DV365: Extremely Long User History Modeling at Instagram

Wenhan Lyu
wenhanl@meta.com
Meta Platforms, Inc.
Menlo Park, USA

Devashish Tyagi†
devashisht@openai.com
OpenAI
San Francisco, USA

Yihang Yang
yihangyang@meta.com
Meta Platforms, Inc.
New York, USA

Ziwei Li
ziweili@meta.com
Meta Platforms, Inc.
Menlo Park, USA

Ajay Somani†
ajay250334@gmail.com
Independent
New York, USA

Karthikeyan Shanmugasundaram
karthikss@meta.com
Meta Platforms, Inc.
New York, USA

Nikola Andrejevic
hmnikola@meta.com
Meta Platforms, Inc.
New York, USA

Ferdi Adeputra
fadeputr@meta.com
Meta Platforms, Inc.
New York, USA

Curtis Zeng
curtiszeng@meta.com
Meta Platforms, Inc.
New York, USA

Arun K. Singh
arunksingh@meta.com
Meta Platforms, Inc.
Menlo Park, USA

Maxime Ransan
maxransan@meta.com
Meta Platforms, Inc.
New York, USA

Sagar Jain†*
jnsagar08@gmail.com
Independent
Menlo Park, USA

Abstract

Long user history is highly valuable signal for recommendation systems, but effectively incorporating it often comes with high cost in terms of data center power consumption and GPU. In this work, we chose offline embedding over end-to-end sequence length optimization methods to enable extremely long user sequence modeling as a cost-effective solution, and propose a new user embedding learning strategy, multi-slicing and summarization, that generates highly generalizable user representation of user's long-term stable interest. History length we encoded in this embedding is up to 70,000 and on average 40,000. This embedding, named as DV365, is proven highly incremental on top of advanced attentive user sequence models deployed in Instagram. Produced by a single upstream foundational model, it is launched in 15 different models across Instagram and Threads with significant impact, and has been production battle-proven for >1 year since our first launch.

CCS Concepts

• **Information systems → Personalization**; **Recommender systems**.

Keywords

User modeling, Representation Learning, User Embedding, Long User Sequence Modeling, Recommender System

*† Work done while at Meta.

ACM Reference Format:
Wenhan Lyu, Devashish Tyagi†, Yihang Yang, Ziwei Li, Ajay Somani†, Karthikeyan Shanmugasundaram, Nikola Andrejevic, Ferdi Adeputra, Curtis Zeng, Arun K. Singh, Maxime Ransan, and Sagar Jain†. 2025. DV365: Extremely Long User History Modeling at Instagram. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3711896.3737209

1 Introduction

Large-scale recommendation systems are evolving fast driven by strong business needs. The most business critical recommendation models have been scaled for years and are often expensive large models. These models are resource-intensive during both training and deployment, including support systems such as feature extraction. Organizations are all resource-constrained, not only by capacity budgets but also by limited GPU supply in the market. This pushes us to scale models cautiously with ROI (return on investment) in mind.

Like previously found by Alibaba [10][12] and Kuaishou [2], we also found scaling user history length a compelling direction of improving recommendation quality. Currently in Instagram, recommendation models can access a user engagement history of up to 2k in length, bottlenecked by feature storage, extraction and processing costs. In our most advanced models, a user history of up to 500 in length is encoded by strong but costly attentive sequence encoder, HSTU [16], which is bottlenecked by model training and serving GPU costs.

Directly scaling up sequence lengths in existing models is not an option since we already reached our ROI limit after years of iterations, and attentive sequence models like HSTU have well-known challenges on sequence length scaling. We have 2 choices on the table: 1) End-to-end (E2E) model redesign with sequence length optimization methods [1] [2] [11] [10] [12]; 2) Pre-train long user history into user embeddings and knowledge transfer to downstream models as features.

We decided to take the offline embedding approach for the following reasons:

(1) **Unlock Ambitious Length Scaling**: We target to unlock as much recommendation quality win as possible and we ended up scaling the combined sequence length to 70K in maximum and 40K on average.
(2) **Stable Interest Hypothesis**: We hypothesize that user interests consist of emerging interest and stable interest. The main incremental value of the long user history is in stable user interest, which does not change rapidly over time and needs a lower cadence of recurring training than production models with online training enabled.
(3) **Cross-domain Knowledge Sharing**: In Instagram, we have 20+ models serving different product surfaces like Reels, Feed, Explore, Stories, and across recommendation stages, including Retrieval, Early stage ranking (ESR), Late stage ranking (LSR), which is a common situation for many organizations. Doing the expensive long history computation only once and sharing with all models is a critical ROI attribute.
(4) **Feature Infra Costs Saving**: Real-time retrieving and processing extremely long sequences in every recommendation request, which is required by E2E methods, is a high cost and can be saved with offline embedding approach.

In this paper, we introduce an offline user embedding approach, with code name **DV365**, that uses a **Multi-slicing and Summarize (MSS)** strategy to encode user's extremely long history into a user embedding. Considering the scale we are handling, we don't apply attentive sequence modeling in the long sequence, but use multi-slicing to divide the long sequence into a set of sub-sequences and apply pooling. Multi-slicing and pooling produces 200 pooled embeddings. We also design a Funnel Summarization Arch as a user encoder to encode them into a condensed embedding. The user encoder module is embedded in a backbone simulation network similar to Instagram's production LSR model to simulate the E2E user encoder joint learning to help the user encoder produce a more consistent embedding for production models.

Our upstream model is trained and published in recurring manner to consume new training data, and store in a key-value store with 3 billion user to embedding pairs for downstream models.

All offline embedding approaches has the following limitations, but we proved in this work that DV365 is a viable solution:

(1) **Freshness:** Model freshness is critical for recommendation quality [4] to timely capture emerging data distribution. In Instagram, most critical models enabled online training [13] to maximize freshness. Compared to E2E learning, offline embedding inevitably introduces delays between when the user history was encoded and when the signal is used in making recommendation decisions. To mitigate this, we design our embedding model objective to focus on "stable interest" which doesn't change much over time and is robust to low freshness, and verified empirically that DV365 embedding quality has negligible regression as model delay increases.
(2) **Knowledge Transfer Efficiency and Generalizability:** Compared to user encoder trained E2E directly in production models, knowledge transferred from offline models inevitably has knowledge loss. Since our goal is to use a single foundational model to produce embedding for serving multiple downstream models, the knowledge loss casts a challenge on generalizability. We tackle this problem by training the embedding in a generic way with cross-surface data, and add adaptation modules in downstream models for domain adaptation. We prove the feasibility of this method using 15 downstream model adoptions with significant quality improvement.

