TransAct V2: Lifelong User Action Sequence Modeling on Pinterest Recommendation

Xue Xia
xxia@pinterest.com
Pinterest
San Francisco, CA, USA

Saurabh Vishwas Joshi
sjoshi@pinterest.com
Pinterest
San Francisco, CA, USA

Kousik Rajesh
krajesh@pinterest.com
Pinterest
San Francisco, CA, USA

Kangnan Li
kangnanli@pinterest.com
Pinterest
San Francisco, CA, USA

Yangyi Lu
yangyilu@pinterest.com
Pinterest
San Francisco, CA, USA

Nikil Pancha*
npancha@pinterest.com
Pinterest
San Francisco, CA, USA

Dhruvil Deven Badani
dbadani@pinterest.com
Pinterest
San Francisco, CA, USA

Jiajing Xu
jiajing@pinterest.com
Pinterest
San Francisco, CA, USA

Pong Eksombatchai
pong@pinterest.com
Pinterest
San Francisco, CA, USA

Abstract

Modeling user action sequences has become a popular focus in industrial recommendation system research, particularly for Click-Through Rate (CTR) prediction tasks. However, industry-scale CTR models often rely on short user sequences, limiting their ability to capture long-term behavior. Additionally, these models typically lack an integrated action-prediction task within a point-wise ranking framework, reducing their predictive power. They also rarely address the infrastructure challenges involved in efficiently serving large-scale sequential models. In this paper, we introduce TransAct V2, a production model for Pinterest's Homefeed ranking system, featuring three key innovations: (1) leveraging very long user sequences to improve CTR predictions, (2) integrating a Next Action Loss function for enhanced user action forecasting, and (3) employing scalable, low-latency deployment solutions tailored to handle the computational demands of extended user action sequences. To overcome latency and storage constraints, we leverage efficient data-processing strategies and model-serving optimizations, enabling seamless industrial-scale deployment. Our approach's effectiveness is further demonstrated through ablation studies. Furthermore, extensive offline and online A/B experiments confirm major gains in key metrics, including engagement volume and recommendation diversity, showcasing TransAct V2's real-world impact.

CCS Concepts

• **Information systems** → **Web searching and information discovery**; **Content ranking**; **Personalization**;

Keywords

Personalization, Recommender Systems, Sequential Recommendation, User Interest Modeling

ACM Reference Format:
Xue Xia, Saurabh Vishwas Joshi, Kousik Rajesh, Kangnan Li, Yangyi Lu, Nikil Pancha, Dhruvil Deven Badani, Jiajing Xu, and Pong Eksombatchai. 2025. TransAct V2: Lifelong User Action Sequence Modeling on Pinterest Recommendation. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

In the digital age, recommender systems have become indispensable tools for navigating the vast sea of online information. By offering personalized recommendations, these systems not only enhance user experience but also drive engagement and revenue growth for businesses across industries.

Pinterest, a leading content-sharing and social media platform, exemplifies the transformative power of recommendation systems. With billions of Pins and a diverse user base of over 500 million monthly active users [15], Pinterest Homefeed, shown in Figure 1, serves as the primary source of inspiration, providing users with recommendations of high-quality and relevant content. The Homefeed recommendation system uses three stages: retrieval, ranking, and blending. When a user visits the Homefeed, the retrieval stage first retrieves thousands of relevant Pins from billions, based on the user's interests, boards, and other factors. The ranking stage, formulated as a Click-Through Rate (CTR) prediction task, then orders these Pins by predicting their personalized relevance. Finally, the ordered Pins are blended according to business requirements.

While user action history is widely used in CTR models, existing methods often struggle to leverage lifelong user sequences end-to-end for training objectives [14, 25]. Serving cost and latency limitations often restrict models to shorter recent user action sequences ($O(10^2)$), neglecting long-term behavioral patterns. Recent attempts to address lifelong user sequences serving either involve compression (losing information) or rely on expensive offline inference and caching [2, 18]. Furthermore, these models typically

*Work done at Pinterest.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

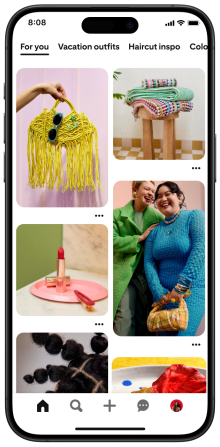

Figure 1: Pinterest Homefeed Page

only encode user actions, lacking the expressive power needed for next-action prediction to improve CTR accuracy.

In this work, we present TransAct V2, a novel model integrating both real-time and lifelong user sequences within the CTR prediction framework. By employing a new next-action prediction task, TransAct V2 enhances the understanding of user preference and improves recommendation diversity. Our system design efficiently handles the full lifelong sequences ($O(10^4)$) with low latency, storage, and network cost. Extensive offline and online experiments validate its effectiveness.

TransAct V2 is now serving production traffic on Pinterest's Homefeed, delivering personalized recommendations to over 500 million users. This deployment not only demonstrates the model's effectiveness in enhencing user engagement but also showcases its practical and scalable approach to solving real-world application challenges. Our major contributions are:

- We introduce TransAct V2, a state-of-the-art model that combines both real-time and lifelong user sequences to improve user engagement and recommendation diversity.
- We implement a novel next-action loss that empowers the CTR model to better predict user behavior, leading to improved predictive accuracy.
- We develop web-scale serving solutions for CTR models, ensuring low-latency performance and efficient resource management, accompanied by comprehensive ablation studies highlighting the contribution of each optimization to overall efficiency.

2 Related Work

2.1 Sequential Recommendation

Sequential recommendation models primarily aim to capture user interests by analyzing past behavior history. These models are divided into two main categories. The first targets direct prediction of the next item in the user's sequence, while the second models user sequences within a CTR framework. Transformer-based architectures have revolutionized sequential recommendation by modeling item dependencies via self-attention [10, 13, 19]. While highly effective for pattern identification, they are typically used for pre-training user representations, not end-to-end learning for CTR prediction, and then applied to downstream tasks like item ranking.

In industrial applications, the recommender system usually uses a CTR prediction model in the ranking stage. Recent research has focused on employing sequential transformer encoders to process user action sequences as part of the CTR model [1, 2, 7, 14, 16, 18, 25, 26] Two-stage user sequence modeling, e.g. SIM [14], UBR4CTR [16], TransAct [25], TWIN [2], retrieves relevant items then applies self-attention. However, they either limit sequence length (restricting long-term history exploration) or rely on expensive offline inference and caching. TWIN v2 [18] attempts to address some challenges by compressing lifelong user action histories using clustering algorithms offline. However, this compression may reduce the sequence's expressive power as critical information can be lost during the compression process. Furthermore, unlike the category of models that can directly predict the next action, these CTR-focused transformers lack direct next-action prediction capabilities.

