.

H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

W. Fedus, J. Dean, and B. Zoph. A review of sparse expert models in deep learning. *arXiv preprint arXiv:2209.01667*, 2022.

A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

H. Guo, R. Tang, Y. Ye, Z. Li, and X. He. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

T. Henighan, J. Kaplan, M. Katz, M. Chen, C. Hesse, J. Jackson, H. Jun, T. B. Brown, P. Dhariwal, S. Gray, et al. Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*, 2020.

J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

S. Hu, Y. Tu, X. Han, C. He, G. Cui, X. Long, Z. Zheng, Y. Fang, Y. Huang, W. Zhao, et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. d. l. Casas, E. B. Hanna, F. Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

D. Lee, C. Kim, S. Kim, M. Cho, and W.-S. Han. Autoregressive image generation using residual quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11523–11532, 2022.

J. Li, D. Li, S. Savarese, and S. Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pages 19730–19742. PMLR, 2023.

A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

X. Luo, J. Cao, T. Sun, J. Yu, R. Huang, W. Yuan, H. Lin, Y. Zheng, S. Wang, Q. Hu, et al. Qarm: Quantitative alignment multi-modal recommendation at kuaishou. *arXiv preprint arXiv:2411.11739*, 2024.

L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Q. Pi, G. Zhou, Y. Zhang, Z. Wang, L. Ren, Y. Fan, X. Zhu, and K. Gai. Search-based user interest modeling with lifelong sequential behavior data for click-through rate prediction. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 2685–2692, 2020.

R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741, 2023.

S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

S. Rajput, N. Mehta, A. Singh, R. Hulikal Keshavan, T. Vu, L. Heldt, L. Hong, Y. Tay, V. Tran, J. Samost, et al. Recommender systems with generative retrieval. *Advances in Neural Information Processing Systems*, 36, 2024.

Y. Ren and D. J. Sutherland. Learning dynamics of llm finetuning. *arXiv preprint arXiv:2407.10490*, 2024.

S. Rendle. Factorization machines. In *2010 IEEE International conference on data mining*, pages 995–1000. IEEE, 2010.

F. Ricci, L. Rokach, and B. Shapira. Introduction to recommender systems handbook. In *Recommender systems handbook*, pages 1–35. Springer, 2010.

Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. Li, Y. Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

N. Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Z. Si, L. Guan, Z. Sun, X. Zang, J. Lu, Y. Hui, X. Cao, Z. Yang, Y. Zheng, D. Leng, et al. Twin v2: Scaling ultra-long user behavior sequence modeling for enhanced ctr prediction at kuaishou. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 4890–4897, 2024.

R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

X. Wang, J. Cao, Z. Fu, K. Gai, and G. Zhou. Home: Hierarchy of multi-gate experts for multi-task learning at kuaishou. *arXiv preprint arXiv:2408.05430*, 2024.

X. Yang, Y. Zhu, Y. Zhang, X. Wang, and Q. Yuan. Large scale product graph construction for recommendation in e-commerce. *CoRR*, abs/2010.05525, 2020.

B. Zheng, Y. Hou, H. Lu, Y. Chen, W. X. Zhao, M. Chen, and J.-R. Wen. Adapting large language models by integrating collaborative semantics for recommendation. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 1435–1448. IEEE, 2024.

G. Zhou, X. Zhu, C. Song, Y. Fan, H. Zhu, X. Ma, Y. Yan, J. Jin, H. Li, and K. Gai. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1059–1068, 2018.

L. Zhu, F. Wei, Y. Lu, and D. Chen. Scaling the codebook size of vqgan to 100,000 with a utilization rate of 99%. *arXiv preprint arXiv:2406.11837*, 2024.

D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

Appendix

A. Contributions

Within each role, authors are listed alphabetically by their first name. Names marked with * denote individuals who have departed from our team.