Our contributions can be summarized as follows. First, we introduce a multi-slicing and summarize user embedding training strategy that produces embeddings that give significant incremental values to production models with state-of-the-art (SOTA) user modeling components such as HSTU. We prove offline embedding as a high ROI solution for recommendation systems that want to unlock knowledge from extremely long user history. Second, we have deployed DV365 embeddings to 15 production models in Instagram, Threads with a single embedding offering with significant impact. This solution is time- and production-proven since our first launch in November 2023.

2 Method

As shown in Figure 1, the objective of our system is to build an offline foundational model that effectively pre-computes long user sequence information into a shareable user embedding to elevate the performance of a fleet of downstream models, agnostic of downstream model objectives and architectures.

Step 1: The upstream foundation model is trained and published recurringly, so that the user embedding is updated every few hours. The foundation model has 2 components:

(1) **User encoder**: We use multi-slicing and summarization strategy in which we preprocess raw user interaction history into many sliced sub-sequences as categorical features, and summarize them into a compact user representation with a neural network. The output of this module becomes the user embedding used by downstream models.
(2) **Backbone Simulation Network**: To simulate the downstream use case and align user encoder training with recommendation model objectives, user encoder is embedded in a backbone simulation network with regular ranking objectives and other ranking features.

Step 2: Downstream models incorporate the embedding with light-weight domain adaptation modules.

2.1 Data Model and Preprocessing

We design a unified user timeline (UniTi) which models user history as unified sequences of structured user interactions. Each user interaction contains fields of engaged media ID, author ID, event timestamp, action type (like, share, comment, etc.), video duration, user watch time, surface type, and media type.

The UniTi format can be described as follows. Let $\mathbf{m} = (e_1, e_2, e_3, \ldots)$ be a vector representing a media item, where element represents one media attribute such as the media id, author id, topic id, and

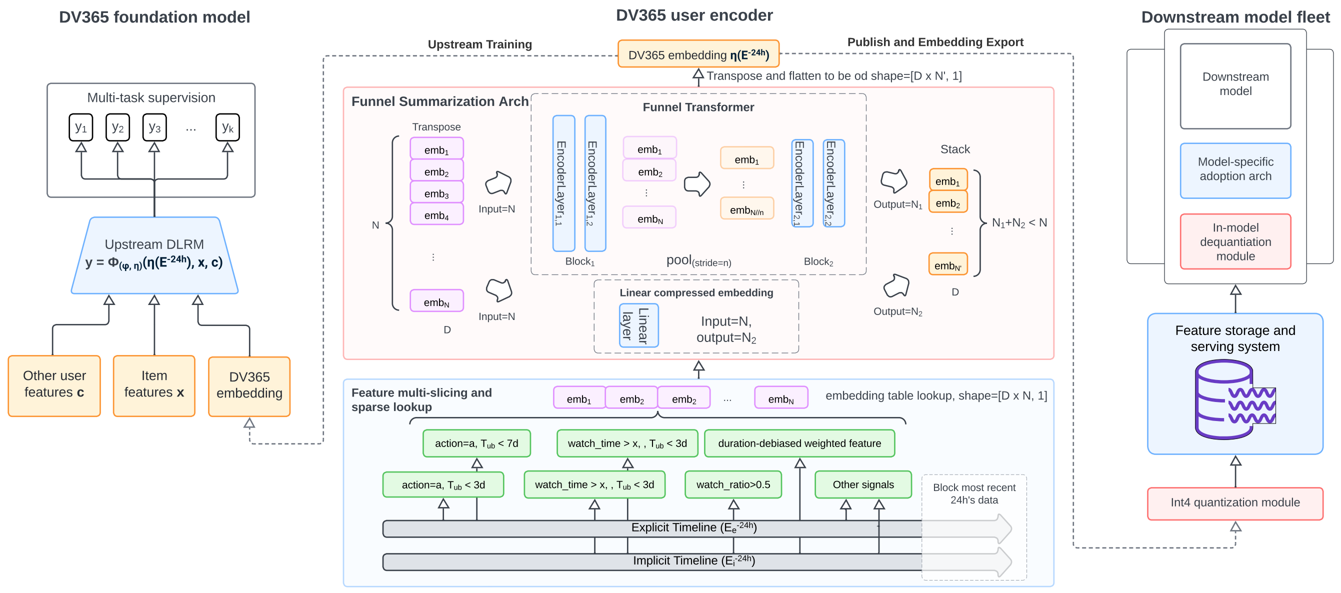
DV365 foundation model
Multi-task supervision
y1 y2 y3 ... yk
Upstream DLRM
y = Φ(φ, η)(η(E-24h), x, c)
Other user features c
Item features x
DV365 embedding
DV365 user encoder
Upstream Training
DV365 embedding η(E-24h)
Publish and Embedding Export
Transpose and flatten to be od shape=[D x N', 1]
Funnel Summarization Arch
Funnel Transformer
Transpose
emb1 emb2 emb3 emb4 ... embN
N
D
Input=N
EncoderLayer1,1
EncoderLayer1,2
Block1
pool(stride=n)
embN/m
EncoderLayer2,1
EncoderLayer2,2
Block2
Output=N1
Output=N2
Stack
embN'
N1+N2 < N
Linear compressed embedding
Linear layer
Input=N, output=N2
Feature multi-slicing and sparse lookup
embedding table lookup, shape=[D x N, 1]
action=a, Tub < 7d
watch_time > x, , Tub < 3d
duration-debiased weighted feature
action=a, Tub < 3d
watch_time > x, , Tub < 3d
watch_ratio>0.5
Other signals
Block most recent 24h's data
Explicit Timeline (Ee-24h)
Implicit Timeline (Ei-24h)
Downstream model fleet
Downstream model
Model-specific adoption arch
In-model dequantiation module
Feature storage and serving system
Int4 quantization module

Figure 1: Architecture of model training and embedding exporting.

media type. Based on user's action, we define two types of user timelines:

$$E_e = [\ldots, (\mathbf{m}_i, \text{action\_type}_i, t_i), \ldots], i = 0, 1, \ldots$$
$$E_i = [\ldots, (\mathbf{m}_j, \text{video\_duration}_j, \text{dwell\_time}_j, t_j), \ldots], j = 0, 1, \ldots \quad (1)$$

where E_e is the **explicit timeline** (explicit user actions such as like, share), E_i is the **implicit timeline** (user impression history of any user dwell time), and i and j are the respective index of the user engagements from a sampled data logger.

Organizing user features as raw timelines ensures flexibility in feature engineering. Machine learning engineers can select any filtering criteria at the in-model preprocessing stage to create features on the fly.

2.2 Feature engineering

We design UniTi features similarly to making rule-based sequence aggregation, in contrast to learnable aggregation used in sequence modeling methods like HSTU which is much more expensive.

For the explicit timeline, we simply slice features by action types

$$f_{\text{action}=a}(E_e) = \text{select } E_e \text{ if action\_type} = a, \quad (2)$$

so that a user's liked and commented media will have separate sequences. Similarly, user engagement in different logging sources (video or photo) will also be in different sub-sequences.