2.2 Efficient Sequential Model Serving

Recommendation systems are typically high-throughput systems subject to strict latency constraints. Each request involves ranking thousands of candidates using a sequential model, which presents challenges when handling long user action sequences in real time. Consequently, many studies employ offline sequence compression techniques, such as clustering methods like TWIN v2 [18] and Trinity [28], rather than managing long sequences in real time. Offline sequence compression methods are suboptimal since they are agnostic to the candidate being ranked. Using real-time user sequences allows us to model interaction of the user sequence with the candidate, providing the model with richer information for ranking candidates. Notably, TWIN [2] performs compression using a separate model, necessitating real-time model updates. Another recent approach employs generative architectures like HSTU [30], but these typically require models with trillions of parameters to perform well. Generative recommender (GR) models are also more computationally expensive than regular CTR models. Given these considerations, our work concentrates on CTR prediction, as it more effectively addresses the latency and cost constraints typical of industrial-scale applications.

3 Methodology

We first describe the ranking model architecture in Section 3.1, then explore lifelong sequence modeling and Next Action Loss. Finally, Section 3.4 details TransAct V2's serving and logging system.

3.1 Preliminary: Ranking Model

Pinterest's Homefeed ranking model uses a point-wise multi-task learning (MTL) architecture. As shown in Figure 2, the model takes context, creator, item, and user features as input to predict the user interaction probability. Similar to existing CTR models [2, 9, 14, 25, 26], it employs a standard wide and deep architecture [4], encoding various features including user sequences before applying feature interaction layers [11] and MLPs to generate head scores.

Each training sample is denoted by the pair $(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x}$ represents the input feature set, and $\boldsymbol{y} \in \{0,1\}^{|\boldsymbol{H}|}$ is the label vector associated with the action heads in $\boldsymbol{H}$. Each element of $\boldsymbol{y}$

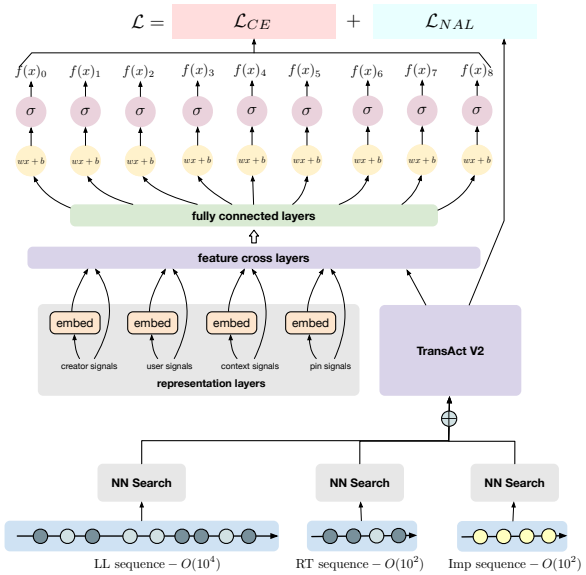

Figure 2: Ranking Model Architecture

corresponds to the label for a respective action head. The core loss function of our ranking model is a weighted cross-entropy loss, tailored for optimizing multi-label classification tasks. We define the multi-head prediction loss function as follows:

$$\mathcal{L}_{CE} = \sum_{h \in H} \{-w_h \left[y_h \log f(\boldsymbol{x})_h + (1 - y_h) \log(1 - f(\boldsymbol{x})_h) \right]\} \quad (1)$$

Here, $f(\boldsymbol{x}) \in (0,1)^H$ are the predicted probabilities, with $f(\boldsymbol{x})_h$ indicating the output probability for head h. Beyond multi-head prediction, we introduce next-action prediction as an auxiliary task, minimizing the Next Action Loss (NAL), detailed in Section 3.3.

3.2 TransAct V2: Transformer for Lifelong User Action Sequence

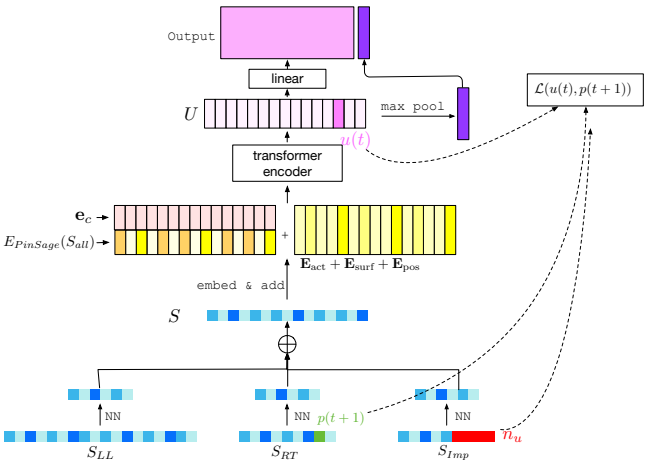

Figure 3: Transact V2 Architecture

In this work, we introduce TransAct V2, a transformer-based model that integrally combines both real-time and lifelong user action sequences within CTR prediction models to enhance user engagement and recommendation diversity. Real-time sequences typically capture short-term user behaviors, focusing on their most recent interests. While this is effective for immediate relevance, it often overlooks users' longer-term historical interests, leading to a narrow focus that lacks diversity. This tendency can result in echo-chamber effects [6, 22], where users are repeatedly exposed to similar content, hindering their ability to discover new interests. Such limitations can adversely affect user retention and diminish the overall long-term user experience [24]. By incorporating lifelong user sequences, our model aims to balance immediate user preferences with historical patterns, fostering a richer and more varied interaction ecosystem. This approach not only improves content diversity but also increases sustained user engagement.

3.2.1 Lifelong User Sequence Features In constructing the lifelong (LL) user sequence S_{LL} for TransAct V2, we focus on capturing meaningful user interactions over an extended period in units of years. The sequence consists of explicit user actions, such as repins[1], clicks, and hides, excluding mere impressions to ensure the relevance of the captured data. The maximum length of the LL sequence is chosen based on the 90th percentile of users' past 2 years of action history lengths weighted by visiting frequency. This weighting is crucial as it reflects the behaviors of our most frequent users who significantly contribute to engagement and revenue. Each lifelong (LL) sequence token has four features: action timestamp, action type (multi-hot vector if multiple interactions with same pin), action surface (e.g., homefeed, search), and 32-d PinSage embedding [29] that encapsulates the content of a pin. Action types can be represented as multi-hot vectors when a user interacts multiple times with the same pin. For example, a user might initially perform a close-up view of a pin and subsequently resave it. Real-time sequences (S_{RT}) and impression sequences S_{imp} use the same features. Among these features, the PinSage embedding is the largest data component. To optimize storage and processing, we apply affine quantization to convert the original 32-dimensional `fp16` PinSage embedding into a 32-dimensional `int8` vector, halving its size:

$$q = \text{clamp}\left(\frac{e}{0.65} \times 127, -127, 127\right) \rightarrow \texttt{int8}$$

The scaling of 0.65 ensures precision and minimizes data loss by normalizing the input distribution.

Table 1: Table of Notations

Notation	Description
S_{LL}, S_{RT}, S_{imp}	Lifelong, Real-time, and impression user sequence
$S_{RT}[:r]$	Most recent r tokens from S_{RT}
S_{all}	Concatenated sequence from all sources
$\mathbf{E}_{act}, \mathbf{E}_{surf}$	Embedding matrix for user action, surface
$\mathbf{E}_{PinSage}(S)$	PinSage embedding matrix of sequence S
$\mathbf{E}_{pos}$	Positional encoding matrix
e_c	PinSage embedding of candidate pin
$u(t)$	User embedding at time t
$p_u(t)$	Positive sample PinSage embedding at t
n_u	Negative samples for user u

[1] "Repin" and "save" both refer to saving a Pin to a board on Pinterest.