Core Contributors

Guorui Zhou
Jiaxin Deng
Jinghao Zhang
Kuo Cai
Lejian Ren
Qiang Luo
Qianqian Wang
Qigen Hu
Rui Huang
Shiyao Wang
Weifeng Ding*
Wuchao Li
Xinchen Luo
Xingmei Wang
Zexuan Cheng
Zixing Zhang

Contributors

Bin Zhang
Boxuan Wang
Chaoyi Ma
Chengru Song
Chenhui Wang
Di Wang
Dongxue Meng
Fan Yang
Fangyu Zhang
Feng Jiang
Fuxing Zhang
Gang Wang
Guowang Zhang
Han Li
Hengrui Hu
Hezheng Lin*
Hongtao Cheng
Hongyang Cao
Huanjie Wang
Jiaming Huang
Jiapeng Chen
Jiaqiang Liu
Jinghui Jia
Kun Gai
Lantao Hu
Liang Zeng
Liao Yu
Qiang Wang
Qidong Zhou
Shengzhe Wang
Shihui He
Shuang Yang
Shujie Yang
Sui Huang
Tao Wu
Tiantian He
Tingting Gao
Wei Yuan
Xiao Liang
Xiaoxiao Xu
Xugang Liu
Yan Wang
Yi Wang
Yiwu Liu
Yue Song
Yufei Zhang
Yunfan Wu
Yunfeng Zhao
Zhanyu Liu

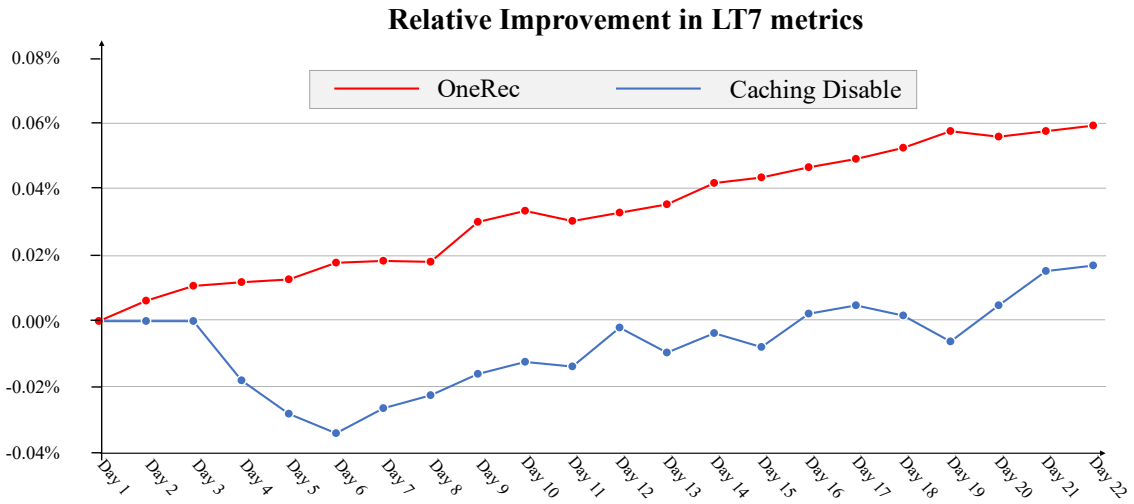

Figure 13 | Comparative analysis of OneRec vs. Caching Disabled Architecture on LT7 growth trends.

B. Implementation Details of Online A/B Test

In this section, we present the implementation details of OneRec in online A/B testing. In recommendation systems, a user's request typically triggers various system modules to generate real-time recommendation results. However, in practical applications, the massive QPS (the peak QPS can exceed 400k) necessitates substantial resources to handle such high concurrency. To address this, our system incorporates a caching mechanism: for each user request, the system returns k recommendation results. Apart from the items actually exposed, the remaining items are stored as candidates in a cache pool. When the system experiences a high QPS load, cached results are retrieved for display, achieving a trade-off between resource usage and real-time performance. Thus, we broadly categorize QPS into real-time and degraded (cached) traffic, and OneRec's online experiment specifically upgrades this degraded portion. There are two primary reasons for this experimental setup:

1. The previous caching mechanism significantly sacrificed the benefits of timeliness, affecting user experience during peak evening hours with high request volumes. While "disabling the caching mechanism" would incur substantial resource costs, OneRec's highly efficient end-to-end pipeline and optimized MFU drastically reduce the system's OPEX while delivering notable performance improvements.

2. OneRec represents an entirely new architecture, introducing a fresh paradigm for technical iteration, business optimization, and team collaboration. We use this portion of traffic as a starting point to continuously explore technical boundaries and team collaboration mechanisms, building a robust foundation for handling more traffic.