For the implicit timeline, we create sub-sequence features based on users' dwell time and the engaged videos' duration. We define 2 criteria based on dwell time and watch percentage:

$$f_{\text{watch\_time}}(E_i) = \text{select } E_i \text{ if dwell\_time} \in [T_{\text{lb}}, T_{\text{ub}}]; \quad (3)$$

$$f_{\text{watch\_ratio}}(E_i) = \text{select } E_i \text{ if } \frac{\text{dwell\_time}}{(\text{video\_duration})^{\beta}} \in [\alpha_{\text{lb}}, \alpha_{\text{ub}}], \quad (4)$$

where $T_{\text{lb}}, T_{\text{ub}}, \alpha_{\text{lb}}, \alpha_{\text{ub}}$ are the lower and upper bounds of the watch time/ratio criteria, and β is a tunable exponent. Having $\beta = 1$ corresponds to the normal watch percentage, and having $\beta < 1$ shifts the watch time distribution to be more fine-grained for shorter videos.

To address the dwell time bias relative to video duration (users tend to watch longer for longer videos), we also created a video duration debiased weighted categorical feature:

$$s(E_i) = \frac{\ln(\text{dwell\_time} + 1)}{\ln(\gamma \cdot \text{video\_duration} + 1)},$$
$$f_{\text{debiased\_wt\_score}}(E_i) = \text{select } (E_i, s(E_i)) \text{ if } s(E_i) > s_0, \quad (5)$$

where $s(\cdot)$ is the score formula, γ and s_0 are tunable hyper-parameters.

For the non-weighted categorical features, we also apply time-slicing:

$$g_{T_{\text{ub}}}(f(E)) = \text{select } f(E) \text{ if } \Delta t \in (0, T_{\text{ub}}], \quad (6)$$

where $f(E)$ can be any non-weighted categorical feature generated from the explicit (E_e) or implicit (E_i) timelines, Δt is equal to recommendation time minus the user's action time in the past, and T_{ub} is the upper bound of the time bucket. In addition to full time horizon, we choose 2 additional time buckets $T_{\text{ub}} \in \{3 \text{ days}, 7 \text{ days}\}$, and apply time-slicing on all action-based explicit features (Eq. 2) and the watchtime-based implicit features (Eq. 3).

We performed exhaustive experiments and selected 200 derived features based on offline eval metrics improvements. The non-weighted categorical features are aggregated to be a single embedding per user using mean-pooling, and the weighted categorical features are aggregated using weighted mean-pooling (apply weighted sum divided by sum of all weights). We also enable embedding table sharing by grouping the categorical features by the following criteria: positive engagements (e.g, watch_time > 15s) vs. negative engagements (e.g., watch_time < 3s), implicit vs. explicit timelines, id type (media id, author id, topic id), non-weighted vs weighted, and product type. This setup forces each of the embedding tables to have its own learning focus, while maintaining a low cost in memory usage thanks to table sharing.

2.3 Model architecture

2.3.1 Backbone Simulation Network. The design goal of backbone simulation network is to create a training wrapper to supervise the user encoder to learn a module that can generate user embeddings that's effective in heterogeneous downstream models.

We use DLRM [8] as backbone network which is a multitask ranker $\mathbf{\Phi}$ performing the following task: given a user with organized UniTi timelines E_i and E_e, targeting item features $\mathbf{x}$, and other user features $\mathbf{c}$, we aim to produce a score vector $\hat{\mathbf{y}} = \mathbf{\Phi}(E_i, E_e, \mathbf{x}, \mathbf{c})$ whose dimension is the same as the number of tasks, so that the total loss of all tasks is minimized. The loss is defined as cross entropy for binary tasks, and mean squared error (MSE) for regression tasks. The binary tasks loss is

$$\begin{aligned} l^{(k)}_{\text{binary}} &= \text{BCEloss}(\hat{y}^{(k)}, y^{(k)}) \\ &= -\frac{1}{B}\sum_{j=1}^{B}\left[y^{(k)}_j \ln(\hat{y}^{(k)}_j) + (1-y^{(k)}_j)\ln(1-\hat{y}^{(k)}_j)\right] \end{aligned} \tag{7}$$

where $y^{(k)}_j$ is the ground truth label for binary task k of the jth item in a mini batch, and $\hat{y}^{(k)}_j$ is the model's prediction. Our model has a single regression task predicting user's watch time, and its MSE loss is $l_{\text{reg}} = \frac{1}{B}\sum_{j=1}^{B}(z_j - \hat{z}_j)^2$. where z and $\hat{z}$ is the ground truth and prediction of the user's dwell time. The total loss is

$$L_{\text{total}} = \sum_k l^{(k)}_{\text{binary}} + a \cdot l_{\text{reg}}, \tag{8}$$

where a is a hyper-parameter to balance the scale of different loss types.

Distant Interest Prediction Objective To ensure that the user embedding is stable, we force the UniTi user timeline to have a 24h gap before the interaction time, i.e., removing the most recent 24h information from the UniTi timelines $\mathbf{E}^{-24\text{h}} = (E^{-24\text{h}}_i, E^{-24\text{h}}_e)$. This 24h gap amounts to introducing a harder prediction task which enforces the model to learn a long-term user interest. As a result, the embedding is more robust against delays in training and data pipelines (Details in B). The training objective then becomes

$$\hat{\mathbf{\Phi}} = \text{argmin}_{\mathbf{\Phi}} L_{\text{total}}(\mathbf{\Phi}(\mathbf{E}^{-24\text{h}}, \mathbf{x}, \mathbf{c}), \mathbf{y}). \tag{9}$$

The backbone model can also be rewritten as

$$\hat{\mathbf{y}} = \mathbf{\Phi}_{\phi}(\eta(\mathbf{E}^{-24\text{h}}), \mathbf{x}, \mathbf{c}), \tag{10}$$

where $\eta(\cdot)$ is the user-tower encoder that only has DV365 user timelines as input, and ϕ represents the rest of the parameters in $\mathbf{\Phi}$ other than η. The parameters in η and ϕ are continuously trained in an recurring fashion, and the resulting user embedding is produced by evaluating the user tower $\eta(\mathbf{E}^{-24\text{h}})$ across all active users on a 6-hourly basis.

2.3.2 Funnel Summarization Arch (FSA). Design goal of DV365 user encoder $\eta(\mathbf{E})$ is to encode the multi-sliced embeddings into a more compact user embeddings as a representation learner plus compressor for lower infra and downstream consumption cost.

First, we apply a token-wise view at raw embeddings. Given a user embedding tensor of size $[N, D]$, where N is the number of raw embedding after sparse lookup, and D is the embedding dimension. We transpose the tensor to be $\mathbf{U}$ of size $[D, N]$ and apply neural networks on the last dimension (token dimension). Compared to the traditional way of viewing it as $[1, N \times D]$ in MLP layers or $[N, D]$ in transformers, this transpose enforces parameter sharing across all dimensions of the raw (pooled) token embeddings and thus preserves the semantic meaning of embedding in the output.