3.2.2 Modeling Lifelong User Sequence **Nearest Neighbor Search**. As shown in Figure 3, we process both the real-time and lifelong user sequences using the same transformer encoder. Following an approach similar to TransAct [25], we begin by using the candidate item, denoted as c, as an anchor to perform nearest neighbor (NN) searches on three distinct sequences: the lifelong sequence S_{LL}, the real-time sequence S_{RT}, and the impression sequence S_{imp}. In addition, we always keep the most recent r actions $S_{RT}[:r]$ to ensure that the model consumes user's most fresh actions regardless of the similarity with the candidate item.

The NN search results in sub-sequences from each S_{LL}, S_{RT}, and S_{imp}, which are concatenated along with $S_{RT}[:r]$ as follows:

$$S_{all} = \text{NN}(S_{LL}, c) \oplus S_{RT}[:r] \oplus \text{NN}(S_{RT[r:]}, c) \oplus \text{NN}(S_{imp}, c) \quad (2)$$

where $\text{NN}(S, c)$ is computed by calculating the dot product between the candidate PinSage embedding $\mathbf{e}_c$ and the PinSage embeddings of sequence S, $\mathbf{E}_{\text{PinSage}}(S)$, then selecting the top K elements based on similarity:

$$\text{NN}(S, c) = \{S_i | i \in \text{argsort}(\mathbf{E}_{\text{PinSage}}(S) \cdot \mathbf{e}_c)[-K:]\} \quad (3)$$

It is important to note that S_{LL} has a length of $O(10^4)$, whereas both S_{RT} and S_{imp} are much shorter, with lengths of $O(10^2)$. This final sequence S_{all} is constrained to several hundred elements ($O(10^2)$), ensuring that the computational cost of processing through the transformer encoder remains manageable.

Feature Encoding: To incorporate the metadata of each token in user sequences (action type, surface), we use trainable embedding tables to project them to low-dimensional vectors. The action type sequence is projected to an embedding matrix $\mathbf{E}_{\text{act}} \in \mathbb{R}^{|S| \times d_{act}}$, where d_{act} is the dimension of the action type embedding. Similarly, the surface feature is embedded as $\mathbf{E}_{\text{surf}} \in \mathbb{R}^{|S| \times d_{surf}}$. Positional encoding is implemented as a learnable parameter $\mathbf{E}_{\text{pos}} \in \mathbb{R}^{|S| \times d_{pos}}$, allowing the model to adaptively learn positional information within the sequence. For each token in S_{all}, we concatenate the candidate pin's PinSage embedding $\mathbf{e}_c$ as our early fusion approach [25]. To make all components additive, we set $d = d_{\text{act}} = d_{\text{surf}} = d_{\text{pos}} = 2d_{\text{PinSage}}$. The final encoded user action sequence is formed by adding all the above components.

$$\boldsymbol{F} = \texttt{CONCAT}(\mathbf{E}_{\text{PinSage}}(S_{all}), \mathbf{e}_c) + \mathbf{E}_{\text{act}} + \mathbf{E}_{\text{surf}} + \mathbf{E}_{\text{pos}} \in \mathbb{R}^{|S| \times d} \quad (4)$$

Transformer Encoder: $\boldsymbol{F}$ is processed using a transformer encoder composed of two layers, with each layer utilizing a single attention head. The model dimensionality is set to 64, and the feed-forward network within each layer has a dimension of 32. Note that a causal mask is applied to the transformer encoder. We provide more details about the hyper-parameter choice in 4.4.2. The transformer encoder output $\boldsymbol{U} = (u(0) : u(|S|-1)) \in \mathbb{R}^{|S| \times 2d}$ is used by two downstream tasks. The first one is a multi-head prediction task. As shown in Figure 3, a linear layer and a max pooling layer are applied on top of $\boldsymbol{U}$ to produce the output. The output will be fed into the feature crossing layers in Figure 2 for multi-head prediction. The second task is the next action prediction task as discussed in next section.

3.3 Next Action Loss

We have discussed how the model summarizes user action history. To fully leverage this user sequence information, we designed a separate next-action prediction task, training the transformer to anticipate a user's next action. Our experimental results demonstrate that this auxiliary task alongside multi-head prediction significantly enhances ranking performance.

3.3.1 Next Action Prediction Task Specifically, for a given user's most recent actions from S_{RT}, we predict whether the engagement of the $(t+1)$-th pin is positive or negative using the transformer-processed user embedding at time t. The prediction loss is constructed in a contrastive loss manner. For a given user u, denote the t-th column in the output of the transformer $\boldsymbol{U}$ by $u(t)$, which can also be viewed as the embedding of the user u at time t when causal masking is applied. We use $p_u(t+1)$ to denote the PinSage [29] embeddings for the positive sample at $t+1$, and n_u to denote the negative samples for user u. The specifics of positive/negative sample selection will be discussed in the next section. The next action loss (NAL), denoted as $\mathcal{L}_{NAL}$, employs a sampled softmax (SSM) loss function. SSM is a common technique in contrastive learning, and has been applied in recommendation systems for learning user representations at Pinterest [13].

$$\mathcal{L}(u(t), p(t+1)) = -\log\left(\frac{e^{\langle u(t), p_u(t+1)\rangle}}{e^{\langle u(t), p_u(t+1)\rangle} + \sum_{n_u} e^{\langle u(t), n_u\rangle}}\right), \quad (5)$$

$$\mathcal{L}_{NAL} = \sum_u \sum_t \mathcal{L}(u(t), p_u(t+1)), \quad (6)$$

where N is the number of negative samples and $\langle \cdot, \cdot \rangle$ is for vector inner product. $\mathcal{L}_{NAL}$ is then weighted, and summed with the multi-head cross entropy loss $\mathcal{L}_{CE}$ during training.

$$\mathcal{L} = \mathcal{L}_{CE} + w_{NAL} * \mathcal{L}_{NAL} \quad (7)$$

3.3.2 Key Modeling Design of NAL Unlike standard self-attention, which allows all tokens to attend to each other, a **causal mask** is used to restrict tokens to attend only to preceding ones, thereby preventing information leakage during the next action prediction task. For instance, when predicting the t-th action, the model has access only to actions from 0 to $t-1$.

The choice of loss function for the next action prediction is another important aspect. While cross-entropy is a popular choice for classification, we utilize **sampled softmax loss** because it offers greater flexibility in adjusting the ratio of positive to negative samples in Equation 5, leading to consistently better performance.