As mentioned in Section 4.5, our experimental group traffic is 5%, with OneRec applied to 25% of the degraded traffic within this group. Despite this limited scope, we observe significant performance gains across two scenarios, achieving 0.54% and 1.24% improvements in app stay time. For a more rigorous comparison, we allocate an additional 1% experimental group with caching disabled (all traffic requesting real-time recommendations). Even against this baseline, OneRec demonstrates superior performance (shown in Table 13). We also observe the LT7 metric growth patterns between OneRec and the caching disabled strategy. Figure 13 indicates that OneRec exhibits significantly stronger improvement trends.

Table 13 | The absolute improvement of OneRec compared to the current multi-stage system and caching disabled experimental group (all traffic requesting real-time recommendations) in the online A/B testing setting.

Scenarios	Online Metrics	vs. Current System	vs. Caching Disabled
Kuaishou	App Stay Time	+0.54%	+0.20%
	LT7	+0.05%	+0.03%
	Watch Time	+1.98%	+0.75%
	Video View	+2.52%	+1.79%
	Engagement Depth	+1.78%	+1.30%
	Like	+2.43%	+0.88%
	Follow	+3.24%	+1.29%
	Comment	+5.27%	+3.18%
	Collect	+2.93%	+0.73%
	Foward	+5.90%	+4.92%
Kuaishou Lite	App Stay Time	+1.24%	+0.55%
	LT7	+0.08%	+0.02%
	Watch Time	+3.28%	+1.58%
	Video View	+3.39%	+1.71%
	Engagement Depth	+2.89%	+2.49%
	Like	+1.49%	-1.71%
	Follow	+2.28%	+0.89%
	Comment	+3.20%	+0.60%
	Collect	+1.91%	-1.03%
	Foward	+3.48%	+1.35%

Through rigorous online A/B testing, our OneRec system has successfully replaced the original caching mechanism and now serves 25% of the traffic in Kuaishou's main scenarios.

C. Case Study for Tokenization

C.1. Representation Cases

To assess our aligned collaborative-aware multimodal representations, we contrast them with collaborative representations from traditional RS and pure multimodal representations extracted from caption/visual/OCR features. Figure 14, Figure 15, and Figure 16 present illustrative cases demonstrating video retrieval results from user history for query videos when leveraging different representations.