Second, we apply Funnel Transformer [3] encoder on the token-wise view. The basic transformer encoder layer can be expressed as

$$\begin{aligned} &\text{EncoderLayer}(\mathbf{U}) := \\ &\quad \text{Norm}(\mathbf{U} + \text{FF}(\text{Norm}(\mathbf{U} + \text{Att}(q, k, v = \mathbf{U})))) \end{aligned} \tag{11}$$

where FF is the position-wise feedforward network, Att is multihead self attention, Norm is layer-normalization. The funnel transformer can have multiple blocks with multiple layers in each block. Between each block, mean pooling is applied on the token dimension with a stride, so that $\mathbf{U}_{\text{out}}$ is of size $[D, N_{\text{out}}]$ where $N_{\text{out}} < N$. Importantly, the pooling is applied to the token dimension instead of the embedding dimension as in original Funnel Transformer [3]. A Funnel Transformer encoder with n blocks, each block having m layers is described in Algorithm 1. In our experiments, the funnel transformer achieves consistent NE performance against the regular transformer when the regular transformer is used as the token-wise setup, but it achieves a higher training speed due to fewer parameters (section 3.4).

Algorithm 1 Funnel Transformer Encoder

```
Require: stride $\in \mathbb{Z}^+$
Require: $n, m \in \mathbb{Z}^+$   ▷ Number of blocks and attention layers per block
Require: EncoderLayer$_{i,j}$, $1 \le i \le n, 1 \le j \le m$
  for $i \leftarrow 1$ to $n$ do
      if $i == 1$ then                     ▷ Don't pool before the 1st block
          $\mathbf{U}' \leftarrow \mathbf{U}$
      else
          $\mathbf{U}' \leftarrow$ Pool($\mathbf{U}$, stride)        ▷ Pool in token dimension
      end if
      for $j \leftarrow 1$ to $m$ do
          $\mathbf{U}' =$ EncoderLayer$_{i,j}(\mathbf{U}')$
      end for
  end for
  return $\mathbf{U}'$
```

We also add a linear compression encoder (LCE) in parallel to the funnel transformer for better performance. During training, output embeddings are stacked and flattened as part of the simulation network's inputs. During publishing, they are quantized with 4-bit quantization before entering the feature store for serving. In total, the DV365 user tower (FSA and 4-bit quantization) compresses 200×256 fp32 numbers into 58×17 long integers, achieving a compression factor of 50.

2.4 Downstream Integration

To integrate with downstrewam models, we need to de-quantize back to float embeddings and do a learnable projection to make DV365 embeddings (58 of 256 dim embeddings). What we adopted can be summarized in 3 types:

1. **DLRM ranking models:** After a linear projection, we concatenate DV365 embeddings with other sparse embeddings originally in the model.
2. **HSTU module:** We prepend DV365 embeddings in the beginning of original HSTU input sequence (latest user engagements)
3. **Retrieval (Siamese network or MoL[15]):** We empirically found GateNet [6] is more effective than linear projection in retrieval integration.

2.5 Serving in Production

2.5.1 Existing System. The existing recommendation system initiates when a user sends a request through the Instagram frontend application. Upon receiving this request, the backend system prepares recommendation candidates and their associated features. These candidates are forwarded to the model inference service, where they are ranked based on predicted relevance. The ranked content is returned to the backend and delivered to the user. User interactions with the recommended content generate interaction events, which are logged to create training data for the core ranking model deployed in the inference service. The core ranking model employs an online training [13] approach, continuously updating model weights with new data to maintain freshness with minimal delay. The components of this system are depicted in Figure 2, highlighted in light green.

2.5.2 Embedding Precomputation and Serving System. Long-term user interests and behavioral patterns remain relatively stable over short timeframes, such as several hours or days. As a result, frequent computation of user embeddings from raw interaction data is unnecessary and computationally expensive. So we propose an offline embedding precomputation and serving system, as illustrated in Figure 2. The system incorporates the following key components and processes:

1. **Offline training data generation**: A data pipeline aggregates long-term user interaction histories and combines them with offline training data, forming a comprehensive dataset for training the DV365 foundation model.
2. **Foundation model training**: The DV365 foundation model is trained on a recurring schedule using the aggregated data. This periodic training ensures that the model catches up with data distribution updates, despite being constantly behind the data consumed by the production online training by about 2 days.
3. **Embedding precomputation and serving**: User embeddings are generated by exporting intermediate outputs from the user tower of the trained model. These embeddings are indexed by user IDs, stored in a database, and cached in an online key-value store for efficient retrieval. Storage optimization techniques, such as embedding precision reduction and quantization, are employed to minimize resource usage while preserving embedding quality.
4. **Recurring job scheduler**: A recurring job scheduler orchestrates the entire workflow, including data collection, model training, embedding generation, and caching. By operating at regular intervals, the scheduler ensures timely updates and balances computational efficiency with embedding freshness.

The offline embedding system is a cost-efficiency highlight since it has lower requirement on keeping things real-time. We are able to update model less frequently, and long user history data are kept in low-cost data warehouse without being exposed to expensive online serving storage. Power consumption is further optimized by refreshing embeddings based on user activity levels and applying incremental updates to capture changes in user behavior.

3 Experiments

We perform extensive experiments to test the performance of DV365 embedding in the upstream ranker model with DV365 user tower, as well as downstream ranker and retrieval models. For ranker models, we use the normalized entropy (NE) as an offline evaluation metric for the ranker model's performance on binary tasks

$$\mathrm{NE}_k = \frac{\mathrm{BCEloss}(\hat{y}^{(k)}, y^{(k)})}{-\left[p_k \ln(p_k) + (1-p_k)\ln(1-p_k)\right]}, \tag{12}$$

where p_k is the average CTR (click-through rate) for task k. The normalization in the denominator helps to prevent the metric from being artificially good when the background CTR is close to 0 or 1 [5]. Instead of absolute NE score, we use **Relative NE Delta** computed on exact same date range for test and control because data distribution and background CTR change over time. At Meta, this metric is widely regarded as one of the most reliable offline metrics, whose movement is consistent with online A/B test results across recommendation and Ads ranking. **>0.1% is considered significant and indicate A/B test gains.**

For retrieval models, we report hit rate @1 and @10 as evaluation metrics with a multitask version of MoL [15] but with simple dot-product as the user-item similarity function.

3.1 Dataset & Experiment Setup

Our dataset is Instagram's internal training data produced from recommendation logs. In existing production training table (used to train existing production models, or referred as downstream models for DV365), each training row consists of user engagements as labels and features of user history and items. User history length in the production training table is on average around 1500 and 2000 at maximum.

In order to train DV365 on extremely long user sequence, we created another table which append longer history on top of production training table offline which is more cost-efficient than appending directly in real-time generated production table. User history in this table is on average around 40,000 and 70,000 at maximum.