In terms of sample selection, for positive samples, we utilize all tokens from $S_{RT}[:r]$ that exhibit positive engagement types. For negative samples, we explored two methods. The first, in-batch random negative sampling, involves selecting a user u_j randomly from the same batch as user u_i and choosing N Pins from u_j's sequence. This assumes that different users within the same batch typically have different engagement interests. The second, impression-based negative sampling, selects Pins from S_{imp} that the current user has viewed but not engaged with further, suggesting low interest. Our findings, detailed in Section 4.4.1, reveal that **impression-based negative samples** are more effective, providing a more challenging set that results in improved ranking performance.

3.4 Serving and Logging System Design

One of the major challenges of utilizing a LL sequence in CTR models lies in the significant **serving** and **storage costs** that come with the extended sequence length. Let L represent the length of S_{LL} and N denote the average number of items per ranking request. Under these conditions:

- The corresponding **feature storage cost** scales as $O(L)$.
- The **network cost** scales as $O(NL)$.

In this work, L is at the scale of $O(10^4)$, introducing substantial challenges in both model serving and network efficiency. Without any optimizations, serving the LL sequence would make the system prohibitively expensive and inefficient. To address both serving latency and storage bottlenecks, we propose a new machine learning data pipeline specifically for super long user sequence models, as illustrated in Figure 4.

These optimizations ensure that our system can handle significantly longer user sequences without compromising latency, storage efficiency, or model performance.

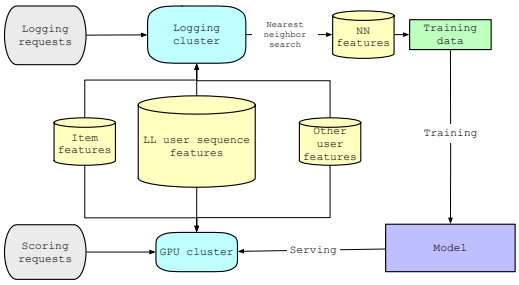

Figure 4: An illustration of the Online Serving System

3.4.1 Data Pipelines All features utilized by the ranking model are usually stored in cache at serving time and subsequently logged into the training data. However, LL sequence features are too large for efficient caching and network transfer. To address the resulting network and storage costs, we implement a **nearest neighbor (NN) feature logging strategy** within the data pipeline, as depicted in Figure 4. Rather than transferring and storing the entire sequence of length L, we perform the NN search detailed in Equation 3 and log only the relevant NN features to the training dataset. This approach reduces data storage complexity from $O(L)$ to $O(1)$, thereby optimizing resource utilization significantly.

Our framework employs distinct data processing pipelines for **training and serving**, optimized for efficiency in both scenarios. During training, the model consumes pre-computed nearest neighbor (NN) features from logged data, significantly reducing data loading overhead. These NN features are passed directly into the transformer encoder. In contrast, during serving, when receiving a ranking request, the system retrieves the full LL sequence of the target user. When the user features and the list of N items are sent to model inference, we broadcast the LL sequence features across all items, followed by an on-device nearest neighbor search to extract relevant NN features. These features are then fed into the transformer encoder for inference. This design balances storage efficiency, computational cost, and alignment between the training and serving stages, ensuring scalability in production environments.

- **Storage Efficiency:** Logging NN features instead of full sequences minimize storage and bandwidth costs.
- **Training-Serving Alignment:** Although the feature processing paths differ, the NN feature representation remains consistent.
- **Inference Scalability:** Real-time NN search on-device ensures scalability and low-latency predictions in industrial deployment environments.

3.4.2 Serving Optimizations To serve LL user sequences in real-time high-QPS systems without incurring huge costs, we jointly optimize the inference server and the model to reduce latency.

Pinterest's recommender models use a custom C++ inference engine and CUDAGraph [12] to reduce the CUDA kernel launch overhead as detailed in TransAct [25]. Requests are dynamically batched into mini-batches, prepared on pageable CPU memory, copied into a pre-allocated pinned memory buffer (per CUDAGraph instance), and finally copied from the pinned memory buffer to static GPU memory locations for the model forward pass.

OpenAI's Triton framework [20, 21] has recently emerged as a powerful tool for writing custom kernels to improve model efficiency. It has been used widely in LLM training as well as recommendation system [8, 23, 27]. Triton allows more fine-grained control over tensor tiling and GPU L2 Cache (Shared Memory or SRAM), enabling smarter caching and in-place fused operations. In our work, we adopt Triton to build custom transformer kernels to optimize model serving. In addition, processing $O(NL)$-sized tensors can lead to substantial CPU-to-GPU and intra-GPU data transfers. By fusing expensive operations into custom kernels, we reduce both the number of kernel calls and the need for intermediate data transfers, thus improving overall serving efficiency.

Request Level De-duplication. In a typical ranking setup, a single inference request ranks N items for one user. Request-level features like user sequences are replicated (broadcasted) to each item on the CPU and then transferred to the GPU for inference. The LL sequence features constitute a significant portion of the total feature size. Instead of broadcasting these request-level sequence features, we introduce a new sparse tensor format that stores de-duplicated request-level features and offsets. As shown in Figure 5, this format supplies all necessary metadata for the model to handle request-level sequences without broadcasting. Furthermore, we developed a custom Triton kernel to conduct an NN search (Eq 3) directly on the de-duplicated requests (broadcast free NN search), eliminating the need to broadcast request-level sequence features on the GPU. This optimization yields an 8x reduction in PCIe data transfers for sequence features.

Fused Sequence Dequantization. As discussed in Section 3.2, the PinSage sequence feature is stored in a quantized int8 format. Before NN search, this tensor must be dequantized to `float16` and normalized. In PyTorch, L2 normalization typically involves launching four separate kernels of squaring, summation, clamping, and division. Each kernel processes this sizable tensor, and at high QPS, these operations can become a performance bottleneck. Our fused implementation combines this with the NN search kernel resulting in a 20% reduction in model latency compared to the native PyTorch implementation.

Single Kernel Unified Transformer (SKUT). Leveraging our model's low dimensionality ($d_{model} = 64$), we developed a novel fused transformer that achieves a forward pass performance that is 6.6 times faster than PyTorch. The transformer is implemented as a single merged custom Triton kernel. The strong latency improvements are achieved mainly by only materializing the QKV sequence tensors on-the-fly during tiled attention. The feed-forward and layer-norm operations are fused immediately after to produce the final output tile.

Our inference server utilizes Nvidia A10G GPUs (8 MB shared memory) to process Transformer inputs. Given the small dimensionality, we can accommodate all transformer weights — the three projection tensors Q, K, V of shape (64, 64) and the feed-forward network weights W_1, W_2^T of shape (64, 32) alongside the input tensor within 6 MB of shared memory. This allows us to fuse all transformer operations - QKV projection, Flash Attention, layer normalization, and the feed-forward network, into a single fused operation directly on SRAM, significantly reducing GPU transfer overhead and kernel calls.

Pinned Memory Arena. Memory Arena is a thread-local, contiguous block of pre-allocated memory, which is reset and reused across subsequent inference requests. Pre-allocation of memory ensures that no tensor memory is allocated for inference on the fly. During inference, each mini-batch of size 128 results in a 64 MB payload. This large payload introduces two copy bottlenecks - Pageable to Pinned memory copy and Pinned CPU memory to GPU copy. The latter is being sufficiently addressed by request-level deduplication. To compensate for the increased memory pinning cost incurred from this new feature, we implemented a Pinned Memory Arena. Our implementation backed the Memory Arena with pinned memory and constructed collated mini-batches directly onto it. This completely eliminated the pageable to pinned copy step and improved inference speed for the final experiment setup by up to 35%.