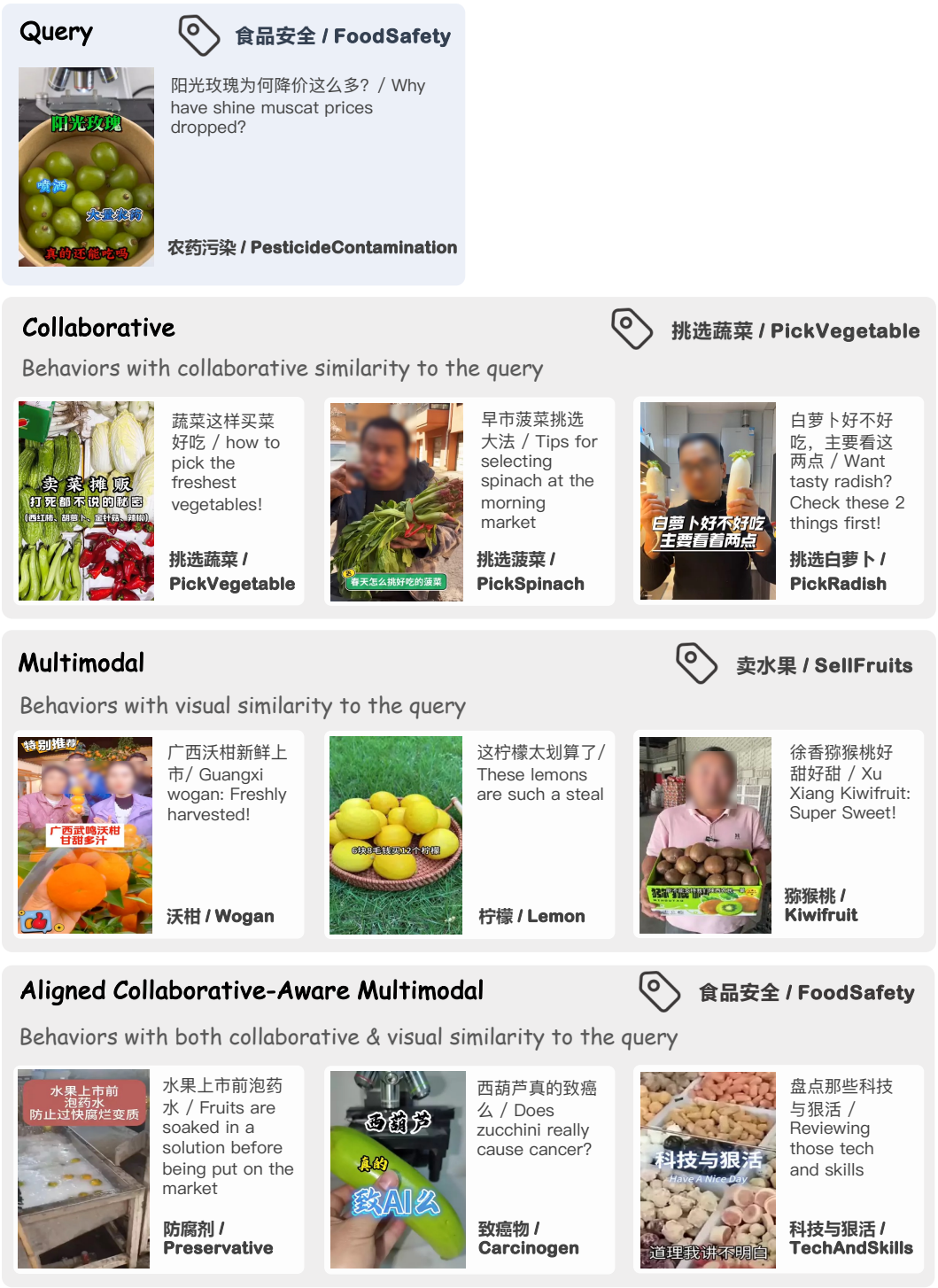

Figure 14 | Cases of top-ranked videos retrieved from user history triggered by the query using different representation types.