The DV365 foundation model described in section 2 is trained on the offline long history table, and we make our model decisions based on NE metrics on the foundation model because we assume

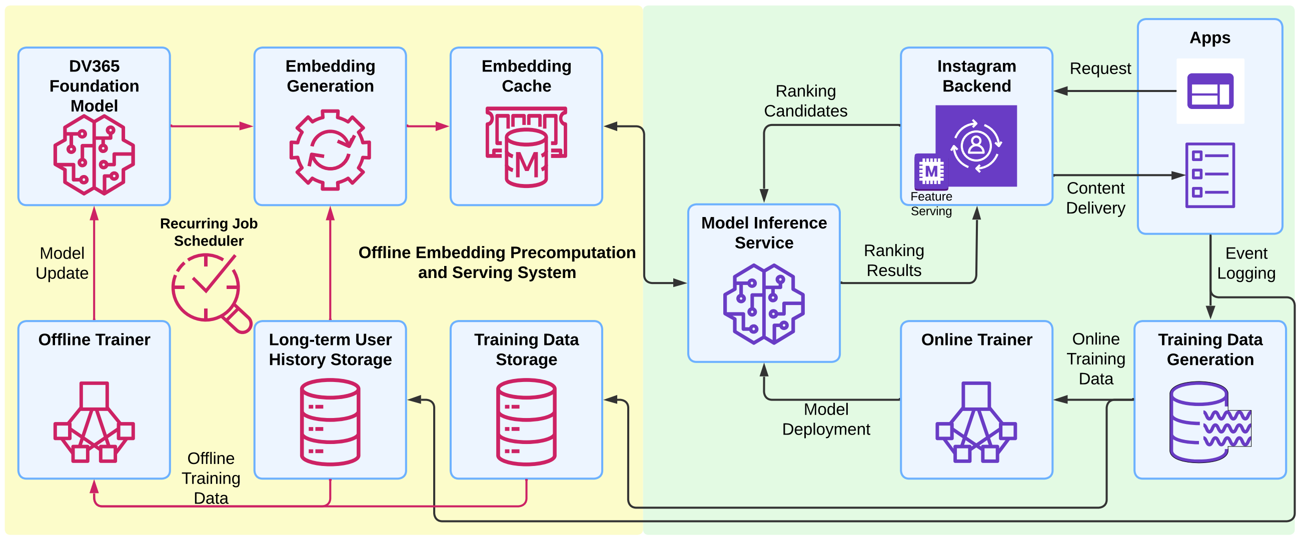

Figure 2: Architecture of the recommendation system architecture incorporating DV365 embeddings. New components introduced are indicated in light yellow.

improvement in foundation model will transfer to downstream models through exported user embeddings.

Downstream models for DV365, or real production models, are diverse in objectives (retrieval or ranking) and verticals (Reels, Feed, Notification). In this section, we report the downstream performance trained on production table in our biggest vertical, which is Reels, on both ranking and retrieval. The ranking model baseline is a DLRM model and retrieval model baseline is a MoL[15]) model. Both of them use HSTU as their user sequence encoder, and were the production model at the time. This established a strong baseline because they are real production model with an already long user sequence ($O(10^3)$) and an attentive encoder HSTU. We measure additional gains when DV365 integrated as features in these baselines.

3.2 Choice of Sequence Encoder

We choose mean-pooling as our sequence aggregation method given the scalability challenge of other methods in dealing with extremely long sequence. We experimented with attentive sequence models but didn't find substantial gain on top of our Multi-slicing & Summarization baseline, details discussed in appendix C.

3.3 Choice of Backbone Simulation Network

We hypothesize consistency of upstream and downstream models are important for the knowledge transfer. So we tested 2 model backbone choices: DLRM, consistent with all of our production ranking models; and siamese network, consistent with our retrieval models. DLRM backed upstream showed similar performance in downstream retrieval models, but performed significantly better in downstream ranking models (0.2% NE delta). So we choose DLRM as our backbone.

We also tested other factors in the simulation network:

(1) **Tasks**: Unimportant. Including only a subset of critical downstream prediction tasks in the backbone maximizes the DV365 embedding's effectiveness, while adding non-critical tasks yields minimal additional NE gains.
(2) **Model Scales**: Unimportant. Increasing the complexity of task-specific modules or interaction architectures provides limited additional benefits to the DV365 embedding's downstream performance.
(3) **Features**: Both sparse and dense features are important for simulation (0.2%+ NE difference with complete removal). But as expected, embedding quality win from more simulation features diminishes after enough features.

3.4 Choice of DV365 User Encoder Architecture

Table 1: FSA experiment setup. N_{out} means output embedding number

Name	User tower setup	N_{out}
baseline	no DV365 timelines	0
pass_thru	no compression	200
LCE	Linear compressed embedding	58
MLP_d	Dim-wise MLP	58
MLP_t	Token-wise MLP	58
$T\text{-}L_d$	Dim-wise Transformer + LCE	58
$T\text{-}L_t$	Token-wise Transformer + LCE	58
FSA	Funnel Transformer + LCE	58

The goal of user encoder is feature interaction learning and compressing multi-sliced embeddings into deployable sizes. There are 2 design choices in the user encoder:

Table 2: FSA compression performance compared to baseline in NE and MSE percentage delta (%). More negative means better performance. The best 2 NE among the experiments are in bold.

Task	pass_thru	LCE	MLP_d	MLP_t	$T\text{-}L_d$	$T\text{-}L_t$	FSA
video_complete	-0.593	-0.609	-0.486	-0.641	-0.500	**-0.659**	**-0.650**
skip	-0.480	-0.497	-0.395	-0.495	-0.398	**-0.519**	**-0.525**
reshare	-0.348	-0.327	-0.185	0.056	-0.169	**-0.342**	**-0.355**
profile_visit	**-0.507**	-0.412	-0.270	-0.433	0.394	-0.475	**-0.480**
external_share	**-3.351**	-2.811	-1.420	-2.420	-2.751	**-3.273**	-3.159
follow	**-1.034**	-0.801	-0.421	-0.858	-0.697	**-0.966**	-0.945
liked	-0.573	-0.564	-0.268	-0.580	-0.191	**-0.637**	**-0.634**
save	**-0.795**	-0.648	-0.206	-0.702	-0.005	**-0.739**	-0.729
comment	**-2.132**	-1.617	-1.025	-1.600	-1.272	**-2.004**	-1.927
$watch\_time_{MSE}$	-0.830	-0.848	-0.632	-0.843	-0.664	**-0.880**	**-0.865**

(1) **Compression perspective:** Dim-wise (apply encoder on [1, N x D] or [N, D]) or token-wise (transpose to [D, N] then encode)
(2) **Encoder architecture:** Linear Compression Encoder (LCE), MLP, Vanilla Transformer (take last N_{out} output embeddings), Funnel Transformer [3]

We also add a no compression baseline in the experiments. The experiment setup is in table 1, and results are in table 2. Results shows token-wise compression significantly outperforms dim-wise in all encoders, which indicate model favors parameter sharing within a single token embedding. Regarding to encoder choice, vanilla Transformer encoder and FSA yields best performance among compressed candidates. We chose FSA because it has fewer parameters and has 10% training QPS advantage.

