Actions Speak Louder than Words: Trillion-Parameter Sequential Transducers for Generative Recommendations

Jiaqi Zhai [1] **Lucy Liao** [1] **Xing Liu** [1] **Yueming Wang** [1] **Rui Li** [1]
Xuan Cao [1] **Leon Gao** [1] **Zhaojie Gong** [1] **Fangda Gu** [1] **Jiayuan He** [1] **Yinghai Lu** [1] **Yu Shi** [1]

Abstract

Large-scale recommendation systems are characterized by their reliance on high cardinality, heterogeneous features and the need to handle tens of billions of user actions on a daily basis. Despite being trained on huge volume of data with thousands of features, most Deep Learning Recommendation Models (DLRMs) in industry fail to scale with compute. Inspired by success achieved by Transformers in language and vision domains, we revisit fundamental design choices in recommendation systems. We reformulate recommendation problems as sequential transduction tasks within a generative modeling framework ("Generative Recommenders"), and propose a new architecture, HSTU, designed for high cardinality, non-stationary streaming recommendation data. HSTU outperforms baselines over synthetic and public datasets by up to 65.8% in NDCG, and is 5.3x to 15.2x faster than FlashAttention2-based Transformers on 8192 length sequences. HSTU-based Generative Recommenders, with 1.5 trillion parameters, improve metrics in online A/B tests by 12.4% and have been deployed on multiple surfaces of a large internet platform with billions of users. More importantly, the model quality of Generative Recommenders empirically scales as a power-law of training compute across three orders of magnitude, up to GPT-3/LLaMa-2 scale, which reduces carbon footprint needed for future model developments, and further paves the way for the first foundation models in recommendations.

1. Introduction

Recommendation systems, quintessential in the realm of online content platforms and e-commerce, play a pivotal role in personalizing billions of user experiences on a daily basis. State-of-the-art approaches in recommendations have been based on Deep Learning Recommendation Models (DLRMs) (Mudigere et al., 2022) for about a decade (Covington et al., 2016; Cheng et al., 2016; Zhou et al., 2018; Tang et al., 2020; Wang et al., 2021; Xia et al., 2023). DLRMs are characterized by their usage of heterogeneous features, such as numerical features – counters and ratios, embeddings, and categorical features such as creator ids, user ids, etc. Due to new content and products being added every minute, the feature space is of extreme high cardinality, often in the range of billions (Eksombatchai et al., 2018). To leverage tens of thousands of such features, DLRMs employ various neural networks to combine features, transform intermediate representations, and compose the final outputs.

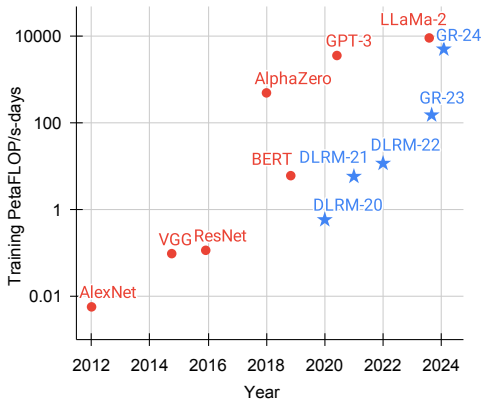


Figure 1. Total compute used to train deep learning models over the years. DLRM results are from (Mudigere et al., 2022); GRs are deployed models from this work. DLRMs/GRs are continuously trained in a streaming setting; we report compute used per year.

Despite utilizing extensive human-engineered feature sets and training on vast amounts of data, most DLRMs in industry scale poorly with compute (Zhao et al., 2023). This limitation is noteworthy and remains unanswered.

Inspired by the success achieved by Transformers in language and vision, we revisit fundamental design choices in modern recommendation systems. We observe that alternative formulations at billion-user scale need to overcome three challenges. First, features in recommendation systems lack explicit structures. While sequential formulations have been explored in small-scale settings (detailed discussions

[1]MRS, Meta AI. Correspondence to: <{jiaqiz, lucyyl, xingl, yuemingw, ruili}@meta.com>. Code available at https://github.com/facebookresearch/generative-recommenders.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

in Appendix B), heterogeneous features, including high cardinality ids, cross features, counters, ratios, etc. play critical roles in industry-scale DLRMs (Mudigere et al., 2022). Second, recommendation systems use billion-scale vocabularies that change continuously. A billion-scale dynamic vocabulary, in contrast to 100K-scale static ones in language (Brown et al., 2020), creates training challenges and necessitates high inference cost given the need to consider tens of thousands of candidates in a target-aware fashion (Zhou et al., 2018; Wang et al., 2020). Finally, computational cost represents the main bottleneck in enabling large-scale sequential models. GPT-3 was trained on a total of 300B tokens over a period of 1-2 months with thousands of GPUs (Brown et al., 2020). This scale appears daunting, until we contrast it with the scale of user actions. The largest internet platforms serve billions of daily active users, who engage with billions of posts, images, and videos per day. User sequences could be of length up to 10^5 (Chang et al., 2023). Consequentially, recommendation systems need to handle a few orders of magnitude more tokens *per day* than what language models process over 1-2 months.

In this work, we treat *user actions* as a new *modality* in generative modeling. Our key insights are, a) core ranking and retrieval tasks in industrial-scale recommenders can be cast as generative modeling problems given an appropriate new feature space; b) this paradigm enables us to systematically leverage redundancies in features, training, and inference to improve efficiency. Due to our new formulation, we deployed models that are *three orders of magnitude* more computationally complex than prior state-of-the-art, while improving topline metrics by 12.4%, as shown in Figure 1.

Our contributions are as follows. We first propose *Generative Recommenders* (GRs) in Section 2, a new paradigm replacing DLRMs. We sequentialize and unify the heterogeneous feature space in DLRMs, with the new approach approximating the full DLRM feature space as sequence length tends to infinity. This enables us to reformulate the main recommendation problems, ranking and retrieval, as pure sequential transduction tasks in GRs. Importantly, this further enables model training to be done in a sequential, generative fashion, which permits us to train on orders of magnitude more data *with the same amount of compute*.

We next address computational cost challenges throughout training and inference. We propose a new sequential transduction architecture, *Hierarchical Sequential Transduction Units* (HSTU). HSTU modifies attention mechanism for large, non-stationary vocabulary, and exploits characteristics of recommendation datasets to achieve 5.3x to 15.2x speedup vs FlashAttention2-based Transformers on 8192 length sequences. Further, through a new algorithm, M-FALCON, that fully amortizes computational costs via micro-batching (Section 3.4), we can serve *285x more complex* GR models while achieving 1.50x-2.99x speedups, all *with the same inference budget* used by traditional DLRMs.

We finally validate the proposed techniques over synthetic datasets, public datasets, and deployments on multiple surfaces of a large internet platform with billions of daily active users in Section 4. To the best of our knowledge, our work represents the first result that shows pure sequential transduction-based architectures, like HSTU, in generative settings (GRs) to significantly outperform DLRMs in large-scale industrial settings. Remarkably, not only did we overcome known scaling bottlenecks in traditional DLRMs, we further succeeded in showing that scaling law (Kaplan et al., 2020) applies to recommendations, representing the potential ChatGPT moment for recommendation systems.

2. Recommendation as Sequential Transduction Tasks: From DLRMs to GRs

2.1. Unifying heterogeneous feature spaces in DLRMs

Modern DLRM models are usually trained with a vast number of categorical ('sparse') and numerical ('dense') features. In GRs, we consolidate and encode these features into a single unified time series, as depicted in Figure 2.

Categorical ('sparse') features. Examples of such features include items that user liked, creators in a category (e.g., Outdoors) that user is following, user languages, communities that user joined, cities from which requests were initiated, etc. We *sequentialize* these features as follows. We first select the longest time series, typically by merging the features that represent items user engaged with, as the main time series. The remaining features are generally time series that slowly change over time, such as demographics or followed creators. We *compress* these time series by keeping the earliest entry per consecutive segment and then merge the results into the main time series. Given these time series change very slowly, this approach does not significantly increase the overall sequence length.

Numerical ('dense') features. Examples of such features include weighted and decayed counters, ratios, etc. For instance, one feature could represent user's past click through rate (CTR) on items matching a given topic. Compared to categorical features, these features change much more frequently, potentially with every single (user, item) interaction. It is therefore infeasible to fully sequentialize such features from computational and storage perspectives. However, an important observation is that the categorical features (e.g., item topics, locations) over which we perform these aggregations are already sequentialized and encoded in GRs. Hence, we can remove numerical features in GRs given a sufficiently expressive sequential transduction architecture *coupled with a target-aware formulation* (Zhou et al., 2018) can meaningfully capture numerical features as we increase

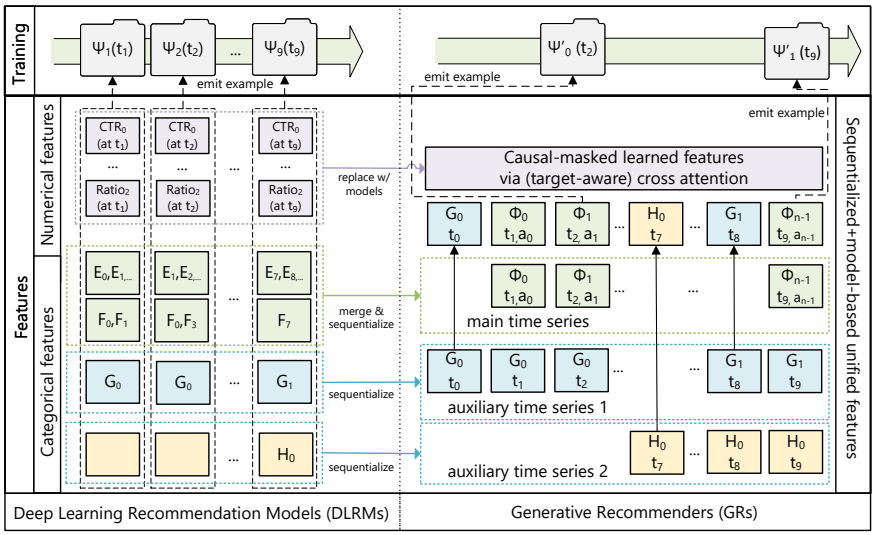


Figure 2. Comparison of features and training procedures: DLRMs vs GRs. E, F, G, H denote categorical features. Φ_i represents the i-th item in the merged main time series. $\Psi_k(t_j)$ denotes training example k emitted at time t_j. Full notations can be found in Appendix A.

the overall sequence length and compute in GRs.

2.2. Reformulating ranking and retrieval as sequential transduction tasks

Given a list of n tokens $x_0, x_1, \ldots, x_{n-1}$ ($x_i \in \mathbb{X}$) ordered chronologically, the time when those tokens are observed $t_0, t_1, \ldots, t_{n-1}$, a sequential transduction task maps this input sequence to the output tokens $y_0, y_1, \ldots, y_{n-1}$ ($y_i \in \mathbb{X} \cup \{\varnothing\}$), where $y_i = \varnothing$ indicates that y_i is undefined.

We use $\Phi_i \in \mathbb{X}_c$ ($\mathbb{X}_c \subseteq \mathbb{X}$) to denote a content (e.g., images or videos) that the system provides to the user. Given new content are constantly created, $\mathbb{X}_c$ and $\mathbb{X}$ are non-stationary. The user can respond to Φ_i with some action a_i (e.g., like, skip, video completion+share) $a_i \in \mathbb{X}$. We denote the total number of contents that a user has interacted with by n_c.