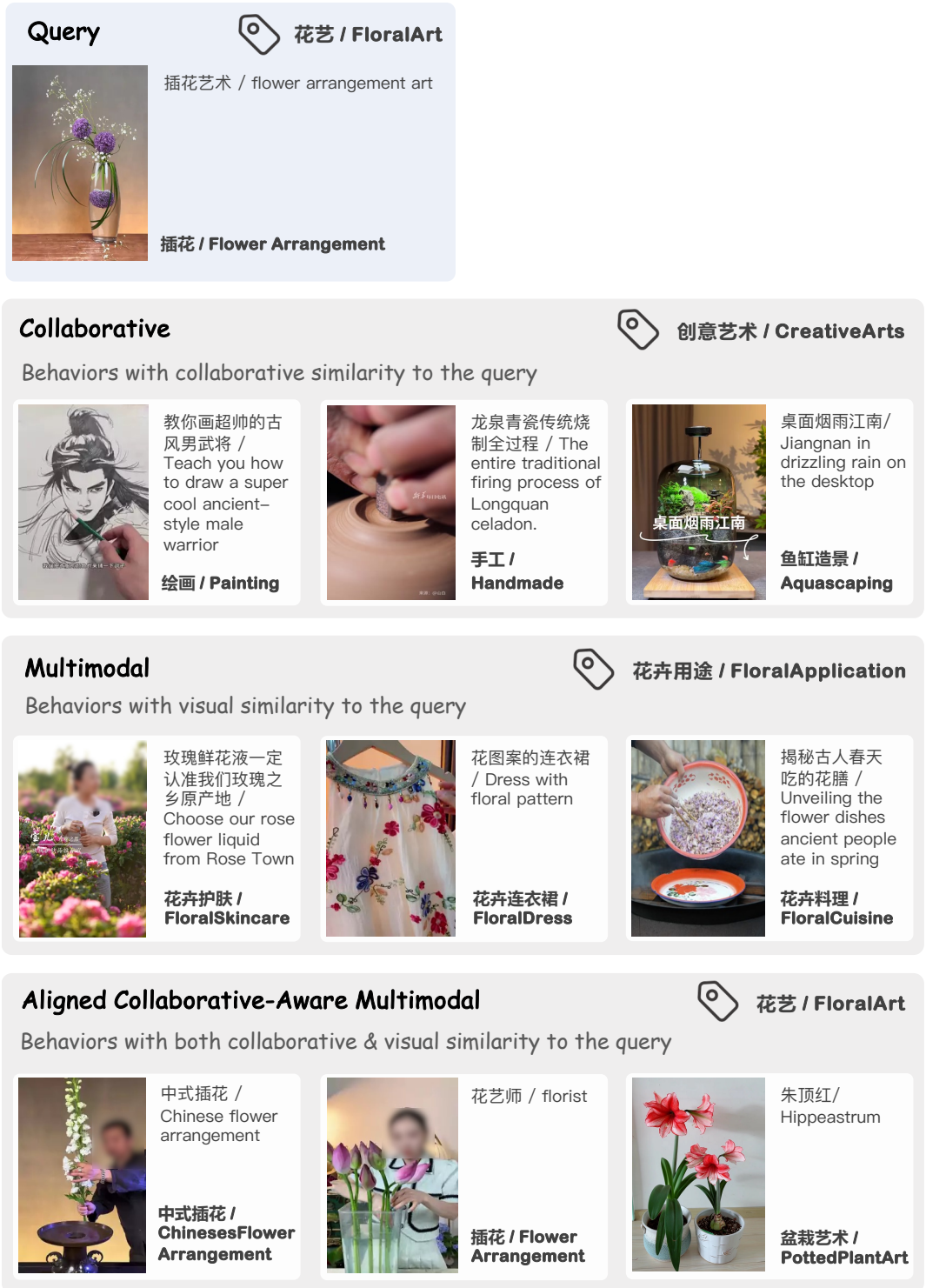

Figure 15 | Cases of top-ranked videos retrieved from user history triggered by the query using different representation types.