3.5 Downstream Integration & Product Launches

For reporting convenience, we report offline result of only Reels surface (biggest engagement traffic), other surfaces yields similar results. For ranking model, DV365 improves NE significantly across all tasks (average over 0.4%) (Table 3) and linear projection results in better adoption. For retrieval model, DV365 increases hit rate by 2% to 8%, with GateNet as adoption module yields more gain in most tasks (table 4).

DV365 has been widely launched in **15** different product models across Instagram and Threads with significant business impacts. Accumulating A/B testing from all launches, it has improved **0.7% of Instagram App time spent** along with many other critical engagement metrics. It's highly generalizable: Despite highly heterogeneous downstream model architectures and objectives, we are able to use only one upstream model that generate a single set of embeddings to serve diverse downstream use cases, including improving content recommendation quality, notification CTR and jump starting new business initiative such as Threads.

We also verified that our design overcame the freshness disadvantage of DV365 as an offline embedding, i.e., embedding quality remains stable as staleness increases (details in Appendix B). This is also a system reliability highlight considering DV365 is a critical dependency of multiple products in Instagram.

Table 3: Downstream Ranking downstream NE difference in percentage delta (%). More negative the better.

Task	no projection	linear projection
video_complete	-0.564	**-0.637**
skip	-0.447	**-0.508**
reshare	-0.077	**-0.179**
profile_visit	-0.261	**-0.394**
follow	-0.282	**-0.592**
liked	-0.455	**-0.590**
save	-0.176	**-0.356**
comment	-0.574	**-1.151**
$watch\_time_{MSE}$	-0.737	**-0.835**

Table 4: Retrieval downstream adoption hit rates (HR). More positive means better performance.

Task HR	base	DV365	$DV365_{GateNet}$
follow@1	0.413	0.431 (4.4%)	**0.437 (5.8%)**
follow@10	0.792	0.796 (0.5%)	**0.802 (1.3%)**
liked@1	0.354	0.370 (4.5%)	**0.370 (4.5%)**
liked@10	0.810	0.827 (2.1%)	**0.830 (2.5%)**
save@1	0.423	0.439 (3.8%)	**0.446 (5.4%)**
save@10	0.852	0.863 (1.3%)	**0.874 (2.6%)**
reshare@1	0.316	**0.341 (7.9%)**	0.337 (6.6%)
reshare@10	0.788	**0.815 (3.4%)**	0.810 (2.8%)
profile_visit@1	0.374	0.398 (6.4%)	**0.402 (7.5%)**
profile_visit@10	0.811	0.830 (2.3%)	**0.836 (3.1%)**
watch_time@1	0.243	0.260 (7.0%)	**0.261 (7.4%)**
watch_time@10	0.710	0.735 (3.5%)	**0.738 (3.9%)**

4 ROI Advantages of Offline Embedding

We compare cost of achieving the same methodology used in DV365 as offline embedding and as part of E2E model. In Meta, we estimate cost by normalizing power usage in data center using Watt.

4.1 Feature Infra Cost Saving

Compared to online E2E long sequence modeling methods, key advantage of offline embedding is it saves significant cost from real-time feature extraction and processing for model inference, and cost from storing the extremely long user history in low latency key-value storage. We measured in details that delta of the two approaches is around 100 MW from feature extraction cost, and 11 PB low latency key value storage (Appendix A.2.2). This extremely high cost is the reason why extremely long user history is currently inaccessible in Meta's real-time serving infra.

4.2 Model Training Cost Saving

Model training cost saving comes from a single foundational model doing computation for all downstream models (15 in our case). In IG Reels model, additional feature preprocessing from long-history timelines to sliced features is measured 50 KW and GPU cost additional 338 A100 pool reservation. Considering not every model cost the same, we multiply total cost factor as 10 instead of 15, which is 500KW and 3380 A100 GPU reservation.

4.3 Model Serving Cost Saving

During model serving, feature preprocessing to slice timeline into features (saved by offline embedding) cost 6 MW and additional inference GPU cost (from user encoder) is 78 H100 (Appendix A.4) for IG Reels model, and estimated 780 H100 additional across fleet.

5 Related Works

5.1 Sequence User Modeling

User sequence modeling is a classic problem for recommendation, and industry state-of-the-art has converged to attentive methods where an attention weight was learnt for each item in the user sequence to aggregate full user interest representation. Notable examples are DIN [18], DIEN [17], SASRec [7], HSTU [16]. And HSTU is an important module currently deployed in Instagram production models. These attentive methods have a well known sequence length scalability challenge that most deployed solutions are only at O(100) level.

5.2 Long-term User Sequence Modeling

MIMN [10] and HPMN [12] use memory networks that improve deployable history access length to O(1000) but find it difficult scaling beyond that. Another theme of works (SIM [11], SDIM [1], TWIN [2]) introduces the two stage end-to-end cascaded framework to enable longer term history modeling. The two stages are: 1) a fast search unit which retrieves the most "relevant" items to the target item from thousands of user engagements, 2) an attentive network to perform target attention over the finalists from stage 1. These works claimed to support deployed solutions that model 10^4 user sequences. Despite the big improvement, these methods still significantly increased the cost and complexity of existing production models by introducing the search unit for both training and serving GPU, and more importantly, it requires serving infra to access the long user history feature in real-time, which is a big additional feature infra cost. Furthermore, as an end-to-end solution, their learnt long-term user interest can only benefit one model at a time without wide knowledge sharing.

5.3 User Embedding

Offline user embedding is another way to offload online computation to offline batch jobs for higher cost efficiency. Pinnerformer [9] introduced a transformer based upstream model to precompute user long-term interest with an offline embedding which is successfully deployed in Pinterest with high business impact, however, since it uses expensive encoder and only report to deploy max sequence length 256. LURM [14] introduced a bag-of-interest method to use pre-learned clustering to compress long history to a vector of per-cluster count, and used a self-supervised training objective to learn user embedding for generalization. Bag-of-interest is a very cost-efficient compression, but introduces a system dependency which is the clustering learning and the paper doesn't share A/B test and deployment results.

6 Conclusion

We proved our new long user history focused user embedding with high deployment success and strong ROI justifications. Specifically, our multi-slicing data model and user embedding summarization framework is able to capture sufficient information from the long user history and yield significant improvements in downstream production models equipped with SOTA model architectures. With this strategy, we are able to deploy the DV365 embedding to tens of downstream models with significant topline impact at Instagram.