The standard ranking and retrieval tasks, in causal autoregressive settings, can then be defined as sequential transduction tasks (Table 1). We make the following observations:

Task		Specification (Inputs / Outputs)
Ranking	x_is	$\Phi_0, a_0, \Phi_1, a_1, \ldots, \Phi_{n_c-1}, a_{n_c-1}$
	y_is	$a_0, \varnothing, a_1, \varnothing, \ldots, a_{n_c-1}, \varnothing$
Retrieval	x_is	$(\Phi_0, a_0), (\Phi_1, a_1), \ldots, (\Phi_{n_c-1}, a_{n_c-1})$
	y_is	$\Phi'_1, \Phi'_2, \ldots, \Phi'_{n_c-1}, \varnothing$ ($\Phi'_i = \Phi_i$ if a_i is positive, otherwise $\varnothing$)

Table 1. Ranking and retrieval as sequential transduction tasks. Other categorical features are omitted for simplicity. We compare GRs with traditional sequential recommenders in Appendix B.2.

Retrieval. In recommendation system's retrieval stage, we learn a distribution $p(\Phi_{i+1}|u_i)$ over $\Phi_{i+1} \in \mathbb{X}_c$, where u_i is the user's representation at token i. A typical objective is to select $\arg\max_{\Phi \in \mathbb{X}_c} p(\Phi|u_i)$ to maximize some reward. This differs from a standard autoregressive setup in two ways. First, the supervision for x_i, y_i, is not necessarily Φ_{i+1}, as users could respond negatively to Φ_{i+1}. Second, y_i is undefined when x_{i+1} represents a non-engagement related categorical feature, such as demographics.

Ranking. Ranking tasks in GRs pose unique challenges as industrial recommendation systems often require a "target-aware" formulation. In such settings, "interaction" of target, Φ_{i+1}, and historical features needs to occur as early as possible, which is infeasible with a standard autoregressive setup where "interaction" happens late (e.g., via softmax after encoder output). We address this by *interleaving* items and actions in Table 1, which enables the ranking task to be formulated as $p(a_{i+1}|\Phi_0, a_0, \Phi_1, a_1, \ldots, \Phi_{i+1})$ (before categorical features). We apply a small neural network to transform outputs at Φ_{i+1} into multi-task predictions in practice. Importantly, this enables us to apply target-aware cross-attention to all n_c engagements in one pass.

2.3. Generative training

Industrial recommenders are commonly trained in a streaming setup, where each example is processed sequentially as they become available. In this setup, the total computational requirement for self-attention based sequential transduction architectures, such as Transformers (Vaswani et al., 2017), scales as $\sum_i n_i(n_i^2 d + n_i d_{ff} d)$, where n_i is the number of tokens of user i, and d is the embedding dimension. The first part in the parentheses comes from self-attention, with assumed $O(n^2)$ scaling factor due to most subquadratic algorithms involving quality tradeoffs and underperforming quadratic algorithms in wall-clock time (Dao et al., 2022). The second part comes from pointwise MLP layers, with hidden layers of size $O(d_{ff}) = O(d)$. Taking $N = \max_i n_i$, the overall time complexity reduces to $O(N^3 d + N^2 d^2)$, which is cost prohibitive for recommendation settings.

To tackle the challenge of training sequential transduc-

tion models over long sequences in a scalable manner, we move from traditional impression-level training to *generative training*, reducing the computational complexity by an $O(N)$ factor, as shown at the top of Figure 2. By doing so, encoder costs are amortized across multiple targets. More specifically, when we sample the i-th user at rate $s_u(n_i)$, the total training cost now scales as $\sum_i s_u(n_i) n_i (n_i^2 d + n_i d^2)$, which is reduced to $O(N^2 d + N d^2)$ by setting $s_u(n_i)$ to $1/n_i$. One way to implement this sampling in industrial-scale systems is to emit training examples at the end of a user's request or session, resulting in $\hat{s_u}(n_i) \propto 1/n_i$.

3. A High Performance Self-Attention Encoder for Generative Recommendations

To scale up GRs for industrial-scale recommendation systems with large, non-stationary vocabularies, we next introduce a new encoder design, *Hierarchical Sequential Transduction Unit* (HSTU). HSTU consists of a stack of identical layers connected by residual connections (He et al., 2015). Each layer contains three sub-layers: Pointwise Projection (Equation 1), Spatial Aggregation (Equation 2), and Pointwise Transformation (Equation 3):

$$U(X), V(X), Q(X), K(X) = \text{Split}(\phi_1(f_1(X))) \quad (1)$$

$$A(X)V(X) = \phi_2\left(Q(X)K(X)^T + \text{rab}^{p,t}\right)V(X) \quad (2)$$

$$Y(X) = f_2\left(\text{Norm}\left(A(X)V(X)\right) \odot U(X)\right) \quad (3)$$

where $f_i(X)$ denotes an MLP; we use one linear layer, $f_i(X) = W_i(X) + b_i$ for f_1 and f_2 to reduce compute complexity and further batches computations for queries $Q(X)$, keys $K(X)$, values $V(X)$, and gating weights $U(X)$ with a fused kernel; ϕ_1 and ϕ_2 denote nonlinearity, for both of which we use SiLU (Elfwing et al., 2017); Norm is layer norm; and $\text{rab}^{p,t}$ denotes relative attention bias (Raffel et al., 2020) that incorporates positional (p) and temporal (t) information. Full notations used can be found in Table 9.

HSTU encoder design allows for the replacement of heterogeneous modules in DLRMs with a single modular block. We observe that there are, effectively, three main stages in DLRMs: *Feature Extraction*, *Feature Interactions*, and *Transformations of Representations*. *Feature Extractions* retrieves the pooled embedding representations of categorical features. Their most advanced versions can be generalized as pairwise attention and target-aware pooling (Zhou et al., 2018), which is captured with HSTU layers.

Feature Interaction is the most critical part of DLRMs. Common approaches used include factorization machines and their neural network variants (Rendle, 2010; Guo et al., 2017; Xiao et al., 2017), higher order feature interactions (Wang et al., 2021), etc. HSTU replaces feature interactions by enabling attention pooled features to directly "interact" with other features via $\text{Norm}\left(A(X)V(X)\right) \odot U(X)$.

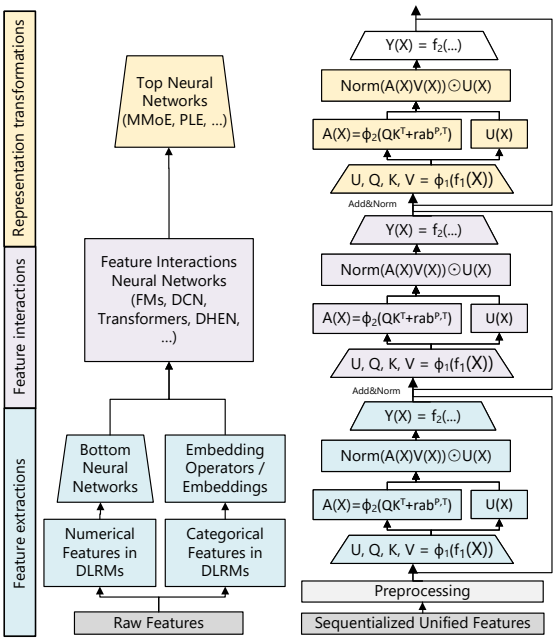


Figure 3. Comparison of key model components: DLRMs vs GRs. The complete DLRM setup (Mudigere et al., 2022) is shown on the left side and a simplified HSTU is shown on the right.

This design is motivated by the difficulty of approximating dot products with learned MLPs (Rendle et al., 2020; Zhai et al., 2023a). Given SiLU is applied to $U(X)$, $\text{Norm}\left(A(X)V(X)\right) \odot U(X)$ can also be interpreted as a variant of SwiGLU (Shazeer, 2020).

Transformations of Representations is commonly done with Mixture of Experts (MoEs) and routing to handle diverse, heterogeneous populations. A key idea is to perform conditional computations by specializing sub-networks for different users (Ma et al., 2018; Tang et al., 2020). Element-wise dot products in HSTU can virtually perform gating operations used in MoEs up to a normalization factor.

3.1. Pointwise aggregated attention

HSTU adopts a new pointwise aggregated (normalized) attention mechanism (in contrast, softmax attention computes normalization factor over the entire sequence). This is motivated by two factors. First, the number of prior data points related to target serves as a strong feature indicating the intensity of user preferences, which is hard to capture after softmax normalization. This is important as we need to predict both the intensity of engagements, e.g., time spent on a given item, and the relative ordering of the items, e.g., predicting an ordering to maximize AUC. Second, while softmax activation is robust to noise by construction, it is less suited for non-stationary vocabularies in streaming settings.

The proposed pointwise aggregated attention mechanism is depicted in Equation (2). Importantly, layer norm is needed

Architecture	HR@10	HR@50
Transformers	.0442	.2025
HSTU (-rab^{p,t}, Softmax)	.0617	.2496
HSTU (-rab^{p,t})	**.0893**	**.3170**

Table 2. Synthetic data in one-pass streaming settings.

after pointwise pooling to stabilize training. One way to understand this design is through synthetic data following a Dirichlet Process that generates streaming data over a nonstationary vocabulary (details in Appendix C). In this setting, we can observe gaps as large as 44.7% between softmax and pointwise attention setups as shown in Table 2.

3.2. Leveraging and algorithmically increasing sparsity

In recommendation systems, the length of user history sequences often follows a skewed distribution, leading to sparse input sequences, particularly in the settings with very long sequences. This sparsity can be leveraged to significantly improve the efficiency of the encoder. To achieve this, we have developed an efficient attention kernel for GPUs that fuses back-to-back GEMMs in a manner similar to (Rabe & Staats, 2021; Dao et al., 2022) but performs *fully raggified* attention computations. This essentially transforms the attention computation into grouped GEMMs of various sizes (Appendix G). As a result, self-attention in HSTU becomes memory-bound and scales as $\Theta(\sum_i n_i^2 d_{qk}^2 R^{-1})$ in terms of memory accesses, where n_i is the sequence length for sample i, d_{qk} is attention dimension, and R is the register size. This approach by itself leads to 2-5x throughput gains as discussed in Section 4.2.

We further *algorithmically* increase the sparsity of user history sequences via *Stochastic Length* (SL). One key characteristic of user history sequences in recommendations is that user behaviors are temporally repetitive, as user behaviors manifest at multiple scales throughout their interaction history. This represents an opportunity to increase sparsity *artificially* without compromising model quality, thereby significantly reducing encoder cost that scales as $\Theta(\sum_i n_i^2)$.

We can represent user j's history as a sequence $(x_i)_{i=0}^{n_{c,j}}$, where $n_{c,j}$ is the number of contents user interacted with. Let $N_c = \max_j n_{c,j}$. Let $(x_{i_k})_{k=0}^{L}$ be a subsequence of length L constructed from the original sequence $(x_i)_{i=0}^{n_{c,j}}$. SL selects input sequences as follows:

$$\begin{cases} (x_i)_{i=0}^{n_{c,j}} \text{ if } n_{c,j} \le N_c^{\alpha/2} \\ (x_{i_k})_{k=0}^{N_c^{\alpha/2}} \text{ if } n_{c,j} > N_c^{\alpha/2}, \text{w/ probability } 1 - N_c^{\alpha}/n_{c,j}^2 \\ (x_i)_{i=0}^{n_{c,j}} \text{ if } n_{c,j} > N_c^{\alpha/2}, \text{w/ probability } N_c^{\alpha}/n_{c,j}^2 \end{cases} \quad (4)$$

which reduces attention-related complexity to $O(N_c^{\alpha} d) = O(N^{\alpha} d)$ for $\alpha \in (1, 2]$. A more thorough discussion of subsequence selection can be found in Appendix F.1. We remark that applying SL to training leads to a cost-effective system design, as training generally involves a significantly higher computational cost compared to inference.