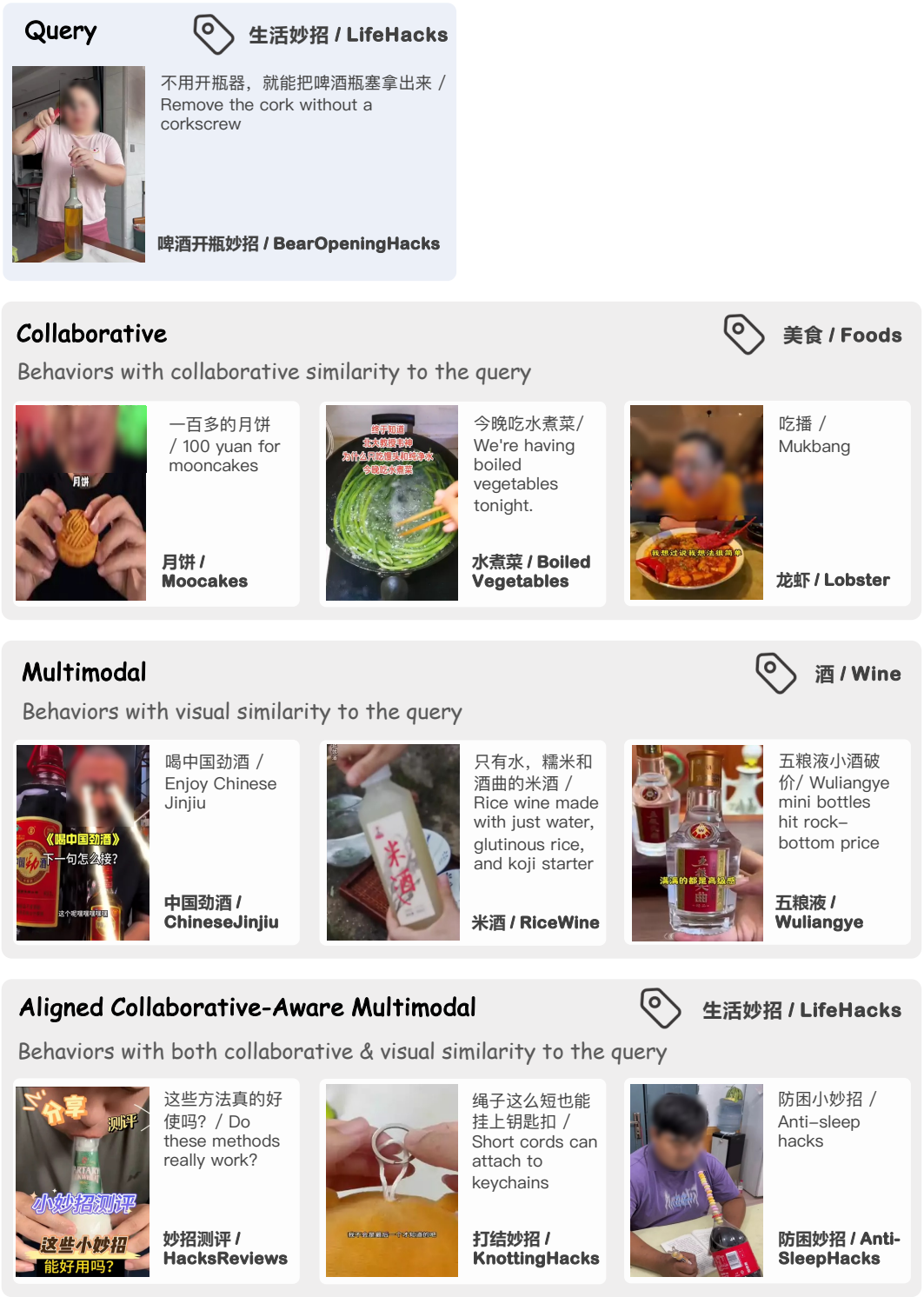

Figure 16 | Cases of top-ranked videos retrieved from user history triggered by the query using different representation types.

Our analysis reveals that collaborative representations—trained solely on collaborative signals—capture co-occurrence patterns but lack semantic relevance. This results in retrieved videos exhibiting categorical misalignment with query videos, as exemplified by painting content retrieved for a floral art query in Figure 15 (row 2). Conversely, pure multimodal representations retrieve videos with surface-level feature similarities (e.g., shared visual elements like fruit in Figure 14 (row 3) or wine in Figure 16 (row 3)) yet fundamental categorical discrepancies relative to query videos. In contrast, our representations integrate multimodal and collaborative signals, enabling the retrieval of videos with multifaceted relevance. This demonstrates that our representations overcome the limitations of unimodal ones by jointly modeling content semantics and behavioral patterns.

C.2. Tokenization Cases

We present cases of discrete item semantic identifiers generated by RQ-Kmeans in Figure 17 and Figure 18. Our tokenization method can produce coarse-to-fine item semantic identifiers, where the first codeword indicates the coarsest category, and the categories of the second and third codewords become increasingly finer.

Figure 17 | Cases of coarse-to-fine item semantic identifiers generated by RQ-Kmeans when $L_t = 5$.