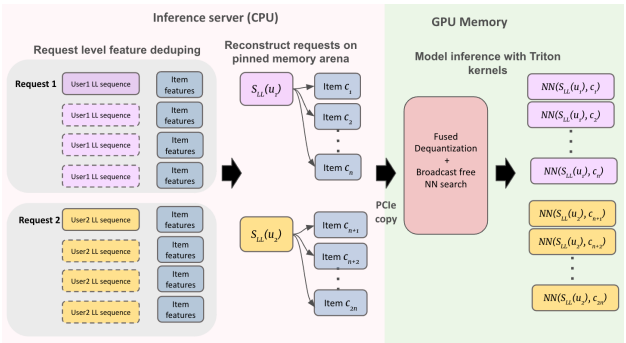

Figure 5: Serving setup with server side optimizations and custom sparse tensor implementation for request level deduplication

4 Experiment

In this section, we will present extensive offline and online A/B experimental results of TransAct V2. We compare the performance of TransAct V2 with baseline models using Pinterest's internal training data.

Table 2: Offline evaluation comparing existing methods with ours. (* statistically insignificant)

Methods	HIT@3/repin	HIT@3/hide
BST (RT sequence) [3]	+6.04%	-0.49%*
TransAct (RT sequence) [25]	+7.74%	-6.86%
TransAct (RT sequence) + $\text{NAL}_{\text{in\_batch}}$	+8.41%	-8.63%
TransAct (RT sequence) + NAL_{imp}	+8.92%	-9.08%
TransAct V2 (RT seq + LL seq + NAL_{imp})	**+13.31%**	**-11.25%**

4.1 Experiment Setup

4.1.1 Dataset Our offline training dataset is gathered and downsampled from two weeks of user activities on Pinterest Homefeed. The training data contains 6.9 billion training instances of 182 million users and 350 million Pins. The model is trained from scratch using randomly initialized weights and evaluated on an evaluation dataset collected 7 days after the last day of training data.

In this paper, we conduct all experiments with the Pinterest dataset. Our experiments require lifelong action sequence, real-time action sequence and impression sequence along with metadata features, such as item embeddings, action types, surface types. To the best of our knowledge, publicly available CTR prediction datasets lack these features, though they are readily reproducible in industry recommender systems.

4.2 Offline Experiment

4.2.1 Metrics Our model performance is evaluated using the HIT@3 metric. A "chunk" of recommendations, denoted as $\boldsymbol{c} = [p_1, p_2, \ldots, p_n]$, represents a set of Pins recommended to a user simultaneously. Each evaluation data instance for the ranking model includes a user identifier u, a pin identifier p, and a chunk identifier c. For evaluation, outputs are grouped by user and chunk identifiers (u, c) to aggregate results from the same ranking request. Within each group, Pins are sorted based on a final ranking score $\mathcal{S}$, a linear combination of the outputs from the ranking model's heads, denoted by $f(\boldsymbol{x})$:

$$\mathcal{S} = \sum_{h \in \mathcal{H}} w_h f(\boldsymbol{x})_h$$

Here, $\mathcal{H}$ represents the set of model heads, and w_h is the weight applied to each head's output. The top K Pins from each chunk are considered, and HIT@K is computed, noted as $\gamma_{c,h}$, which counts the number of top-K Pins bearing a label of 1 for head h. For example, if chunk $\boldsymbol{c} = [p_1, p_2, p_3, \ldots, p_n]$ is sorted by $\mathcal{S}$, and the user repins p_1 and p_4, then HIT@K for repin is $\gamma_{c,\text{repin}} = 1$ when $K = 3$. The aggregated HIT@3 for each head h is calculated over all users U and their corresponding chunks C_u as follows:

$$\text{HIT@3}/h = \frac{\sum_{u \in U} \sum_{c \in C_u} \gamma_{c,h}}{|U|}$$

For positive engagement (e.g., repins, clicks), higher HIT@K is better. For negative engagement like hides, lower HIT@K/hide is preferable.

4.2.2 Results We compare our proposed method with existing models for CTR prediction that utilize user sequence features. We first compare with the Wide and Deep Learning model [4] without incorporating user sequence features. Additionally, we compare our

Table 3: Online Evaluation Metrics compared with TransAct as baseline. (* statistically insignificant)

Online Metrics	NAL_{imp}	TransAct V2
Homefeed Repin Volume	+0.27%*	+6.35%
Homefeed Hide Volume	-6.26%	-12.80%
Impression Diversity	+0.08%*	+0.45%
Time Spent on App	+0.10%*	+1.41%

Note: 1% increase in repin volume is considered as a substantial gain.

model with BST [3] and TransAct [25], which both focus on real-time sequence modeling. TWIN[2] is not compared in this work because it relies on a computationally expensive offline inference system. Models such as BERT4Rec [19] are omitted from direct comparison due to differences in problem formulation.

As presented in Table 2, TransAct V2, which incorporates both RT and LL sequences along with impression-based NAL, achieves superior performance, outperforming all other approaches. It improves HIT@3/repin by 13.31% and reduces HIT@3/hide by 11.25%, demonstrating its ability to enhance positive user engagement and minimize negative interactions. Impression-based Negative Sampling (NAL_{imp}) is crucial to this improvement. By incorporating these impression-based samples, the model benefits from a more nuanced understanding of user disinterest, leading to better calibration of the ranking model. As a result, our experiments confirm that NAL_{imp} along with the lifelong user sequence delivers significant improvements in ranking effectiveness.

4.3 Online Experiment

In the online experiment, we deployed the models in offline experiments. Each group served 1.5% of the Homefeed page visitors. To protect user experience, we used a strong baseline — **TransAct (RT sequence)** from Table 2.

4.3.1 Metrics Key Homefeed metrics include **Homefeed Repin Volume** (higher is better, correlating with HIT@3/repin offline) and **Homefeed Hide Volume** (lower is better, indicating irrelevant recommendations and poor user experience). We also consider **Impression Diversity** (higher is better, reflecting broader content exposure).

4.3.2 Online Results Table 3 shows the online performance of our model. We observe that the NAL_{imp} performs particularly well in reducing the Homefeed Hide Volume by 6.26%. This indicates that using impression-based negative sampling effectively filters out irrelevant recommendations, leading to a better user experience by presenting more pertinent content. TransAct V2 integrated with NAL_{imp} achieves the best overall performance. Specifically, it led to a significant increase of 6.35% in Homefeed Repin Volume, suggesting that the model provides highly relevant recommendations to users. Additionally, it reduces Homefeed Hide Volume by 12.80%, further demonstrating the model's capability in curating content that users find desirable. Moreover, the model also improves Impression Diversity, enhancing the variety of content presented to users and enriching their browsing experience.

Table 4: Offline evaluation of NAL negative sample selection.