Alpha (α)	Max Sequence Lengths			
	1,024	2,048	4,096	8,192
1.6	71.5%	76.1%	80.5%	84.4%
1.7	56.1%	63.6%	69.8%	75.6%
1.8	40.2%	45.3%	54.1%	66.4%
1.9	17.2%	21.0%	36.3%	64.1%
2.0	3.1%	6.6%	29.1%	64.1%

Table 3. Impact of *Stochastic Length* (SL) on sequence sparsity.

Table 3 presents the *sparsity* (see Appendix F) for different sequence lengths and α values, for a representative industry-scale configuration with 30-day user history. The settings that result in negligible regression in model quality are underlined and highlighted in blue. The rows labeled "$\alpha = 2.0$" represents the base sparsity case where SL is not applied. Lower α's are applicable to longer sequences up to the longest sequence length we tested, 8,192.

3.3. Minimizing activation memory usage

In recommendation systems, using large batch sizes is important for both training throughput (Mudigere et al., 2022) and model quality (Yang et al., 2020; Chen et al., 2020; Zhai et al., 2023a). Consequently, activation memory usage becomes a major scaling bottleneck, in contrast to large language models that are commonly trained with small batch sizes and dominated by parameter memory usage.

Compared to Transformers, HSTU employs a simplified and fully fused design that significantly reduces activation memory usage. Firstly, HSTU reduces the number of linear layers outside of attention from six to two, aligning with recent work that uses elementwise gating to reduce MLP computations (Hua et al., 2022; Gu et al., 2022). Secondly, HSTU aggressively fuses computations into single operators, including $\phi_1(f_1(\cdot))$ in Equation (1), and layer norm, optional dropout, and output MLP in Equation (3). This simplified design reduces the activation memory usage to $2d+2d+4hd_{qk}+4hd_v+2hd_v = 14d$ per layer in bfloat16.

For comparison, Transformers use a feedforward layer and dropout after attention (intermediate state of $3hd_v$), followed by a pointwise feedforward block consisting of layer norm, linear, activation, linear, and dropout, with intermediate states of $2d + 4d_{ff} + 2d + 1d = 4d + 4d_{ff}$. Here, we make standard assumptions that $hd_v \ge d$ and that $d_{ff} = 4d$ (Vaswani et al., 2017; Brown et al., 2020). Thus, after accounting for input and input layer norm ($4d$) and qkv projections, the total activation states is $33d$. HSTU's design hence enables scaling to $>$ 2x deeper layers.

Additionally, large scale atomic ids used to represent vocabularies also require significant memory usage. With a 10b vocabulary, 512d embeddings, and Adam optimizer, storing embeddings and optimizer states in fp32 already requires 60TB memory. To alleviate memory pressure, we employ

rowwise AdamW optimizers (Gupta et al., 2014; Khudia et al., 2021) and place optimizer states on DRAM, which reduces HBM usage per float from 12 bytes to 2 bytes.

3.4. Scaling up inference via cost-amortization

The last challenge we address is the large number of candidates recommendation systems need to process at serving time. We focus on ranking as for retrieval, encoder cost is fully amortizable, and efficient algorithms exist for both MIPS leveraging quantization, hashing, or partitioning (Jegou et al., 2011; Shrivastava & Li, 2014; Li et al., 2002; Zhai et al., 2011) and non-MIPS cases via beam search or hierarchical retrieval (Zhuo et al., 2020; Zhai et al., 2023a).

For ranking, we have up to tens of thousands of candidates (Covington et al., 2016; Wang et al., 2020). We propose an algorithm M-FALCON (Microbatched-Fast Attention Leveraging Cacheable OperatioNs) to perform inference for m candidates with an input sequence size of n.

Within a forward pass, M-FALCON handles b_m candidates in parallel by modifying attention masks and $\text{rab}^{p,t}$ biases such that the attention operations performed for b_m candidates are exactly the same. This reduces the cost of applying cross-attention from $O(b_m n^2 d)$ to $O((n+b_m)^2 d) = O(n^2 d)$ when b_m can be considered a small constant relative to n. We optionally divide the overall m candidates into $\lceil m/b_m \rceil$ microbatches of size b_m to leverage encoder-level KV caching (Pope et al., 2022) either across forward passes to reduce cost, or across *requests* to minimize tail latency (detailed discussions can be found in Appendix H).

Overall, M-FALCON enables model complexity to linearly scale up with the number of candidates in traditional DLRMs's ranking stages; we succeeded in applying *a 285x* more complex target-aware cross attention model at 1.5x-3x throughput with a constant inference budget for a typical ranking configuration discussed in Section 4.3.

4. Experiments

4.1. Validating Inductive Hypotheses of HSTU Encoder

4.1.1. Traditional Sequential Settings

We first evaluate the performance of HSTU on two popular recommender datasets, MovieLens and Amazon Reviews. We follow traditional sequential recommender settings in literature, including *full shuffle* and *multi-epoch* training. For baseline, we use SASRec, a state-of-the-art Transformer implementation (Kang & McAuley, 2018) [1]. We report Hit Rate@K and NDCG@K over the entire corpus, consistent with recent work (Dallmann et al., 2021; Zhai et al., 2023a).

Results are presented in Table 4. "SASRec (2023)" denotes the best SASRec recipe reported in (Zhai et al., 2023a). The rows labeled "HSTU" use identical configurations as SASRec (same number of layers, heads, etc.). "HSTU-large" represents larger HSTU encoders (4x number of layers and 2x number of heads). Results show that a) HSTU, with its design optimized for recommendations, significantly outperforms the baseline when using the same configuration, and b) HSTU further improves performance when scaled up.

It is important to note that the evaluation methodology used here differs significantly from industrial-scale settings, as *full-shuffle* and *multi-epoch* training are generally not practical in streaming settings used in industry (Liu et al., 2022).

4.1.2. Industrial-scale Streaming Settings

We next compare the performance of HSTU, ablated HSTUs, and transformers using industrial-scale datasets in a streaming setting. Throughout the rest of this section, we report Normalized Entropy (NE) (He et al., 2014) for ranking. We train the models over 100B examples (DLRM equivalent), with 64-256 H100s used per job. Given ranking is done in a multi-task setting, we report the main engagement event ("E-Task") and the main consumption event ("C-Task"). In our context, we consider a 0.001 reduction in NE significant as it generally leads to .5% topline metric improvements for billions of users. For retrieval, given the setup is similar to language modeling, we report log perplexity. We fix encoder parameters in a smaller-scale setting ($l = 3$, $n = 2048$, $d = 512$ for ranking and $l = 6$, $n = 512$, $d = 256$ for retrieval), and grid-search other hyperparameters due to resource limits.

We show results in Table 5. First, HSTU significantly outperforms Transformers, especially in ranking, likely due to pointwise attention and improved relative attention biases. Second, the gaps between the ablated HSTUs and HSTU confirm the effectiveness of our designs. Optimal learning rates are about 10x lower for Softmax-based HSTU and Transformer vs the rest due to training stability. Even with lower learning rates and pre-norm residual connections (Xiong et al., 2020), we encountered frequent loss explosions with standard Transformers in ranking. Finally, HSTU outperforms a popular Transformer variant used in LLMs, Transformer++ (Touvron et al., 2023a), which uses RoPE (Su et al., 2023), SwiGLU, etc. Overall, in this small-scale setting, HSTU shows better quality at 1.5x-2x faster wall-clock time and 50% less HBM usage.

4.2. Encoder Efficiency

Stochastic Length. Figure 4 and Figure 5 (a) show the impact of *stochastic length* (SL) on model metrics. At $\alpha = 1.6$, a sequence of length 4096 is turned into a sequence of length 776 the majority of the time, or removing more than 80% of the tokens. Even after sparsity ratio increases to 64%-

[1] Results for other baselines are reported in Appendix D.

Table 4. Evaluations of methods on public datasets in multi-pass, full-shuffle settings.

	Method	HR@10	HR@50	HR@200	NDCG@10	NDCG@200
ML-1M	SASRec (2023)	.2853	.5474	.7528	.1603	.2498
	HSTU	.3097 (+8.6%)	.5754 (+5.1%)	.7716 (+2.5%)	.1720 (+7.3%)	.2606 (+4.3%)
	HSTU-large	**.3294 (+15.5%)**	**.5935 (+8.4%)**	**.7839 (+4.1%)**	**.1893 (+18.1%)**	**.2771 (+10.9%)**
ML-20M	SASRec (2023)	.2906	.5499	.7655	.1621	.2521
	HSTU	.3252 (+11.9%)	.5885 (+7.0%)	.7943 (+3.8%)	.1878 (+15.9%)	.2774 (+10.0%)
	HSTU-large	**.3567 (+22.8%)**	**.6149 (+11.8%)**	**.8076 (+5.5%)**	**.2106 (+30.0%)**	**.2971 (+17.9%)**
Books	SASRec (2023)	.0292	.0729	.1400	.0156	.0350
	HSTU	.0404 (+38.4%)	.0943 (+29.5%)	.1710 (+22.1%)	.0219 (+40.6%)	.0450 (+28.6%)
	HSTU-large	**.0469 (+60.6%)**	**.1066 (+46.2%)**	**.1876 (+33.9%)**	**.0257 (+65.8%)**	**.0508 (+45.1%)**

Table 5. Evaluation of HSTU, ablated HSTU, and Transformers on industrial-scale datasets in one-pass streaming settings.

Architecture	Retrieval	Ranking (NE)	
	log pplx.	E-Task	C-Task
Transformers	4.069	NaN	NaN
HSTU (-rab^{p,t}, Softmax)	4.024	.5067	.7931
HSTU (-rab^{p,t})	4.021	.4980	.7860
Transformer++	4.015	.4945	.7822
HSTU (original rab)	4.029	.4941	.7817
HSTU	**3.978**	**.4937**	**.7805**

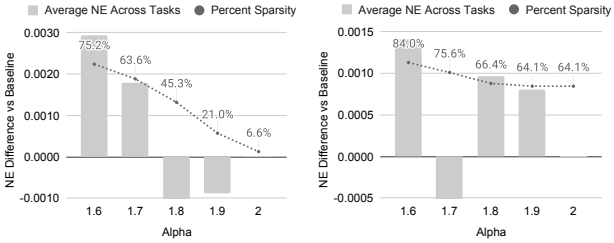


Figure 4. Impact of *Stochastic Length* (SL) on metrics. Left: $n = 4096$. Right: $n = 8192$. Full results can be found in Appendix F.

84%, the NEs we obtained for main tasks did not degrade by more than 0.002 (0.2%). This evidence supports that SL, for suitable αs, does not negatively impact model quality and allows for high sparsity to reduce training cost. We further verify in Appendix F.3 that SL significantly outperforms existing length extrapolation techniques.

Encoder Efficiency. Figure 5 compares the efficiency of HSTU and Transformer encoders in training and inference settings. For Transformers, we use the state-of-the-art FlashAttention-2 (Dao, 2023) implementation. We consider sequence lengths ranging from 1,024 to 8,192 and apply *Stochastic Length* (SL) during training. In the evaluation, we use the same configuration for HSTU and Transformer ($d = 512$, $h = 8$, $d_{qk} = 64$) and ablate *relative attention bias* considering HSTU outperforms Transformers without rab^{p,t}, as demonstrated in Section 4.1.2. We compare the encoder-level performance in bfloat16 on NVIDIA H100 GPUs. Overall, HSTU is up to 15.2x and 5.6x more efficient than Transformers in training and inference, respectively.