Acknowledgments

We highly acknowledge Max Kaplan, Parichay Kapoor, Dhruv Choudhary, Hanxiong Chen, Zhengze Zhou, Mark Gluzman, Hanxiong Chen, Gufan Yin, Keyi Wu, Sami Bendouba, Xiaoyang Gong, Enzhou Liu, Ziheng Huang, Erheng Zhong, Joyce Liu for hands-on contributions, Ji Liu for insightful discussions, and Maxime Ransan, Arun Kumar Singh, Misael Manjarres, Nathan Berrebbi, Deepti Chheda, Abhishek Kumar, Haotian Wu for sponsorship and management support.

References

[1] Yue Cao, XiaoJiang Zhou, Jiaqi Feng, Peihao Huang, Yao Xiao, Dayao Chen, and Sheng Chen. 2022. Sampling Is All You Need on Modeling Long-Term User Behaviors for CTR Prediction. arXiv:2205.10249 [cs.IR] https://arxiv.org/abs/2205.10249

[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. arXiv:2302.02352 [cs.IR] https://arxiv.org/abs/2302.02352

[3] Zihang Dai, Guokun Lai, Yiming Yang, and Quoc V. Le. 2020. Funnel-Transformer: Filtering out Sequential Redundancy for Efficient Language Processing. https://api.semanticscholar.org/CorpusID:219401850

[4] Pinterest Engineering. 2022. How pinterest leverages Realtime user actions in recommendation to boost Homefeed engagement volume. https://medium.com/pinterest-engineering/how-pinterest-leverages-realtime-user-actions-in-recommendation-to-boost-homefeed-engagement-volume-165ae2e8cde8

[5] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, and Joaquin Quiñonero Candela. 2014. Practical Lessons from Predicting Clicks on Ads at Facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising* (New York, NY, USA) *(ADKDD'14)*. Association for Computing Machinery, New York, NY, USA, 1–9. doi:10.1145/2648584.2648589

[6] Tongwen Huang, Qingyun She, Zhiqiang Wang, and Junlin Zhang. 2020. GateNet: Gating-Enhanced Deep Network for Click-Through Rate Prediction. https://api.semanticscholar.org/CorpusID:220381423

[7] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. arXiv:1808.09781 [cs.IR] https://arxiv.org/abs/1808.09781

[8] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Zhihao Jia, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, Liang Luo, Jie (Amy) Yang, Leon Gao, Dmytro Ivchenko, Aarti Basant, Yuxi Hu, Jiyan Yang, Ehsan K. Ardestani, Xiaodong Wang, Rakesh Komuravelli, Ching-Hsiang Chu, Serhat Yilmaz, Huayu Li, Jiyuan Qian, Zhuobo Feng, Yinbin Ma, Junjie Yang, Ellie Wen, Hong Li, Lin Yang, Chonglin Sun, Whitney Zhao, Dimitry Melts, Krishna Dhulipala, KR Kishore, Tyler Graf, Assaf Eisenman, Kiran Kumar Matam, Adi Gangidi, Guoqiang Jerry Chen, Manoj Krishnan, Avinash Nayak, Krishnakumar Nair, Bharath Muthiah, Mahmoud khorashadi, Pallab Bhattacharya, Petr Lapukhov, Maxim Naumov, Ajit Mathews, Lin Qiao, Mikhail Smelyanskiy, Bill Jia, and Vijay Rao. 2022. Software-hardware co-design for fast and scalable training of deep learning recommendation models. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 993–1011. doi:10.1145/3470496.3533727

[9] Nikil Pancha, Andrew Zhai, Jure Leskovec, and Charles Rosenberg. 2022. PinnerFormer: Sequence Modeling for User Representation at Pinterest. arXiv:2205.04507 [cs.LG] https://arxiv.org/abs/2205.04507

[10] Qi Pi, Weijie Bian, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Practice on Long Sequential User Behavior Modeling for Click-Through Rate Prediction. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '19)*. ACM, New York, NY, USA, 2671–2679. doi:10.1145/3292500.3330666

[11] Pi Qi, 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. arXiv:2006.05639 [cs.IR] https://arxiv.org/abs/2006.05639

[12] Kan Ren, Jiarui Qin, Yuchen Fang, Weinan Zhang, Lei Zheng, Weijie Bian, Guorui Zhou, Jian Xu, Yong Yu, Xiaoqiang Zhu, and Kun Gai. 2019. Lifelong Sequential Modeling with Personalized Memorization for User Response Prediction. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '19)*. ACM, New York, NY, USA, 565–574. doi:10.1145/3331184.3331230

[13] Wikipedia. 2024. https://en.wikipedia.org/wiki/Online_machine_learning

[14] Bei Yang, Jie Gu, Ke Liu, Xiaoxiao Xu, Renjun Xu, Qinghui Sun, and Hong Liu. 2023. Empowering General-purpose User Representation with Full-life Cycle Behavior Modeling. arXiv:2110.11337 [cs.LG] https://arxiv.org/abs/2110.11337

[15] Jiaqi Zhai, Zhaojie Gong, Yueming Wang, Xiao Sun, Zheng Yan, Fu Li, and Xing Liu. 2023. Revisiting Neural Retrieval on Accelerators. 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, 5520–5531. doi:10.1145/3580305.3599897

[16] 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

[17] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2018. Deep Interest Evolution Network for Click-Through Rate Prediction. arXiv:1809.03672 [stat.ML] https://arxiv.org/abs/1809.03672

[18] Guorui Zhou, Chengru Song, Xiaoqiang Zhu, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep Interest Network for Click-Through Rate Prediction. arXiv:1706.06978 [stat.ML] https://arxiv.org/abs/1706.06978

A Detailed Cost Estimation

A.1 Stats of User History Length Modeled

Implicit timelines (user impression history): Max and average length are 35K and 30K; Explicit timelines (user explicit action history): Max and average length are 35K and 10K. The 35K maximum is a parameter we chose based on ROI. So potential total length is 70K in maximum and 40K on average.

A.2 Details of Feature Serving Infra Cost Estimation

A.2.1 Feature Extraction Cost.

(1) **Serving long sequence online** For cheaper estimation, we did a online load testing of extracting average length 60 user history in our feature infra, measured as 150 KW power, and estimate full DV365 (40K average) cost by multiply length ratio, which is 150 KW * (40K / 60) = 100 MW.
(2) **Serving DV365 embedding:** 21 KW as measured by production load testing.

A.2.2 Key-value Storage.

(1) **Serving long-sequence online:** 8 bytes (int64 type) * ((30K average length * 12 implicit attributes) + (10K average length * 10 explicit attributes)) * 3B (users) = 11 PB.
(2) **Serving DV365 embedding:** With int4 quantization, each output embedding takes 136 bytes and there are 58 output embeddings in total. Therefore, the total storage is 136 bytes / embedding * 58 embedding * 3B (users) = 24 TB.