Negative sample selection	HIT@3/repin	HIT@3/hide
In-batch neg samples	+0.63%	-1.90%
Impression-based neg samples	**+1.10%**	**-2.39%**

4.4 Ablation Study

We first evaluate design choices for Next Action Loss and the transformer encoder. Then, we examine different components of serving optimizations.

4.4.1 Next Action Loss Ablation We present the ablation study results for NAL design choices. Impression-based negative sampling significantly outperforms in-batch sampling as shown in Table 4. This is because impressions provide stronger and more challenging negatives, improving model generalization and, crucially, enhancing model accuracy in predicting negative outcomes and reducing hides. By training the transformer to recognize and respond to more nuanced negative signals, the model achieves a more comprehensive understanding of user preferences. Additional NAL ablations are provided in appendix.

4.4.2 Transformer Hyperparameters We conducted extensive tuning of transformer hyperparameters, specifically focusing on the feed-forward dimension and input sequence length $|S_{all}|$. Our primary goal is to determine the configuration that would deliver optimal performance, balancing both repin metrics and inference costs/latency. Comprehensive evaluation identified the optimal configuration: sequence length 192, 2 layers and feed-forward dimension 32. This balanced strong repin metrics with low inference latency. Figure 6 shows that increasing sequence length and feed-forward dimension improves HIT@3/repin but significantly increases inference latency.

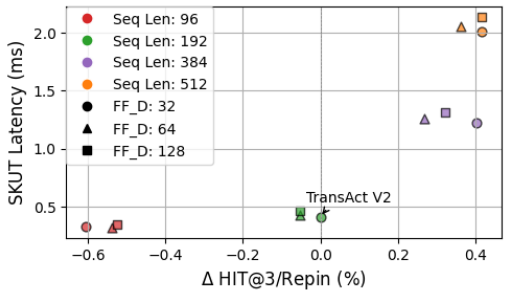

Figure 6: Model Performance and inference latency trade-offs on different transformer hyper-parameter settings

4.4.3 Serving Optimization Ablation **Single Kernel Unified Transformer** - SKUT performance was evaluated by measuring forward latency across sequence lengths 64-2048. Using PyTorch's memory-efficient implementation [17] as the baseline, SKUT achieved 85.09% lower latency and 13.24% less GPU memory utilization in our production setting (Batch size 256, Sequence length 192). Table 5 shows SKUT's advantage over baseline implementation across different sequence length. SKUT also outperforms baseline across different batch sizes (see Appendix B for details). These gains primarily come from avoiding Q, K, V tensor materialization. The peak latency reduction occurs at a sequence length of 192 due to kernel tuning for

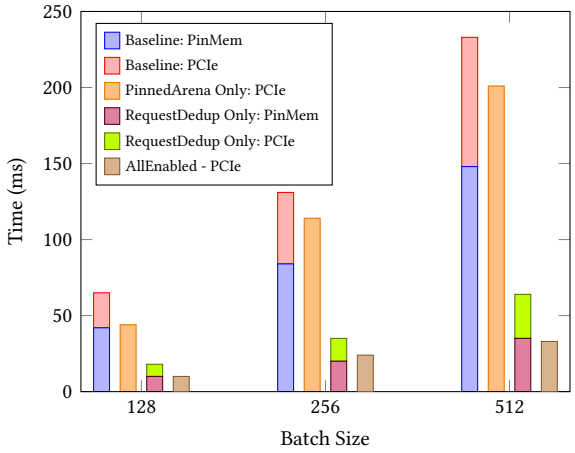

Figure 7: Copy Time components by batch size and configurations at P99 latency.

production input shapes. SKUT also maintains its latency advantage over PyTorch's implementation with for longer sequences. It is noteworthy that FlashAttention-2 [5] was not directly compared because it lacks support for the custom masks required for masking tokens for NAL and padding. However, we still provide comparison without masks in AppendixB.

Table 5: Latency and Memory Reduction (SKUT vs. PyTorch)

Seq Length	Latency Decrease	Memory Decrease
64	-59.29%	-19.60%
128	-81.58%	-10.06%
192	**-85.09%**	**-13.24%**
256	-84.79%	-15.42%
512	-82.27%	-18.12%
1024	-74.85%	-15.22%
2048	-73.10%	-9.80%

Server Optimizations. Results in Figures 7, 8, and 9 are from a high-throughput online serving environment (29,000 examples per second per host), designed to mimic peak production traffic. This setup includes parallel inference streams, dynamic batching, batch queuing, and request de-duplication to accurately reflect online latency reductions. We use p99 latency (the latency within which 99% of requests are completed over a 1-minute window) as our evaluation metric. Real-time, low-latency systems commonly monitor p99 latency to account for worst-case scenarios while excluding outliers.

Figure 7 shows the impact of our optimizations (Section 3.4.2) on the CPU-to-GPU copy time. PinMem denotes the latency incurred from copying to Pinned Memory, and PCIe refers to copy latency over PCIe from PinMem to the GPU. Results are shown for batch sizes of 128 - 512. We report the ablated data for baseline, pinned memory arena only, request de-dup only, and all optimizations combined, making up the 4 bars reported per batch size. Since the pinned memory arena only and all enabled groups directly construct batches on Pinned memory, the PinMem component is zero. With all optimizations, the copy time improves by 85% for batch size 128, 82% for 256 and 85% for 512 compared to the baseline.

Figure 8 shows model inference latency improvements. Model Run latency comprises CPU-to-GPU feature copy, model forward pass, and GPU-to-CPU output copy. Results are shown for batch sizes 128, 256, and 512, comparing baseline, pinned memory, request deduplication, and the combined approach. Pinned Memory Arena, when tested independently, has neutral to slightly positive latency reduction results. This is because, on enabling the pinned memory arena exclusively, the setup becomes PCIe bandwidth bound at high throughput. However, Request de-dup only shows consistent reductions across all latencies and batch sizes. The best reduction in Model Run latency is achieved by stacking both optimizations together, resulting in a reduction of 75%, 75%, and 81% for batch sizes 128, 256, 512, respectively, compared to the baseline.

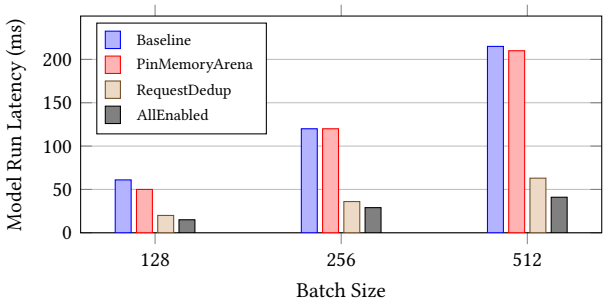

Figure 8: P99 Latency comparison across techniques for various batch sizes.