Additionally, the decrease in activation memory usage as discussed in Section 3.3 enables us to construct over 2x deeper networks with HSTUs compared to Transformers.

Table 6. Offline/Online Comparison of Retrieval Models.

Methods	Offline HR@K		Online metrics	
	K=100	K=500	E-Task	C-Task
DLRM	29.0%	55.5%	+0%	+0%
DLRM (abl. features)	28.3%	54.3%	–	
GR (content-based)	11.6%	18.8%	–	
GR (interactions only)	35.6%	61.7%	–	
GR (new source)	**36.9%**	**62.4%**	**+6.2%**	**+5.0%**
GR (replace source)			**+5.1%**	**+1.9%**

Table 7. Offline/Online Comparison of Ranking Models.

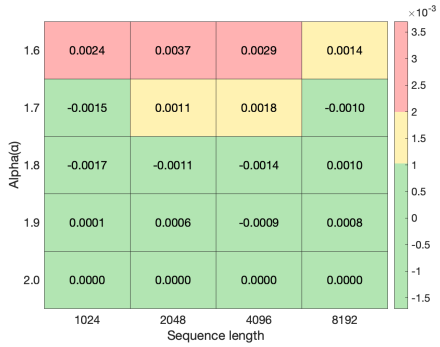
Methods	Offline NEs		Online metrics	
	E-Task	C-Task	E-Task	C-Task
DLRM	.4982	.7842	+0%	+0%
DLRM (DIN+DCN)	.5053	.7899	–	–
DLRM (abl. features)	.5053	.7925	–	–
GR (interactions only)	.4851	.7903	–	–
GR	**.4845**	**.7645**	**+12.4%**	**+4.4%**

4.3. Generative Recommenders vs DLRMs in Industrial-scale Streaming Settings

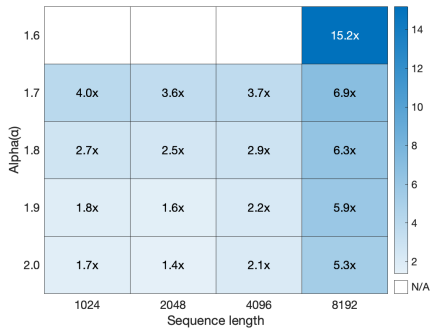
Lastly, we compare the end-to-end performance of GRs against state-of-the-art DLRM baselines in industrial-scale streaming settings. Our GR implementation reflects a typical configuration used in production, whereas the DLRM settings reflect iterations of hundreds of people over multiple years. Given multiple generators are used in the retrieval stage of a recommendation system, we report both the online result for adding GR ("add source") and replacing existing main DLRM source ("replace source"). Table 6 and Table 7 show that GR not only significantly outperforms DLRMs offline, but also brings 12.4% wins in A/B tests.

As discussed in Section 2, GRs build upon raw categorical engagement features, while DLRMs are typically trained with a significantly larger number of features, the majority of which are handcrafted from raw signals. If we give the same set of features used in GRs to DLRMs ("DLRM (abl. features)"), the performance of DLRMs is significantly degraded, which suggests GRs can meaningfully capture those features via their architecture and unified feature space.

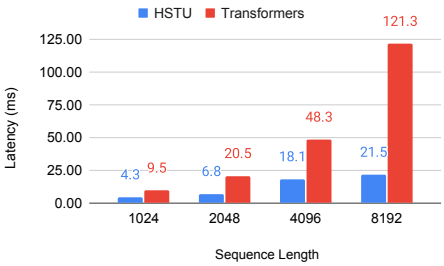
We further validate the GR formulation in Section 2.2 by comparing it with a traditional sequential recommender setup that only considers items user interacted with (Kang



(a) Training NE.



(b) Training Speedup.



(c) Inference Speedup.

Figure 5. Encoder-level efficiency: HSTU vs FlashAttention2-based Transformers for Training (a, b) and Inference (c).

& McAuley, 2018) ("GR (interactions only)"). The results are significantly worse, with its ranking variant underperforming GRs by 2.6% in NE in the main consumption task.

Considering the popularity of content-based methods (including LMs), we also include a GR baseline with only content features ("GR (content-based)"). The substantial gap in performance of content-based baselines and DLRMs/GRs underscores the significance of high cardinality user actions.

We finally compare the efficiency of GRs with our production DLRMs in Figure 6. Despite the GR model being 285x more computationally complex, we achieved 1.50x/2.99x *higher* QPS when scoring 1024/16384 candidates, due to HSTU and the novel M-FALCON algorithm in Section 3.4.

4.3.1. Scaling Law for Recommendation Systems

It is commonly known that in large-scale industrial settings, DLRMs saturate in quality at certain compute and params

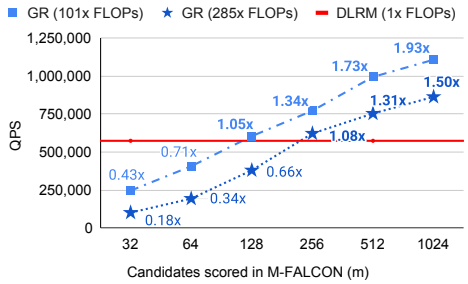


Figure 6. Comparison of inference throughput, in the most challenging ranking setup. Full results can be found in Appendix H.1.

regimes (Zhao et al., 2023). We compare the scalability of GRs and DLRMs to better understand this phenomenon.

Since feature interaction layers are crucial for DLRM's performance (Mudigere et al., 2022), we experimented with Transformers (Vaswani et al., 2017), DHEN (Zhang et al., 2022), and a variant of DCN (Wang et al., 2021) augmented with residual connections (He et al., 2015) used in our production settings to scale up the DLRM baseline in the ranking setting. For the retrieval baseline, given our baseline used a residual setup, we scaled up hidden layer sizes, embedding dimensions, and number of layers. For HSTU-based Generative Recommenders (GRs), we scaled up the model by adjusting the hyperparameters for HSTU, including the number of residual layers, sequence length, embedding dimensions, number of attention heads, etc. We additionally adjust the number of negatives for retrieval.

Results are shown in Figure 7. In the low compute regime, DLRMs might outperform GRs due to handcrafted features, corroborating the importance of feature engineering in traditional DLRMs. However, GRs demonstrate substantially better scalability with respect to FLOPs, whereas DLRM performance plateaus, consistent with findings in prior work. We also observe better scalability w.r.t. both embedding parameters and non-embedding parameters, with GRs leading to 1.5 trillion parameter models, whereas DLRMs performance saturate at about 200 billion parameters.

Finally, all of our main metrics, including Hit Rate@100 and Hit Rate@500 for retrieval, and NE for ranking, empirically scale as a power law of compute used given appropriate hyperparameters. We observe this phenomenon across three orders of magnitude, up till the largest models we were able to test (8,192 sequence length, 1,024 embedding dimension, 24 layers of HSTU), at which point the total amount of compute we used (normalized over 365 days as we use a standard streaming training setting) is close to the total training compute used by GPT-3 (Brown et al., 2020) and LLaMa-2 (Touvron et al., 2023b), as shown in Figure 1. Within a reasonable range, the exact model hyperparameters play less important roles compared to the total amount of training compute applied. In contrast to language modeling (Kaplan et al., 2020), sequence length play a significantly more important role in GRs, and it's important to scale up sequence

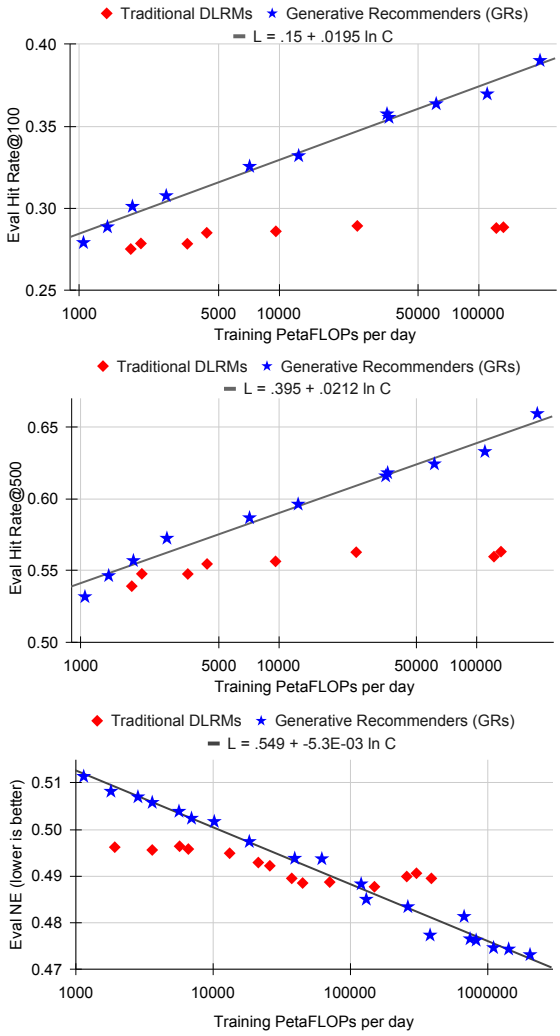


Figure 7. Scalability: DLRMs vs GRs in large-scale industrial settings across retrieval (top, middle) and ranking (bottom). +0.005 in HR and -0.001 in NE represent significant improvements.

length and other parameters in tandem. This is perhaps the most important advantage of our proposed method, as we've shown for the first time that scaling law from LLMs may also apply to large-scale recommendation systems.

5. Related Work

Prior work on sequential recommenders reduces user interactions to a single homogeneous sequence over items (Hidasi et al., 2016; Kang & McAuley, 2018). Industrial-scale applications of sequential approaches are primarily pairwise attention (Zhou et al., 2018) or sequential encoders as part of DLRMs (Chen et al., 2019; Xia et al., 2023). Multi-stage attention has been explored in lieu of self-attention to improve efficiency (Chang et al., 2023). Generative approaches that represent ids as a token series have been explored in retrieval (Zhuo et al., 2020). We give a more extensive discussion of prior work in Appendix B.1.

Efficient attention has been a major research focus area due to self-attention's $O(n^2)$ scaling factor, with major work like factorized attentions (Child et al., 2019), low-rank approximations (Katharopoulos et al., 2020), etc. Recently, alternative formulations for sequential transduction settings have been explored (Gu et al., 2022; Hua et al., 2022). HSTU's elementwise gating design, in particular, is inspired by FLASH (Hua et al., 2022). Recent hardware-aware formulations have been shown to significantly reduce memory usage (Rabe & Staats, 2021; Korthikanti et al., 2022; Zhai et al., 2023b) and give significantly better wallclock time results (Dao et al., 2022). Length extrapolation enables models trained on shorter sequences to generalize, though most work focuses on finetuning or improving bias mechanisms (Press et al., 2022). Our work instead introduces stochasticity in the length dimension, inspired by work on stochasticity in the depth dimension (Huang et al., 2016).

Interests in large language models (LLMs) have motivated work to treat various recommendation tasks as in-context learning (Sileo et al., 2022), instruction tuning (Bao et al., 2023), or transfer learning (Li et al., 2023) on top of pre-trained LLMs. World knowledge embedded in LLMs can be transferred to downstream tasks (Cui et al., 2022) and improve recommendations in zero-shot or few-shot cases. Textual representations of user behavior sequences have also demonstrated good scaling behaviors on medium-scale datasets (Shin et al., 2023). Most studies of LLMs for recommendation have been centered around low-data regimes; in large-scale settings, they have yet to outperform collaborative filtering on MovieLens (Hou et al., 2024).