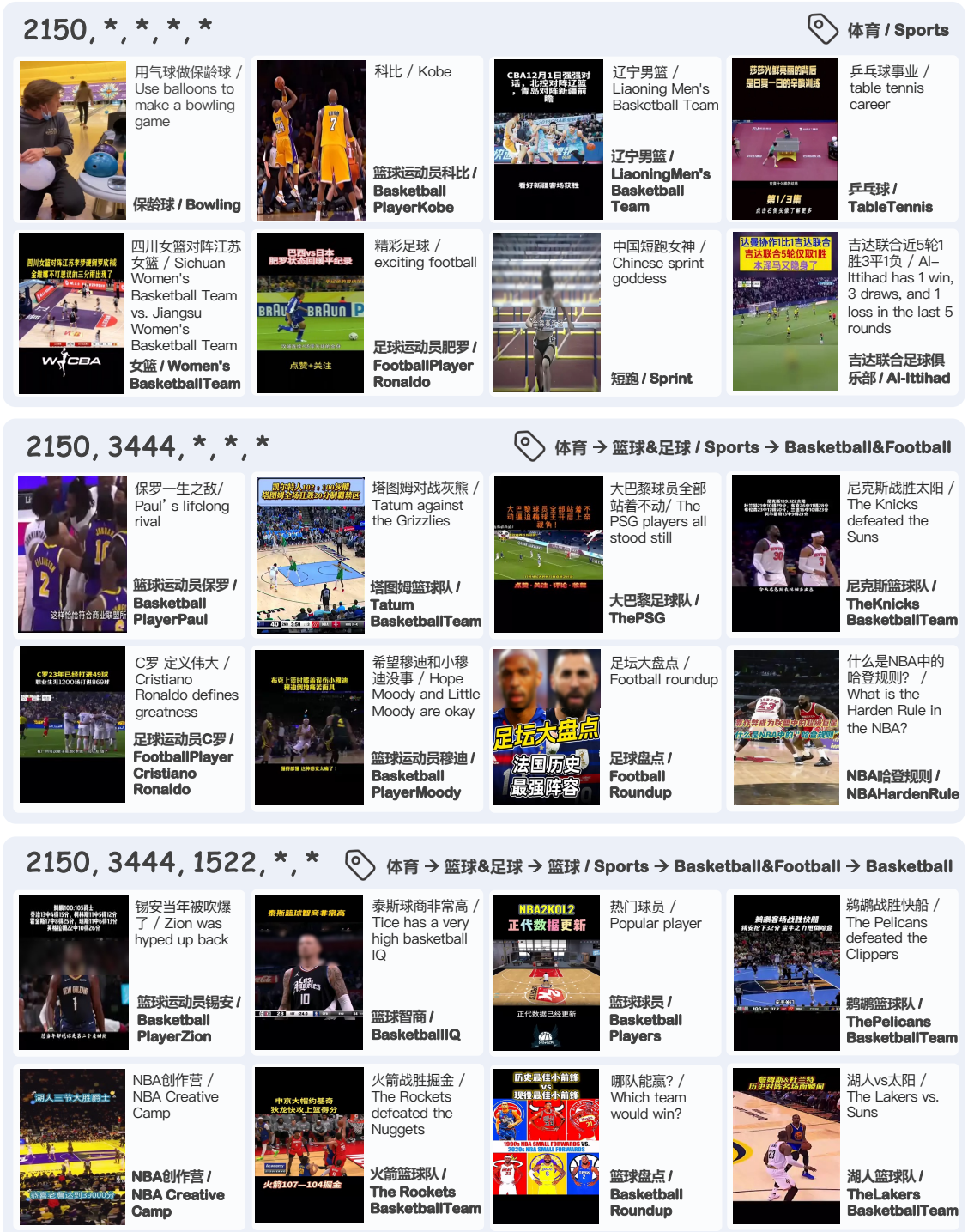

Figure 18 | Cases of coarse-to-fine item semantic identifiers generated by RQ-Kmeans when $L_t = 5$.

D. Notations

We summarize key notations used in this paper in Table 14 and Table 15.

Table 14 | Notation and Symbol Definitions in OneRec (Part 1)