Model Forward	2.3×	2.2×	2.2×
Pin Memory	46×	31×	37.5×
PCIe Copy	2.5×	1.8×	1.8×
Batch Preparation	5.7×	4.9×	4.7×
Queuing Delay	1,360×	1,306×	1,267×
E2E Inference Latency	103×	338×	250×
	p50	p90	p99

Figure 9: Heatmap of Reduction Factors (Baseline vs. Final)

Figure 9 shows the improvement ratio of our final optimized setup versus the baseline, across p50, p90, and p99 latency percentiles. The y-axis breaks down end-to-end RPC latency: model forward, pin memory, and PCIe copy time, batch preparation time (CPU tensor construction) and queuing delay (dynamic batching wait time). As expected, model forward, pin memory, and PCIe copy latencies improve, in line with the previous results in Figures 7 and 8. Furthermore, request de-dup reduces processed bytes, speeding up batch preparation by a factor of 5x. Faster mini-batch processing reduces queuing delay by 1300x. Overall, end-to-end inference latency improves by a factor of 103x, 338x and 250x for p50, p90, p99 respectively.

5 Conclusions

This paper introduced TransAct V2, a novel model that integrates real-time and lifelong user sequences to enhance CTR predictions. Leveraging next-action loss, TransAct V2 improves action prediction, boosting engagement and model expressiveness. Its deployment in real-world applications at Pinterest showcases its capability to efficiently manage extensive user histories, tackling latency and storage challenges. Rigorous ablation of serving optimizations demonstrated reduced latency and storage costs, with A/B testing confirming significant improvements in engagement metrics. TransAct V2 advances sequential recommendation, providing a robust framework for deploying complex models at scale.

References

[1] Fedor Borisyuk, Mingzhou Zhou, Qingquan Song, Siyu Zhu, Birjodh Tiwana, Ganesh Parameswaran, Siddharth Dangi, Lars Hertel, Qiang Charles Xiao, Xiaochen Hou, Yunbo Ouyang, Aman Gupta, Sheallika Singh, Dan Liu, Hailing Cheng, Lei Le, Jonathan Hung, Sathiya Keerthi, Ruoyan Wang, Fengyu Zhang, Mohit Kothari, Chen Zhu, Daqi Sun, Yun Dai, Xun Luan, Sirou Zhu, Zhiwei Wang, Neil Daftary, Qianqi Shen, Chengming Jiang, Haichao Wei, Maneesh Varshney, Amol Ghoting, and Souvik Ghosh. 2024. LiRank: Industrial Large Scale Ranking Models at LinkedIn. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Barcelona, Spain) *(KDD '24)*. Association for Computing Machinery, New York, NY, USA, 4804–4815. https://doi.org/10.1145/3637528.3671561

[2] Jianxin Chang, Chenbin Zhang, Zhiyi Fu, Xiaoxue Zang, Lin Guan, Jing Lu, Yiqun Hui, Dewei Leng, Yanan Niu, Yang Song, and Kun Gai. 2023. 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* (Long Beach, CA, USA) *(KDD '23)*. Association for Computing Machinery, New York, NY, USA, 3785–3794. https://doi.org/10.1145/3580305.3599922

[3] Qiwei Chen, Huan Zhao, Wei Li, Pipei Huang, and Wenwu Ou. 2019. Behavior Sequence Transformer for E-Commerce Recommendation in Alibaba. In *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data* (Anchorage, Alaska) *(DLP-KDD '19)*. Association for Computing Machinery, New York, NY, USA, Article 12, 4 pages. https://doi.org/10.1145/3326937.3341261

[4] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.

[5] Tri Dao. 2023. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. arXiv:2307.08691 [cs.LG] https://arxiv.org/abs/2307.08691

[6] Yingqiang Ge, Shuya Zhao, Honglu Zhou, Changhua Pei, Fei Sun, Wenwu Ou, and Yongfeng Zhang. 2020. Understanding Echo Chambers in E-commerce Recommender Systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20)*. ACM, 2261–2270. https://doi.org/10.1145/3397271.3401431

[7] Lars Hertel, Neil Daftary, Fedor Borisyuk, Aman Gupta, and Rahul Mazumder. 2024. Efficient user history modeling with amortized inference for deep learning recommendation models. arXiv:2412.06924 [cs.LG] https://arxiv.org/abs/2412.06924

[8] Pin-Lun Hsu, Yun Dai, Vignesh Kothapalli, Qingquan Song, Shao Tang, Siyu Zhu, Steven Shimizu, Shivam Sahni, Haowen Ning, and Yanning Chen. 2025. Liger Kernel: Efficient Triton Kernels for LLM Training. arXiv:2410.10989 [cs.LG] https://arxiv.org/abs/2410.10989

[9] Yi-Ping Hsu, Po-Wei Wang, Chantat Eksombatchai, and Jiajing Xu. 2024. Taming the One-Epoch Phenomenon in Online Recommendation System by Two-stage Contrastive ID Pre-training. In *Proceedings of the 18th ACM Conference on Recommender Systems* (Bari, Italy) *(RecSys '24)*. Association for Computing Machinery, New York, NY, USA, 838–840. https://doi.org/10.1145/3640457.3688053

[10] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)*. IEEE, 197–206.

[11] Siddarth Malreddy, Matthew Lawhon, Usha Amrutha Nookala, Aditya Mantha, and Dhruvil Deven Badani. 2024. Improving feature interactions at Pinterest under industry constraints. *arXiv preprint arXiv:2412.01985* (2024).

[12] Vinh Nguyen, Michael Carilli, Sukru Burc Eryilmaz, Vartika Singh, Michelle Lin, Natalia Gimelshein, Alban Desmaison, and Edward Yang. 2021. Accelerating PyTorch with CUDA Graphs. (2021). https://pytorch.org/blog/accelerating-pytorch-with-cuda-graphs/

[13] Nikil Pancha, Andrew Zhai, Jure Leskovec, and Charles Rosenberg. 2022. PinnerFormer: Sequence Modeling for User Representation at Pinterest. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Washington DC, USA) *(KDD '22)*. Association for Computing Machinery, New York, NY, USA, 3702–3712. https://doi.org/10.1145/3534678.3539156

[14] Qi Pi, Xiaoqiang Zhu, Guorui Zhou, Yujing Zhang, Zhe Wang, Lejian Ren, Ying Fan, and Kun Gai. 2020. Search-based User Interest Modeling with Lifelong Sequential Behavior Data for Click-Through Rate Prediction. *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (2020). https://api.semanticscholar.org/CorpusID:219558850

[15] Pinterest. 2024. Third Quarter 2024 Earnings Conference Call. (2024). https://investor.pinterestinc.com/news-and-events/events-and-presentations/event-details/2024/Third-Quarter-2024-Earnings-Conference-Call/default.aspx

[16] Jiarui Qin, Weinan Zhang, Xin Wu, Jiarui Jin, Yuchen Fang, and Yong Yu. 2020. User Behavior Retrieval for Click-Through Rate Prediction. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20)*. ACM, 2347–2356. https://doi.org/10.1145/3397271.3401440

[17] Markus N. Rabe and Charles Staats. 2022. Self-attention Does Not Need $O(n^2)$ Memory. arXiv:2112.05682 [cs.LG] https://arxiv.org/abs/2112.05682

[18] Zihua Si, Lin Guan, Zhongxiang Sun, Xiaoxue Zang, Jing Lu, Yiqun Hui, Xingchao Cao, Zeyu Yang, Yichen Zheng, Dewei Leng, Kai Zheng, Chenbin Zhang, Yanan Niu, Yang Song, and Kun Gai. 2024. 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 (CIKM '24)*. ACM, 4890–4897. https://doi.org/10.1145/3627673.3680030