6. Conclusions

We have proposed Generative Recommenders (GRs), a new paradigm that formulates ranking and retrieval as sequential transduction tasks, allowing them to be trained in a generative manner. This is made possible by the novel HSTU encoder design, which is 5.3x-15.2x faster than state-of-the-art Transformers on 8192 length sequences, and through the use of new training and inference algorithms such as M-FALCON. With GRs, we deployed models that are *285x* more complex while using *less* inference compute. GRs and HSTU have led to 12.4% metric improvements in production and have shown superior scaling performance compared to traditional DLRMs. Our results corroborate that user actions represent an underexplored modality in generative modeling – to echo our title, *"Actions speak louder than words"*.

The dramatic simplification of features in our work paves the way for the first foundation models for recommendations, search, and ads by enabling a unified feature space to be used across domains. The fully sequential setup of GRs also enables recommendation to be formulated in an end-to-end, generative setting. Both of these enable recommendation systems to better assist users holistically.

Impact Statement

We believe that our work has broad positive implications. Reducing reliance of recommendation, search, and ads systems on the large number of heterogeneous features can make these systems much more privacy friendly while improving user experiences. Enabling recommendation systems to attribute users' long term outcomes to short-term decisions via fully sequential formulations could reduce the prevalence of content that do not serve users' long term goals (including clickbaits and fake news) across the web, and better align incentives of platforms with user values. Finally, applications of foundation models and scaling law can help reduce carbon footprints incurred with model research and developments needed for recommendations, search, and related use cases.

Acknowledgements

This work represents the joint efforts of hundreds of people, and would not be possible without work from the following contributors (alphabetical order): Adnan Akhundov, Bugra Akyildiz, Shabab Ayub, Alex Bao, Renqin Cai, Jennifer Cao, Guoqiang Jerry Chen, Lei Chen, Sean Chen, Xianjie Chen, Huihui Cheng, Weiwei Chu, Ted Cui, Shiyan Deng, Nimit Desai, Fei Ding, Francois Fagan, Lu Fang, Liang Guo, Liz Guo, Jeevan Gyawali, Yuchen Hao, Daisy Shi He, Samuel Hsia, Jie Hua, Yanzun Huang, Hongyi Jia, Rui Jian, Jian Jin, Rahul Kindi, Changkyu Kim, Yejin Lee, Fu Li, Hong Li, Shen Li, Wei Li, Zhijing Li, Xueting Liao, Emma Lin, Hao Lin, Jingzhou Liu, Xingyu Liu, Kai Londenberg, Liang Luo, Linjian Ma, Matt Ma, Yun Mao, Bert Maher, Matthew Murphy, Satish Nadathur, Min Ni, Jongsoo Park, Jing Qian, Lijing Qin, Alex Singh, Timothy Shi, Dennis van der Staay, Xiao Sun, Colin Taylor, Shin-Yeh Tsai, Rohan Varma, Omkar Vichare, Alyssa Wang, Pengchao Wang, Shengzhi Wang, Wenting Wang, Xiaolong Wang, Zhiyong Wang, Wei Wei, Bin Wen, Carole-Jean Wu, Eric Xu, Bi Xue, Zheng Yan, Chao Yang, Junjie Yang, Zimeng Yang, Chunxing Yin, Daniel Yin, Yiling You, Keke Zhai, Yanli Zhao, Zhuoran Zhao, Hui Zhang, Jingjing Zhang, Lu Zhang, Lujia Zhang, Na Zhang, Rui Zhang, Xiong Zhang, Ying Zhang, Zhiyun Zhang, Charles Zheng, Erheng Zhong, Xin Zhuang. We would like to thank Shikha Kapoor, Rex Cheung, Lana Dam, Ram Ramanathan, Nipun Mathur, Bo Feng, Yanhong Wu, Zhaohui Guo, Hongjie Bai, Zellux Wang, Arun Singh, Bruce Deng, Yisong Song, Haotian Wu, Meihong Wang for product support, and Joseph Laria, Akshay Hegde, Abha Jain, Raj Ganapathy for assistance with program management. Finally, we would like to thank Ajit Mathews, Shilin Ding, Hong Yan, Lars Backstrom for their leadership support, and insightful discussions with Andrew Tulloch, Liang Xiong, Kaushik Veeraraghavan, and Gaofeng Zhao.

References

Bao, K., Zhang, J., Zhang, Y., Wang, W., Feng, F., and He, X. Tallrec: An effective and efficient tuning framework to align large language model with recommendation. In *Proceedings of the 17th ACM Conference on Recommender Systems*, RecSys '23. ACM, September 2023. doi: 10.1145/3604915.3608857. URL http://dx.doi.org/10.1145/3604915.3608857.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. 2020.

Chang, J., Zhang, C., Fu, Z., Zang, X., Guan, L., Lu, J., Hui, Y., Leng, D., Niu, Y., Song, Y., and Gai, K. Twin: Two-stage interest network for lifelong user behavior modeling in ctr prediction at kuaishou, 2023.

Chen, Q., Zhao, H., Li, W., Huang, P., and Ou, W. 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*, DLP-KDD '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367837. doi: 10.1145/3326937.3341261. URL https://doi.org/10.1145/3326937.3341261.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20, 2020.

Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., Anil, R., Haque, Z., Hong, L., Jain, V., Liu, X., and Shah, H. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, pp. 7–10, 2016. ISBN 9781450347952.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers. *CoRR*, abs/1904.10509, 2019. URL http://arxiv.org/abs/1904.10509.

Covington, P., Adams, J., and Sargin, E. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pp. 191–198, 2016. ISBN 9781450340359.

Cui, Z., Ma, J., Zhou, C., Zhou, J., and Yang, H. M6-rec: Generative pretrained language models are open-ended recommender systems, 2022.

Dallmann, A., Zoller, D., and Hotho, A. A case study on sampling strategies for evaluating neural sequential item recommendation models. In *Proceedings of the 15th ACM Conference on Recommender Systems*, RecSys '21, pp. 505–514, 2021. ISBN 9781450384582.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning, 2023.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. In Burstein, J., Doran, C., and Solorio, T. (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pp. 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-1423. URL https://doi.org/10.18653/v1/n19-1423.

Eksombatchai, C., Jindal, P., Liu, J. Z., Liu, Y., Sharma, R., Sugnet, C., Ulrich, M., and Leskovec, J. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pp. 1775–1784, 2018. ISBN 9781450356398.

Elfwing, S., Uchibe, E., and Doya, K. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *CoRR*, abs/1702.03118, 2017. URL http://arxiv.org/abs/1702.03118.

Gao, W., Fan, X., Wang, C., Sun, J., Jia, K., Xiao, W., Ding, R., Bin, X., Yang, H., and Liu, X. Learning an end-to-end structure for retrieval in large-scale recommendations. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, CIKM '21, pp. 524–533, 2021. ISBN 9781450384469.

Gillenwater, J., Kulesza, A., Fox, E., and Taskar, B. Expectation-maximization for learning determinantal point processes. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pp. 3149–3157, Cambridge, MA, USA, 2014. MIT Press.

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=uYLFoz1vlAC.

Guo, H., Tang, R., Ye, Y., Li, Z., and He, X. Deepfm: A factorization-machine based neural network for ctr prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, pp. 1725–1731, 2017. ISBN 9780999241103.

Gupta, M. R., Bengio, S., and Weston, J. Training highly multiclass classifiers. *J. Mach. Learn. Res.*, 15(1): 1461–1492, jan 2014. ISSN 1532-4435.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

He, X., Pan, J., Jin, O., Xu, T., Liu, B., Xu, T., Shi, Y., Atallah, A., Herbrich, R., Bowers, S., and Candela, J. Q. Practical lessons from predicting clicks on ads at facebook. In *ADKDD'14: Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450329996.

Hidasi, B., Karatzoglou, A., Baltrunas, L., and Tikk, D. Session-based recommendations with recurrent neural networks. In Bengio, Y. and LeCun, Y. (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.06939.

Hou, Y., Zhang, J., Lin, Z., Lu, H., Xie, R., McAuley, J., and Zhao, W. X. Large language models are zero-shot rankers for recommender systems. In *Advances in Information Retrieval - 46th European Conference on IR Research, ECIR 2024*, 2024.

Hua, W., Dai, Z., Liu, H., and Le, Q. V. Transformer quality in linear time. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvári, C., Niu, G., and Sabato, S. (eds.), *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9099–9117. PMLR, 2022. URL https://proceedings.mlr.press/v162/hua22a.html.

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Deep networks with stochastic depth, 2016.

Jegou, H., Douze, M., and Schmid, C. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal.*

Mach. Intell., 33(1):117–128, jan 2011. ISSN 0162-8828. doi: 10.1109/TPAMI.2010.57. URL https://doi.org/10.1109/TPAMI.2010.57.

Kang, W.-C. and McAuley, J. Self-attentive sequential recommendation. In *2018 International Conference on Data Mining (ICDM)*, pp. 197–206, 2018.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020. URL https://arxiv.org/abs/2001.08361.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

Khudia, D., Huang, J., Basu, P., Deng, S., Liu, H., Park, J., and Smelyanskiy, M. Fbgemm: Enabling high-performance low-precision deep learning inference. *arXiv preprint arXiv:2101.05615*, 2021.

Klenitskiy, A. and Vasilev, A. Turning dross into gold loss: is bert4rec really better than sasrec? In *Proceedings of the 17th ACM Conference on Recommender Systems*, RecSys '23, pp. 1120–1125, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702419. doi: 10.1145/3604915.3610644. URL https://doi.org/10.1145/3604915.3610644.

Korthikanti, V., Casper, J., Lym, S., McAfee, L., Andersch, M., Shoeybi, M., and Catanzaro, B. Reducing activation recomputation in large transformer models, 2022.

Li, C., Chang, E., Garcia-Molina, H., and Wiederhold, G. Clustering for approximate similarity search in high-dimensional spaces. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):792–808, 2002.

Li, J., Wang, M., Li, J., Fu, J., Shen, X., Shang, J., and McAuley, J. Text is all you need: Learning language representations for sequential recommendation. In *KDD*, 2023.

Liu, Z., Zou, L., Zou, X., Wang, C., Zhang, B., Tang, D., Zhu, B., Zhu, Y., Wu, P., Wang, K., and Cheng, Y. Monolith: Real time recommendation system with collisionless embedding table, 2022.

Ma, J., Zhao, Z., Yi, X., Chen, J., Hong, L., and Chi, E. H. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. KDD '18, 2018.

Mudigere, D., Hao, Y., Huang, J., Jia, Z., Tulloch, A., Sridharan, S., Liu, X., Ozdal, M., Nie, J., Park, J., Luo, L., Yang, J. A., Gao, L., Ivchenko, D., Basant, A., Hu, Y., Yang, J., Ardestani, E. K., Wang, X., Komuravelli, R., Chu, C.-H., Yilmaz, S., Li, H., Qian, J., Feng, Z., Ma, Y., Yang, J., Wen, E., Li, H., Yang, L., Sun, C., Zhao, W., Melts, D., Dhulipala, K., Kishore, K., Graf, T., Eisenman, A., Matam, K. K., Gangidi, A., Chen, G. J., Krishnan, M., Nayak, A., Nair, K., Muthiah, B., khorashadi, M., Bhattacharya, P., Lapukhov, P., Naumov, M., Mathews, A., Qiao, L., Smelyanskiy, M., Jia, B., and Rao, V. 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*, ISCA '22, pp. 993–1011, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450386104. doi: 10.1145/3470496.3533727. URL https://doi.org/10.1145/3470496.3533727.