A.3 Details of Model Training Cost Saving

Model training cost is the same for E2E and offline model training if there is only one consumer model of the offline embedding. Our saving mainly comes from one offline model that serves 15 downstream models.

Introducing DV365 components in a single model's cost has 2 parts:

(1) **Feature Preprocessing from Long-history timeline to sliced features:** 1K CPU hosts * 3 hour (training time) * 100 W / CPU host * 4 (training occurrence per day) / 24 hour (normalized over a day) = 50 KW.

(2) **Training GPU:** Additional model components in DV365 cost training QPS regression that translate to 338 A100 cards in our resource pool for both production and experimental training added together.

A.4 Details of Model Serving Cost Saving

Additional cost of E2E method modeling long sequence mainly comes from feature pre-processing computed in CPU hosts before sending processed features to GPU inference. We did an online A/B testing and measured 900KW to run DV365 feature preprocessing in user history average length of 60. Since we assume there is big optimization space, we apply a generous 0.01 optimization factor for fair estimation. So final estimation is 900KW * (40000 length we use / 60 tested length) * 0.01 optimization space factor = 6MW

Additional GPU inference cost comes from Funnel Summarization Arch, which cost inference QPS regression that translates to 78 H100 cards in IG Reels.

B Robustness to staleness

DV365 embeddings are designed to capture users' long-term, stable interests, making them inherently resilient to delays and staleness. This robustness ensures their effectiveness under variable update cadences, a crucial requirement for real-world applications.

To systematically assess the impact of staleness on model performance, we conducted a controlled experiment involving three downstream models: (1) a baseline model without DV365 embeddings, (2) a model with daily updated DV365 embeddings (referred to as "fresh embeddings"), and (3) a model using a fixed, non-updating version of the embeddings (referred to as "stale embeddings"). All models were trained on seven days of data, and the NE gain relative to the baseline model was continuously monitored.

Throughout the training period, the staleness of the fresh embeddings remained relatively stable, averaging around two days due to the daily updates. In contrast, the staleness of the stale embeddings increased steadily over time. Despite this disparity, both the fresh and stale embedding models demonstrated consistent NE gains of approximately 0.35% compared to the baseline, as shown in Figure 3. These results underscore the DV365 embeddings' robustness to staleness, ensuring reliable performance even when updates are delayed or entirely absent.

To further assess the stability of the DV365 embeddings over time, we analyzed the cosine similarity between different embedding versions. Remarkably, even when two versions were separated by seven days, the cosine similarity consistently exceeded 90%. This indicates that the embeddings maintain a high degree of stability over extended periods, as shown in Figure 4.

These findings highlight that the DV365 embeddings not only resist performance degradation caused by staleness but also exhibit exceptional consistency in representing user interests. This stability and reliability stem from the design of the DV365 embeddings, which leverage extensive user history to capture long-term patterns effectively, ensuring dependable performance across downstream models.

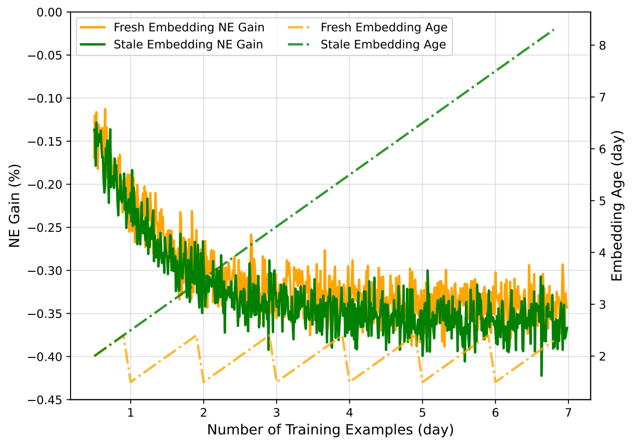

Figure 3: This figure illustrates the relationship between embedding NE gain (relative to the baseline model) and the number of training examples. The x-axis represents the cumulative number of training examples, measured in days. For the regularly updated embeddings, staleness fluctuates periodically due to daily updates. In contrast, the staleness of the fixed embeddings increases steadily over time, as these embeddings are not updated. The NE gain initially increases over the first three days of training data and then converges, stabilizing at a consistent level.

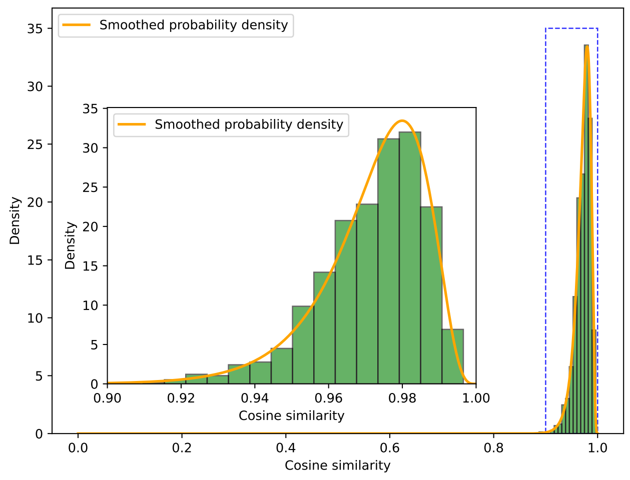

Figure 4: Distribution of cosine similarity between two user embedding versions. The similarity distribution is sharply unimodal, centered close to 1. The inset chart provides a closer view of the cosine similarity range between [0.9, 1.0].

C Choice of Sequence Encoder

We choose mean-pooling as our sequence aggregation method given the scalability challenge of other methods in dealing with extremely long sequence. But we still conduct experiments on attentive sequence models to understand its headroom on top of our multi-slicing pooling baseline:

(1) **Position-weighted pooling:** As a sequence model baseline, we simply learn a positional weight for each item in user sequence, and use it to apply weighted pooling.
(2) **HSTU [16]:** We added a HSTU encoder that encode user sequence into a summarized embedding with self attention. Given semantic similarity between HSTU and Transformer, we believe conclusion of this implementation can also represent Transformer based sequence encoders.

Our experiment results show both options is quite incremental if overall user sequence length is relatively (we apply a sequence length cap in the baseline to simulate this situation), but as we scale sequence length to 35000 for both explicit and implicit timeline in the baseline multi-slicing implementation, gains coming from additional sequence model encoded embedding fully diminished. (For HSTU, due to its scalability limitation we scaled it to 1000 instead of 35000).

Based on this observation, we decide to drop attentive sequence pooling in DV365 upstream model at the moment. We hypothesize that sequential information is mainly useful for user's emerging interest, but not so much on long-term stable interest. For stable interest, counting based summarization (captured in mean pooling) represents a strong baseline.

As mentioned in 6, we apply a cost-neutral time-slicing on the sequence before apply mean pooling, which gives us 0.1% NE gain in many tasks.