[19] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. *CoRR* abs/1904.06690 (2019). arXiv:1904.06690 http://arxiv.org/abs/1904.06690

[20] Philippe Tillet. 2021. Introducing Triton: Open-source GPU programming for neural networks. (2021). https://openai.com/index/triton/

[21] Philippe Tillet, H. T. Kung, and David Cox. 2019. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages* (Phoenix, AZ, USA) *(MAPL 2019)*. Association for Computing Machinery, New York, NY, USA, 10–19. https://doi.org/10.1145/3315508.3329973

[22] Yuyan Wang, Cheenar Banerjee, Samer Chucri, Fabio Soldo, Sriraj Badam, Ed H. Chi, and Minmin Chen. 2024. Beyond Item Dissimilarities: Diversifying by Intent in Recommender Systems. arXiv:2405.12327 [cs.IR] https://arxiv.org/abs/2405.12327

[23] Erik Wijmans, Brody Huval, Alexander Hertzberg, Vladlen Koltun, and Philipp Krähenbühl. 2024. Cut Your Losses in Large-Vocabulary Language Models. arXiv:2411.09009 [cs.LG] https://arxiv.org/abs/2411.09009

[24] Mark Wilhelm, Ajith Ramanathan, Alexander Bonomo, Sagar Jain, Ed H. Chi, and Jennifer Gillenwater. 2018. Practical Diversified Recommendations on YouTube with Determinantal Point Processes. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (Torino, Italy) *(CIKM '18)*. Association for Computing Machinery, New York, NY, USA, 2165–2173. https://doi.org/10.1145/3269206.3272018

[25] Xue Xia, Pong Eksombatchai, Nikil Pancha, Dhruvil Deven Badani, Po-Wei Wang, Neng Gu, Saurabh Vishwas Joshi, Nazanin Farahpour, Zhiyuan Zhang, and Andrew Zhai. 2023. TransAct: Transformer-based Realtime User Action Model for Recommendation at Pinterest. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Long Beach, CA, USA) *(KDD '23)*. Association for Computing Machinery, New York, NY, USA, 5249–5259. https://doi.org/10.1145/3580305.3599918

[26] Jiajing Xu, Andrew Zhai, and Charles Rosenberg. 2022. Rethinking Personalized Ranking at Pinterest: An End-to-End Approach. In *Proceedings of the 16th ACM Conference on Recommender Systems* (Seattle, WA, USA) *(RecSys '22)*. Association for Computing Machinery, New York, NY, USA, 502–505. https://doi.org/10.1145/3523227.3547394

[27] Rengan Xu, Junjie Yang, Yifan Xu, Hong Li, Xing Liu, Devashish Shankar, Haoci Zhang, Meng Liu, Boyang Li, Yuxi Hu, Mingwei Tang, Zehua Zhang, Tunhou Zhang, Dai Li, Sijia Chen, Gian-Paolo Musumeci, Jiaqi Zhai, Bill Zhu, Hong Yan, and Srihari Reddy. 2024. Enhancing Performance and Scalability of Large-Scale Recommendation Systems with Jagged Flash Attention. In *18th ACM Conference on Recommender Systems (RecSys '24)*. ACM, 778–780. https://doi.org/10.1145/3640457.3688040

[28] Jing Yan, Liu Jiang, Jianfei Cui, Zhichen Zhao, Xingyan Bin, Feng Zhang, and Zuotao Liu. 2024. Trinity: Syncretizing Multi-/Long-tail/Long-term Interests All in One. arXiv:2402.02842 [cs.IR] https://arxiv.org/abs/2402.02842

[29] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (London, United Kingdom) *(KDD '18)*. Association for Computing Machinery, New York, NY, USA, 974–983. https://doi.org/10.1145/3219819.3219890

[30] Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Michael He, Yinghai Lu, and Yu Shi. 2024. Actions Speak Louder than Words: Trillion-Parameter Sequential Transducers for Generative Recommendations. arXiv:2402.17152 [cs.LG] https://arxiv.org/abs/2402.17152

A Additional Ablation Studies On NAL

A.1 NAL Loss Weight

We tuned loss weight w_{NAL} in Eq 7. As shown in Table 6, adjusting the NAL loss weight is necessary for maintaining a balance between the Next Action Loss task and the multi-head prediction task. This equilibrium is essential because the next action prediction serves as an auxiliary task, whereas multi-head prediction is the primary objective of the ranking model. If w_{NAL} is too large, the model's performance deteriorates on multi-head prediction due

Table 8: Latency and Memory Reduction by Batch Size (SKUT vs. PyTorch [17])

Batch Size	Latency Decrease	Memory Decrease
16	-60.36%	-1.34%
32	-65.96%	-3.20%
64	-76.79%	-4.45%
128	-82.79%	-8.80%
256	**-85.09%**	**-13.24%**
512	-86.69%	-23.01%
1024	-87.28%	-31.47%
2048	-87.79%	-38.55%

Table 6: Offline evaluation of NAL loss weight tuning (impression-based negatives). (* statistically insignificant)

NAL loss weight	HIT@3/repin	HIT@3/hide
0.0001	+0.58%	-2.09%
0.001	+0.86%	**-2.6%**
0.01	**+1.10%**	-2.39%
0.1	+0.54%	-2.39%

Table 7: Offline evaluation of NAL loss types. (* statistically insignificant)

NAL loss type	HIT@3/repin	HIT@3/hide
cross entropy	**-1.31%**	**0.27%***
sampled softmax	**0.18%**	**-1.84%**

to the disproportionate influence of NAL. Conversely, if w_{NAL} is too small, the contribution of NAL becomes negligible. Based on our ablation study, we determined that setting $w_{NAL} = 0.01$ is optimal for our model. However, it is worth noting that the ideal weight may vary across different models and datasets, as it needs to accommodate the diverse magnitudes of loss values encountered in various applications.

A.2 NAL Loss Type

Table 7 shows that the sampled softmax loss outperforms the cross-entropy loss. Cross-entropy loss is typically the most popular choice for classification tasks, such as predicting positive or negative actions in our setup. However, sampled softmax loss offers greater flexibility by allowing fine-tuning of the positive-to-negative sample size ratio, as illustrated in Equation 5. This additional freedom contributes to consistently enhanced performance.

As demonstrated in Table 7, the sampled softmax loss leads to superior outcomes compared to cross-entropy loss. This advantage underscores the efficacy of adjusting sampling ratios to achieve improved classification results.

B Additional Ablation Studies on SKUT

The comparison presented in Table 8 underscores SKUT's superiority over PyTorch's memory-efficient implementation across various batch sizes, with a fixed sequence length of 192. Although FlashAttention-2 is not directly applicable to our work due to its lack of support for custom masking, we conducted a performance comparison with SKUT in the absence of masking. Our SKUT implementation demonstrates a substantial advantage, achieving 66.4% lower latency and 5.5% reduced memory usage compared to a FlashAttention-2 transformer without attention masks.