Peng, B., Quesnelle, J., Fan, H., and Shippole, E. YaRN: Efficient context window extension of large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=wHBfxhZu1u.

Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Levskaya, A., Heek, J., Xiao, K., Agrawal, S., and Dean, J. Efficiently scaling transformer inference, 2022.

Press, O., Smith, N. A., and Lewis, M. Train short, test long: Attention with linear biases enables input length extrapolation. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=R8sQPpGCv0.

Rabe, M. N. and Staats, C. Self-attention does not need $o(n^2)$ memory, 2021.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1), jan 2020. ISSN 1532-4435.

Rendle, S. Factorization machines. In *2010 IEEE International Conference on Data Mining (ICDM)*, pp. 995–1000, 2010. doi: 10.1109/ICDM.2010.127.

Rendle, S., Krichene, W., Zhang, L., and Anderson, J. Neural collaborative filtering vs. matrix factorization revisited. In *Fourteenth ACM Conference on Recommender Systems (RecSys'20)*, pp. 240–248, 2020. ISBN 9781450375832.

Shazeer, N. Glu variants improve transformer, 2020.

Shin, K., Kwak, H., Kim, S. Y., Ramström, M. N., Jeong, J., Ha, J.-W., and Kim, K.-M. Scaling law for recommendation models: towards general-purpose user representations. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press, 2023. ISBN 978-1-57735-880-0. doi: 10.1609/aaai.v37i4.25582. URL https://doi.org/10.1609/aaai.v37i4.25582.

Shrivastava, A. and Li, P. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *Advances in Neural Information Processing Systems*, volume 27, 2014.

Sileo, D., Vossen, W., and Raymaekers, R. Zero-shot recommendation as language modeling. In Hagen, M., Verberne, S., Macdonald, C., Seifert, C., Balog, K., Nørvåg, K., and Setty, V. (eds.), *Advances in Information Retrieval - 44th European Conference on IR Research, ECIR 2022, Stavanger, Norway, April 10-14, 2022, Proceedings, Part II*, volume 13186 of *Lecture Notes in Computer Science*, pp. 223–230. Springer, 2022. doi: 10.1007/978-3-030-99739-7_26. URL https://doi.org/10.1007/978-3-030-99739-7_26.

Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding, 2023.

Sun, F., Liu, J., Wu, J., Pei, C., Lin, X., Ou, W., and Jiang, P. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, pp. 1441–1450, 2019. ISBN 9781450369763.

Tang, H., Liu, J., Zhao, M., and Gong, X. Progressive layered extraction (ple): A novel multi-task learning (mtl) model for personalized recommendations. In *Proceedings of the 14th ACM Conference on Recommender Systems*, RecSys '20, pp. 269–278, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450375832. doi: 10.1145/3383313.3412236. URL https://doi.org/10.1145/3383313.3412236.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models, 2023a.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., and Scialom, T. Llama 2: Open foundation and fine-tuned chat models, 2023b.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6000–6010, 2017. ISBN 9781510860964.

Wang, R., Shivanna, R., Cheng, D., Jain, S., Lin, D., Hong, L., and Chi, E. Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the Web Conference 2021*, WWW '21, pp. 1785–1797, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383127. doi: 10.1145/3442381.3450078. URL https://doi.org/10.1145/3442381.3450078.

Wang, Z., Zhao, L., Jiang, B., Zhou, G., Zhu, X., and Gai, K. Cold: Towards the next generation of pre-ranking system, 2020.

Xia, X., Eksombatchai, P., Pancha, N., Badani, D. D., Wang, P.-W., Gu, N., Joshi, S. V., Farahpour, N., Zhang, Z., and Zhai, A. 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*, KDD '23, pp. 5249–5259, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701030. doi: 10.1145/3580305.3599918. URL https://doi.org/10.1145/3580305.3599918.

Xiao, J., Ye, H., He, X., Zhang, H., Wu, F., and Chua, T.-S. Attentional factorization machines: Learning the weight of feature interactions via attention networks. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, pp. 3119–3125. AAAI Press, 2017. ISBN 9780999241103.

Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., and Liu, T.-Y. On layer normalization in the transformer architecture. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

Yang, J., Yi, X., Zhiyuan Cheng, D., Hong, L., Li, Y., Xiaoming Wang, S., Xu, T., and Chi, E. H. Mixed negative sampling for learning two-tower neural networks in recommendations. In *Companion Proceedings of the Web Conference 2020*, WWW '20, pp. 441–447, 2020. ISBN 9781450370240.

Zhai, J., Lou, Y., and Gehrke, J. Atlas: A probabilistic algorithm for high dimensional similarity search. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pp. 997–1008, 2011. ISBN 9781450306614.

Zhai, J., Gong, Z., Wang, Y., Sun, X., Yan, Z., Li, F., and Liu, X. Revisiting neural retrieval on accelerators. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, pp. 5520–5531, New York, NY, USA, 2023a. Association for Computing Machinery. ISBN 9798400701030. doi: 10.1145/3580305.3599897. URL https://doi.org/10.1145/3580305.3599897.

Zhai, Y., Jiang, C., Wang, L., Jia, X., Zhang, S., Chen, Z., Liu, X., and Zhu, Y. Bytetransformer: A high-performance transformer boosted for variable-length inputs. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 344–355, Los Alamitos, CA, USA, may 2023b. IEEE Computer Society. doi: 10.1109/IPDPS54959.2023.00042. URL https://doi.ieeecomputersociety.org/10.1109/IPDPS54959.2023.00042.

Zhang, B., Luo, L., Liu, X., Li, J., Chen, Z., Zhang, W., Wei, X., Hao, Y., Tsang, M., Wang, W., Liu, Y., Li, H., Badr, Y., Park, J., Yang, J., Mudigere, D., and Wen, E. Dhen: A deep and hierarchical ensemble network for large-scale click-through rate prediction, 2022.

Zhao, X., Xia, L., Zhang, L., Ding, Z., Yin, D., and Tang, J. Deep reinforcement learning for page-wise recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, pp. 95–103, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450359016. doi: 10.1145/3240323.3240374. URL https://doi.org/10.1145/3240323.3240374.

Zhao, Z., Yang, Y., Wang, W., Liu, C., Shi, Y., Hu, W., Zhang, H., and Yang, S. Breaking the curse of quality saturation with user-centric ranking, 2023.

Zhou, G., Zhu, X., Song, C., Fan, Y., Zhu, H., Ma, X., Yan, Y., Jin, J., Li, H., and Gai, K. Deep interest network for click-through rate prediction. KDD '18, 2018.

Zhou, K., Wang, H., Zhao, W. X., Zhu, Y., Wang, S., Zhang, F., Wang, Z., and Wen, J.-R. S3-rec: Self-supervised learning for sequential recommendation with mutual information maximization. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, pp. 1893–1902, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368599. doi: 10.1145/3340531.3411954. URL https://doi.org/10.1145/3340531.3411954.

Zhuo, J., Xu, Z., Dai, W., Zhu, H., Li, H., Xu, J., and Gai, K. Learning optimal tree models under beam search. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

A. Notations

We summarize key notations used in this paper in Table 8 and Table 9.

Symbol	Description
$\Psi_k(t_j)$	The k-th training example (k is ordered globally) emitted by the feature logging system at time t_j. In a typical DLRM recommendation system, after the user consumes some content Φ_i (by responding with an action a_i such as skip, video completion and share), the feature logging system joins the tuple (Φ_i, a_i) with the features used to rank Φ_i, and *emits* $(\Phi_i, a_i,$ features for $\Phi_i)$ as a training example $\Psi_k(t_j)$. As discussed in Section 2.3, DLRMs and GRs deal with different numbers of training examples, with the number of examples in GRs typically being 1-2 orders of magnitude smaller.
n_c $(n_{c,i})$	Number of contents that user has interacted with (of user/sample i).
$\Phi_0, \ldots, \Phi_{n_c-1}$	List of contents that a user has interacted with, in the context of a recommendation system.
$a_0, \ldots, a_{n_c-1}$	List of user actions corresponding to Φ_is. When all predicted events are binary, each action can be considered a multi-hot vector over (atomic) events such as like, share, comment, image view, video initialization, video completion, hide, etc.
E, F	Categorical features in DLRMs, in Figure 2. $E_0, E_1, \ldots, E_7, E_8$, and $F_0, F_1, \ldots, F_7$ represent transformations of $(\Phi_0, a_0, t_0), \ldots, (\Phi_{n_c-1}, a_{n_c-1}, t_{n_c-1})$ obtained at various points in time via feature extraction (e.g., most recent 10 liked images, most similar 50 urls that the user clicked on in the past compared to the current candidate, etc.). "merge & sequentialize" denotes the (virtual) reverse process of obtaining the raw engagement series $(\Phi_0, a_0, t_0), \ldots, (\Phi_{n_c-1}, a_{n_c-1}, t_{n_c-1})$.
G, H	Categorical features in DLRMs, in Figure 2 that are not related to user-content engagements. These features (e.g., demographics or followed creators) are merged into the main time series (list of contents user engaged with, e.g., $\Phi_0, a_0, \ldots, \Phi_{n_c-1}, a_{n_c-1}$), as discussed in Section 2.1 and illustrated in Figure 2.
n (n_i)	Number of tokens in the sequential transduction task (of user/sample i). While $O(n) = O(n_c)$, n can differ from n_c even without any non-interaction related categorical features; see e.g., Table 1.
$x_0, \ldots, x_{n-1}$	List of input tokens in the sequential transduction task.
$y_0, \ldots, y_{n-1}$	List of output tokens in the sequential transduction task.
$t_0, \ldots, t_{n-1}$	List of timestamps corresponding to when $x_0, \ldots, x_{n-1}$ were observed.
$\mathbb{X}$, $\mathbb{X}_c$	Vocabulary of all input/output tokens ($\mathbb{X}$) and its content subset ($\mathbb{X}_c$).
N, N_c	$\max_i n_i$, $\max_i n_{c,i}$.
u_t	User representation at time t.
$s_u(n_i)$, $\hat{s}_u(n_i)$	Sampling rate for user i, used in generative training (Section 2.3).
d	Model dimension (embedding dimension).
d_{qk}	Attention dimension size in HSTU and Transformers. This applies to $Q(X)$ and $K(X)$ in Equation (1).
d_v	Value dimension size in HSTU. For Transformers, we typically have $d_{qk} = d_v$.
d_{ff}	Hidden dimension size in pointwise feedforward layers of Transformers. HSTU does not utilize feedforward layers; see $U(X)$ below.
h	Number of attention heads.
l	Number of layers in HSTU. For Transformers, attention and pointwise feedforward layers together constitute a layer.

Table 8. Table of Notations (continued on the next page).