General Notation	
d_{model}	Model hidden dimension (embedding dimension)
L_t	Number of quantization layers in tokenization (set to 3)
N_t	Codebook size for each quantization layer
$\{s_m^1, s_m^2, \ldots, s_m^{L_t}\}$	Coarse-to-fine semantic identifiers for item m
Item Tokenization	
d_t	Embedding dimension in tokenization (set to 512)
N_M	The number of original multimodal token vectors of an item (set to 1280)
$\mathbf{M}$	Multimodal token vectors from miniCPM-V-8B, $\mathbf{M} \in \mathbb{R}^{N_M \times d_t}$
$N_{\tilde{M}}$	The number of compressed multimodal token vectors of an item (set to 4)
$\mathbf{Q}^{(i)}$	Query tokens in QFormer at layer i, $\mathbf{Q}^{(i)} \in \mathbb{R}^{N_{\tilde{M}} \times d_t}$
$\tilde{\mathbf{M}}$	Compressed multimodal representation after QFormer, $\tilde{\mathbf{M}} \in \mathbb{R}^{N_{\tilde{M}} \times d_t}$
N_c	Number of QFormer layers (set to 4)
$\mathcal{R}^{(l)}$	Residual vectors at quantization layer l
$\mathcal{C}^{(l)}$	Codebook (K-means centroids) at quantization layer l
$\boldsymbol{c}_k^{(l)}$	k-th centroid in the codebook at layer l
s_i^l	Semantic identifier for item i at quantization layer l
$\mathcal{D}_{pair}$	Dataset of item pairs with high collaborative similarity
τ	Temperature coefficient for item-to-item loss
$\text{sim}(\cdot, \cdot)$	Similarity function used in item-to-item contrastive loss
t^k	The k-th caption token
Multi-Scale Feature Engineering	
L_s	Length of short-term behavior sequence (set to 20)
L_p	Length of positive-feedback behavior sequence (set to 256)
L_l	Length of lifelong behavior sequence (set to 2000)
$\mathbf{f}_u$	Concatenated user static features before dense transformation
$\mathbf{f}_s$	Concatenated short-term behavior features before dense transformation
$\mathbf{f}_p$	Concatenated positive-feedback behavior features before dense transformation
$\mathbf{f}_l$	Concatenated lifelong behavior features before dense transformation
$\mathbf{e}_*$	Individual feature embeddings (e.g., $\mathbf{e}_{\text{uid}}, \mathbf{e}_{\text{gender}}, \mathbf{e}_{\text{age}}$ for user static)
$\mathbf{e}_*^s$	Feature embeddings in the short-term pathway (e.g., $\mathbf{e}_{\text{vid}}^s, \mathbf{e}_{\text{aid}}^s, \mathbf{e}_{\text{tag}}^s$, etc.)
$\mathbf{e}_*^p$	Feature embeddings in the positive-feedback pathway
$\mathbf{e}_*^l$	Feature embeddings in the lifelong pathway
$\mathbf{h}_u$	User static pathway representation, $\mathbf{h}_u \in \mathbb{R}^{1 \times d_{\text{model}}}$
$\mathbf{h}_s$	Short-term pathway representation, $\mathbf{h}_s \in \mathbb{R}^{L_s \times d_{\text{model}}}$
$\mathbf{h}_p$	Positive-feedback pathway representation, $\mathbf{h}_p \in \mathbb{R}^{L_p \times d_{\text{model}}}$
$\mathbf{v}_l$	Processed lifelong features before QFormer compression, $\mathbf{v}_l \in \mathbb{R}^{L_l \times d_{\text{model}}}$
$\mathbf{h}_l^{(i)}$	Query vectors at QFormer layer i in the lifelong pathway
$\mathbf{h}_l$	Final lifelong pathway representation, $\mathbf{h}_l \in \mathbb{R}^{N_q \times d_{\text{model}}}$
N_q	Number of query tokens in lifelong pathway compression (set to 128)
N_l	Number of QFormer blocks in lifelong pathway (set to 2)
M	Threshold for hierarchical clustering termination

Table 15 | Notation and Symbol Definitions in OneRec (Part 2)

Encoder-Decoder Architecture	
L_{enc}	Number of transformer encoder layers
L_{dec}	Number of transformer decoder layers
$\mathbf{e}_{\text{pos}}$	Positional embeddings, $\mathbf{e}_{\text{pos}} \in \mathbb{R}^{(1+L_s+L_p+N_q)\times d_{\text{model}}}$
$\mathbf{z}^{(i)}$	Hidden states at encoder layer i
$\mathbf{z}_{\text{enc}}$	Final encoder output
$\mathbf{d}_m^{(i)}$	Decoder hidden states for item m at layer i
$\mathcal{S}_m$	Input sequence for item m: $\{s_{[\text{BOS}]}, s_m^1, s_m^2, \cdots, s_m^{L_t}\}$
$s_{[\text{BOS}]}$	Beginning-of-sequence token
N_{experts}	Number of expert networks in MoE layers
k	Top-k routing strategy parameter in MoE
$\text{Gate}_j(\mathbf{x})$	Gating weights for j-th expert in MoE layer
$\text{Expert}_j(\mathbf{x})$	Output of j-th expert network in MoE layer
Preference Alignment & Reinforcement Learning	
π_θ	Policy model with parameters θ
$\pi_{\theta_{old}}$	Old policy model (before update)
$\pi'_{\theta_{old}}$	Modified old policy with early clipping
G	Number of generated samples per user
K	Number of samples selected for format reward
r_i	Reward for generated item i (P-Score)
A_i	Advantage for generated item i
ϵ	Clipping parameter in ECPO
δ	Early clipping parameter in ECPO ($\delta > 0$)
$\mathcal{J}_{ECPO}(\theta)$	ECPO optimization objective
$\text{sg}(\cdot)$	Stop gradient operation
Industrial Constraints	
I_{legal}	Set of legal (valid) generated items
I_{viral}	Set of viral content items
f	Optimal proportion threshold for viral content
α	Down-weighting factor for viral content reward ($0 < \alpha < 1$)