B. Generative Recommenders: Background and Formulations

Many readers are likely more familiar with classical Deep Learning Recommendation Models (DLRMs) (Mudigere et al., 2022) given its popularity from YouTube DNN days (Covington et al., 2016) and its widespread usage in every single large online content and e-commerce platform (Cheng et al., 2016; Zhou et al., 2018; Wang et al., 2021; Chang et al., 2023; Xia et al., 2023; Zhai et al., 2023a). DLRMs operate on top of heterogeneous feature spaces using various neural

Symbol	Description
X	Input to an HSTU layer. In standard terminology (before batching), $X \in \mathbb{R}^{N \times d}$ assuming we have a input sequence containing N tokens.
$Q(X), K(X), V(X)$	Query, key, value in HSTU obtained for a given input X based on Equation (1). The definition is similar to Q, K, and V in standard Transformers. $Q(X), K(X) \in \mathbb{R}^{h \times N \times d_{qk}}$, and $V(X) \in \mathbb{R}^{h \times N \times d_v}$.
$U(X)$	HSTU uses $U(X)$ to "gate" attention-pooled values ($V(X)$) in Equation (3), which together with $f_2(\cdot)$, enables HSTU to avoid feedforward layers altogether. $U(X) \in \mathbb{R}^{h \times N \times d_v}$.
$A(X)$	Attention tensor obtained for input X. $A(X) \in \mathbb{R}^{h \times N \times N}$.
$Y(X)$	Output of a HSTU layer obtained for the input X. $Y(X) \in \mathbb{R}^d$.
Split$(\cdot)$	The operation that splits a tensor into chunks. $\phi_1(f_1(X))) \in \mathbb{R}^{N \times (2hd_{qk} + 2hd_v)}$ in Equation (1); we obtain $U(X)$, $V(X)$ (both of shape $h \times N \times d_v$), $Q(X)$, $K(X)$ (both of shape $h \times N \times d_{qk}$) by splitting the larger tensor (and permuting dimensions) with $U(X), V(X), Q(X), K(X) = \text{Split}(\phi_1(f_1(X)))$.
$\text{rab}^{p,t}$	relative attention bias that incorporates both positional (Raffel et al., 2020) and temporal information (based on the time when the tokens are observed, $t_0, \ldots, t_{n-1}$; one possible implementation is to apply some bucketization function to $(t_j - t_i)$ for (i, j)). In practice, we share $\text{rab}^{p,t}$ across different attention heads within a layer, hence $\text{rab}^{p,t} \in \mathbb{R}^{1 \times N \times N}$.
α	Parameter controlling sparsity in the *Stochastic Length* algorithm used in HSTU (Section 3.2).
R	Register size on GPUs, in the context of the HSTU algorithm discussed in Section 3.2.
m	Number of candidates considered in a recommendation system's ranking stage.
b_m	Microbatch size, in the M-FALCON algorithm discussed in Section 3.4.

Table 9. Table of Notations (continued)

networks including feature interaction modules (Guo et al., 2017; Xiao et al., 2017; Wang et al., 2021), sequential pooling or target-aware pairwise attention modules (Hidasi et al., 2016; Zhou et al., 2018; Chang et al., 2023) and advanced multi-expert multi-task modules (Ma et al., 2018; Tang et al., 2020). We hence provided an overview of Generative Recommenders (GRs) by contrasting them with classical DLRMs explicitly in Section 2 and Section 3. In this section, we give the readers an alternative perspective starting from the classical sequential recommender literature.

B.1. Background: Sequential Recommendations in Academia and Industry

B.1.1. Academic Research (Traditional Sequential Recommender Settings)

Recurrent neural networks (RNNs) were first applied to recommendation scenarios in GRU4Rec (Hidasi et al., 2016). Hidasi et al. (2016) considered Gated Recurrent Units (GRUs) and applied them over two datasets, RecSys Challenge 2015 [2] and VIDEO (a proprietary dataset). In both cases, *only positive events (clicked e-commerce items or videos where users spent at least a certain amount of time watching)* were kept as part of the input sequence. We further observe that in a classical industrial-scale two-stage recommendation system setup consisting of retrieval and ranking stages (Covington et al., 2016), the task that Hidasi et al. (2016) solved primarily maps to the retrieval task.

Transformers, sequential transduction architectures, and their variants. Advances in sequential transduction architectures in later years, in particular Transformers (Vaswani et al., 2017), have motivated similar advancements in recommendation systems. SASRec (Kang & McAuley, 2018) first applied Transformers in an autoregressive setting. They considered the *presence* of a review or rating as *positive* feedback, thereby converting classical datasets like Amazon Reviews [3] and MovieLens [4] *to sequences of positive items*, similar to GRU4Rec. A binary cross entropy loss was employed, where positive target is defined as the next "positive" item (recall this is in essence just presence of a review or rating), and negative target is randomly sampled from the item corpus $\mathbb{X} = \mathbb{X}_c$.

[2] http://2015.recsyschallenge.com/

[3] https://jmcauley.ucsd.edu/data/amazon/

[4] https://grouplens.org/datasets/movielens/1m/, https://grouplens.org/datasets/movielens/20m/

Most subsequent research were built upon similar settings as GRU4Rec (Hidasi et al., 2016) and SASRec (Kang & McAuley, 2018) discussed above, such as BERT4Rec (Sun et al., 2019) applying bidirectional encoder setting from BERT (Devlin et al., 2019), S3Rec (Zhou et al., 2020) introducing an explicit pre-training stage, and so on.

B.1.2. Industrial Applications as part of Deep Learning Recommendation Models (DLRMs).

Sequential approaches, including sequential encoders and pairwise attention modules, have been widely applied in industrial settings due to their ability to enhance user representations as part of DLRMs. DLRMs commonly use relatively small sequence lengths, such as 20 in BST (Chen et al., 2019), 1,000 in DIN (Zhou et al., 2018), and 100 in TransAct (Xia et al., 2023). We observe that these are 1-3 orders of magnitude smaller compared with 8,192 in this work (Section 4.3).

Despite using short sequence lengths, most DLRMs can successfully capture long-term user preferences. This can be attributed to two key aspects. First, precomputed user profiles/embeddings (Xia et al., 2023) or external vector stores (Chang et al., 2023) are commonly used in modern DLRMs, both of which effectively extend lookback windows. Second, a significant number of contextual-, user-, and item-side features were generally employed (Zhou et al., 2018; Chen et al., 2019; Chang et al., 2023; Xia et al., 2023) and various heterogeneous networks, such as FMs (Xiao et al., 2017; Guo et al., 2017), DCNs (Wang et al., 2021), MoEs, etc. are used to transform representations and combine outputs.

In contrast to sequential settings discussed in Appendix B.1.1, all major industrial work defines loss over (user/request, candidate item) pairs. In the ranking setting, a multi-task binary cross-entropy loss is commonly used. In the retrieval setting, two tower setting (Covington et al., 2016) remains the dominant approach. Recent work has investigated representing the next item to recommend as a probability distribution over a sequence of (sub-)tokens, such as OTM (Zhuo et al., 2020), and DR (Gao et al., 2021) (note that in other recent work, the same setting is sometimes denoted as "generative retrieval"). They commonly utilize beam search to decode the item from sub-tokens. Advanced learned similarity functions, such as mixture-of-logits (Zhai et al., 2023a), have also been proposed and deployed as an alternative to two-tower setting and beam search given proliferation of modern accelerators such as GPUs, custom ASICs, and TPUs.

From a problem formulation perspective, we consider all work discussed above part of DLRMs (Mudigere et al., 2022) given the model architectures, features used, and losses used differ significantly from academic sequential recommender research discussed in Appendix B.1.1. It's also worth remarking that there have been no successful applications of fully sequential ranking settings in industry, especially not at billion daily active users (DAU) scale, prior to this work.

B.2. Formulations: Ranking and Retrieval as Sequential Transduction Tasks in Generative Recommenders (GRs)

We next discuss three limitations in the traditional sequential recommender settings and DLRM settings, and how Generative Recommenders (GRs) address them from a problem formulation perspective.

Ignorance of features other than user-interacted items. Past sequential formulations only consider contents (items) users explicitly interacted with (Hidasi et al., 2016; Kang & McAuley, 2018; Sun et al., 2019; Zhou et al., 2020), while industry-scale recommendation systems prior to GRs are trained over a vast number of features to enhance the representation of users and contents (Covington et al., 2016; Cheng et al., 2016; Zhou et al., 2018; Chen et al., 2019; Chang et al., 2023; Xia et al., 2023; Zhai et al., 2023a). GR addresses this limitation by a) compressing other categorical features and merging them with the main time series, and b) capturing numerical features through cross-attention interaction utilizing a target-aware formulation as discussed in Section 2.1 and Figure 2. We validate this by showing that the traditional "interaction-only" formulation that ignores such features degrades model quality significantly; experiment results can be found in the rows labeled "GR (interactions only)" in Table 7 and Table 6, where we show utilizing only interaction history led to a 1.3% decrease in Hit Rate@100 for retrieval and a 2.6% NE gap in ranking (recall a 0.1% change in NE is significant, as discussed in Sections 4.1.2 and 4.3.1).

User representations are computed in a target-independent setting. A second issue is most traditional sequential recommenders, including GRU4Rec (Hidasi et al., 2016), SASRec (Kang & McAuley, 2018), BERT4Rec (Sun et al., 2019), S3Rec (Zhou et al., 2020), etc. are formulated in a target-independent fashion where for a target item Φ_i, $\Phi_0, \Phi_1, \ldots, \Phi_{i-1}$ are used as encoder input to compute user representations, which is then used to provide predictions. In contrast, most major DLRM approaches used in industrial settings formulated the sequential modules used in a target-aware fashion, with the ability to incorporate "target" (ranking candidate) information into the user representations. These include DIN (Zhou et al., 2018) (Alibaba), BST (Chen et al., 2019) (Alibaba), TWIN (Chang et al., 2023) (Kwai), and TransAct (Xia et al., 2023) (Pinterest).

Generative Recommenders (GRs) combines the best of both worlds by *interleaving* the content and action sequences (Section 2.2) to enable applying target-aware attention in causal, autoregressive settings. We categorize and contrast prior work and this work in Table 10 [5].

	Input for target item i	**Expected output for target item i**	**Architecture**	**Training Procedure**
GRs	$\Phi_0, a_0, \Phi_1, a_1, \ldots, \Phi_i$	a_i **(target-aware)**	Self-attention (HSTU)	**Causal autoregressive (streaming/single-pass)**
GRU4Rec SASRec	$\Phi_0, \Phi_1, \ldots, \Phi_{i-1}$	Φ_i	RNNs (GRUs) Self-attention (Transformers)	**Causal autoregressive** (multi-pass)
BERT4Rec S3Rec	$\Phi_0, \Phi_1, \ldots, \Phi_{i-1}$ (at inference time)	Φ_i	Self-attention (Transformers)	Sequential multi-pass [6]
DIN BST TWIN TransAct	$\Phi_0, \Phi_1, \ldots, \Phi_i$ $(\Phi_0, a_0), \ldots, (\Phi_{i-1}, a_{i-1}), \Phi_i$	a_i (**target aware**, implicitly as part of DLRMs)	Pairwise attention Self-attention (Transformers) Two-stage pairwise attention Self-attention (Transformers)	Pointwise (**generally streaming/single pass**)

Table 10. Comparison of prior work on sequential recommenders and GRs, in the ranking setting, with DLRMs included for completeness.

Discriminative formulations restrict applicability of prior sequential recommender work to pointwise settings. Finally, traditional sequential recommenders are discriminative by design. Existing sequential recommender literature, including seminal work such as GRU4Rec and SASRec, model $p(\Phi_i|\Phi_0, a_0, \ldots, \Phi_{i-1}, a_{i-1})$, or the conditional distribution of the next item to recommend given users' current states. On the other hand, we observe that there are two probabilistic processes in standard recommendation systems, namely *the process of the recommendation system suggesting a content Φ_i (e.g., some photo or video) to the user*, and *the process of the user reacting to the suggested content Φ_i via some action a_i* (which can be a combination of like, video completion, skip, etc.).

A generative approach needs to model the joint distribution over the sequence of suggested contents and user actions, or $p(\Phi_0, a_0, \Phi_1, a_1, \ldots, \Phi_{n_c-1}, a_{n_c-1})$, as discussed in Section 2.2. Our proposal of *Generative Recommenders* enables modeling of such distributions, as shown in Table 11 (Figure 8). Note that the next action token (a_i) prediction task is exactly the GR ranking setting discussed in Table 1, whereas the next content (Φ_i) prediction task is similar to the retrieval setting adapted to the interleaved setting, with the target changed in order to learn the input data distribution.

Task		**Specification (Inputs / Outputs / Length)**
Next action token (a_i) prediction	x_is	$\Phi_0, a_0, \Phi_1, a_1, \ldots, \Phi_{n_c-2}, a_{n_c-2}, \Phi_{n_c-1}, a_{n_c-1}$
	y_is	$a_0, \varnothing, a_1, \varnothing, \ldots, a_{n_c-2}, \varnothing, a_{n_c-1}, \varnothing$
	n	$2n_c$
Next content token (Φ_i) prediction	x_is	$\Phi_0, a_0, \Phi_1, a_1, \ldots, \Phi_{n_c-2}, a_{n_c-2}, \Phi_{n_c-1}, a_{n_c-1}$
	y_is	$\varnothing, \Phi_1, \varnothing, \Phi_2, \ldots, \varnothing, \Phi_{n_c-1}, \varnothing, \varnothing$
	n	$2n_c$

Table 11. Generative modeling over $p(\Phi_0, a_0, \ldots, \Phi_{n_c-1}, a_{n_c-1})$. An illustration is provided in Figure 8.

Importantly, this formulation not only enables proper modeling of data distribution but further enables sampling sequences of items to recommend to the user directly via e.g., beam search. We hypothesize that this will lead to a superior approach compared with traditional listwise settings (e.g., DPP (Gillenwater et al., 2014) and RL (Zhao et al., 2018)), and we leave the full formulation and evaluation of such systems (briefly discussed in Section 6) as a future work.

C. Evaluation: Synthetic Data

As previously discussed in Section 3.1, standard softmax attention, due to its normalization factor, makes it challenging to capture intensity of user preferences which is important for user representation learning. This aspect is important in

[5] Most large-scale industrial recommenders need to be trained in a streaming/single-pass setting due to vast amount of logged data and the need to maintain up-to-date representations for the high cardinality categorical features used in DLRMs.

[6] BERT4Rec leverages multi-pass training with a mixture of Cloze and pointwise (last item) supervision losses; S3Rec utilizes multi-pass training with pre-training and finetuning as two separate stages.

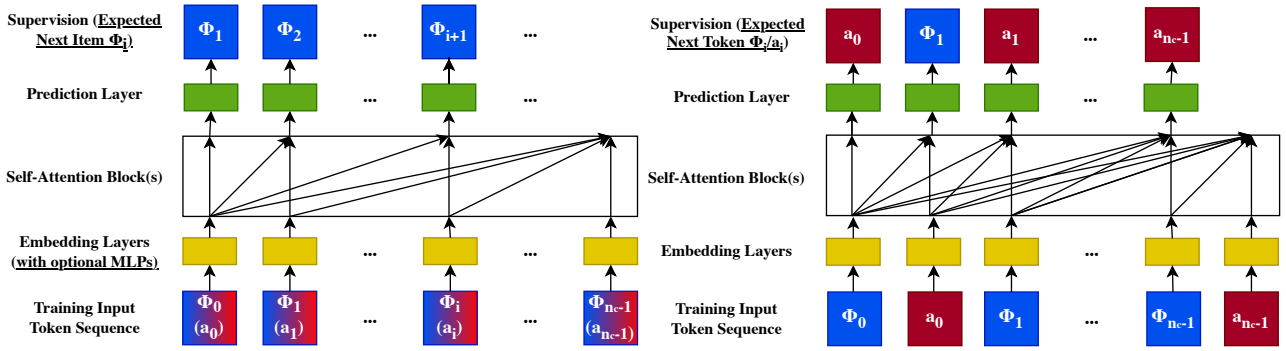


Figure 8. Comparison of traditional sequential recommenders (left) and Generative Recommenders (right). We illustrate sequential recommenders in causal autoregressive settings and GRs without contextual features to facilitate comparison. On the left hand side, the action types a_is are either ignored or combined with item information Φ_is using MLPs, before going into self-attention blocks.

recommendation scenarios as the system may need to predict the intensity of engagements (e.g., number of future positive actions on a particular topic) in addition to the relative ordering of items.

To understand this behavior, we construct synthetic data following a Dirichlet Process that generates streaming data over a dynamic set of vocabulary. Dirichlet Process captures the behavior that 'rich gets richer' in user engagement histories. We set up the synthetic experiment as follows:

- We randomly assign each one of 20,000 item ids to exactly one of 100 categories.
- We generate 1,000,000 records of length 128 each, with the first 90% being used for training and the final 10% used for testing. To simulate the streaming training setting, we make the initial 40% of item ids available initially and the rest available progressively at equal intervals; i.e., at record 500,000, the maximum id that can be sampled is $(40\% + 60\% * 0.5) * 20,000 = 14,000$.
- We randomly select up to 5 categories out of 100 for each record and randomly sample a prior H_c over these 5 categories. We sequentially sample category for each position following a Dirichlet process over possible categories as follows:
  - for $n > 1$:
    - with probability $\alpha/(\alpha + n - 1)$, draw category c from H_c.
    - with probability $n_c/(\alpha + n - 1)$, draw category c, where n_c is the number of previous items with category c.
    - randomly sample an assigned item matching category c subject to streaming constraints.

  where α is uniformly sampled at random from $(1.0, 500.0)$.

The results can be found in Table 2. We always ablate $\text{rab}^{p,t}$ for HSTU as this dataset does not have timestamps. We observe HSTU increasing Hit Rate@10 by more than 100% relative to standard Transformers. Importantly, replacing HSTU's pointwise attention mechanism with softmax ("HSTU w/ Softmax") also leads to a significant reduction in hit rate, verifying the importance of pointwise attention-like aggregation mechanisms.

D. Evaluation: Traditional Sequential Recommender Settings

Our evaluations in Section 4.1.1 focused on comparing HSTU with a state-of-the-art Transformer baseline, SASRec, utilizing latest training recipe. In this section, we further consider two other alternative approaches.

Recurrent neural networks (RNNs). We consider the classical work on sequential recommender, GRU4Rec (Hidasi et al., 2016), to help readers understand how self-attention models, including Transformers and HSTU, compare to traditional RNNs, when all the latest modeling and training improvements are fully incorporated.

Self-supervised sequential approaches. We consider the most popular work, BERT4Rec (Sun et al., 2019), to understand how bidirectional self-supervision (leveraged in BERT4Rec via a Cloze objective) compares with unidirectional causal autoregressive settings, such as SASRec and HSTU.

	Method	HR@10	HR@50	HR@200	NDCG@10	NDCG@200
ML-1M	SASRec (2023)	.2853	.5474	.7528	.1603	.2498
	BERT4Rec	.2843 (-0.4%)	–	–	.1537 (-4.1%)	–
	GRU4Rec	.2811 (-1.5%)	–	–	.1648 (+2.8%)	–
	HSTU	.3097 (+8.6%)	.5754 (+5.1%)	.7716 (+2.5%)	.1720 (+7.3%)	.2606 (+4.3%)
	HSTU-large	**.3294 (+15.5%)**	**.5935 (+8.4%)**	**.7839 (+4.1%)**	**.1893 (+18.1%)**	**.2771 (+10.9%)**
ML-20M	SASRec (2023)	.2906	.5499	.7655	.1621	.2521
	BERT4Rec	.2816 (-3.4%)	–	–	.1703 (+5.1%)	–
	GRU4Rec	.2813 (-3.2%)	–	–	.1730 (+6.7%)	–
	HSTU	.3252 (+11.9%)	.5885 (+7.0%)	.7943 (+3.8%)	.1878 (+15.9%)	.2774 (+10.0%)
	HSTU-large	**.3567 (+22.8%)**	**.6149 (+11.8%)**	**.8076 (+5.5%)**	**.2106 (+30.0%)**	**.2971 (+17.9%)**
Books	SASRec (2023)	.0292	.0729	.1400	.0156	.0350
	HSTU	.0404 (+38.4%)	.0943 (+29.5%)	.1710 (+22.1%)	.0219 (+40.6%)	.0450 (+28.6%)
	HSTU-large	**.0469 (+60.6%)**	**.1066 (+46.2%)**	**.1876 (+33.9%)**	**.0257 (+65.8%)**	**.0508 (+45.1%)**

Table 12. Evaluations of methods on public datasets in traditional sequential recommender settings (multi-pass, full-shuffle). Compared with Table 4, two other baselines (GRU4Rec and BERT4Rec) are included for completeness.

Results are presented in Table 12. We reuse BERT4Rec results and GRU4Rec results on ML-1M and ML-20M as reported by Klenitskiy & Vasilev (2023). Given a sampled softmax loss is used, we hold the number of negatives used constant (128 for ML-1M, ML-20M and 512 for Amazon Books) to ensure a fair comparison between methods.

The results confirm that SASRec remains one of the most competitive approaches in traditional sequential recommendation settings when sampled softmax loss is used (Zhai et al., 2023a; Klenitskiy & Vasilev, 2023), while HSTU significantly outperforms evaluated transformers, RNNs, and self-supervised bidirectional transformers.

E. Evaluation: Traditional DLRM Baselines

The DLRM baseline configurations used in Section 4 reflect continued iterations of hundreds of researchers and engineers over multiple years and a close approximation of production configurations on a large internet platform with billions of daily active users before HSTUs/GRs were deployed. We give a high level description of the models used below.

Ranking Setting. The baseline ranking model, as described in (Mudigere et al., 2022), employs approximately one thousand dense features and fifty sparse features. We incorporated various modeling techniques such as Mixture of Experts (Ma et al., 2018), variants of Deep & Cross Network (Wang et al., 2021), various sequential recommendation modules including target-aware pairwise attention (one commonly used variant in industrial settings can be found in (Zhou et al., 2018)), and residual connection over special interaction layers (He et al., 2015; Zhang et al., 2022). For the low FLOPs regime in the scaling law section (Section 4.3.1), some modules with high computational costs were simplified and/or replaced with other state-of-the-art variants like DCNs to achieve desired FLOPs.

To the best of our knowledge, our DLRM baseline setting represents one of the best known DLRM approaches when recent research are fully incorporated. To validate this claim and to facilitate readers' understanding, we report a typical setup based on identical features but only utilizing major published results including DIN (Zhou et al., 2018), DCN (Wang et al., 2021), and MMoE (Ma et al., 2018) ("DLRM (DIN+DCN)") in Table 7, with the combined architecture illustrated in Figure 9. This setup significantly underperformed our production DLRM setup by 0.71% in NE for the main E-Task and 0.57% in NE for the main C-Task (where 0.1% NE is significant).

Retrieval Setting. The baseline retrieval model employs a standard two-tower neutral retrieval setting (Covington et al., 2016) with mixed in-batch and out-of-batch sampling. The input feature set consists of both high cardinality sparse features (e.g., item ids, user ids) and low cardinality sparse features (e.g. languages, topics, interest entities). A stack of feed forward layers with residual connections (He et al., 2015) is used to compress the input features into user and item embeddings.

Features and Sequence Length. The features used in both of the DLRM baselines, including main user interaction history that is utilized by various sequential encoder/pairwise attention modules, are *strict supersets* of the features used in all GR candidates. This applies to all studies conducted in this paper, including those used in the scaling studies (Section 4.3.1).

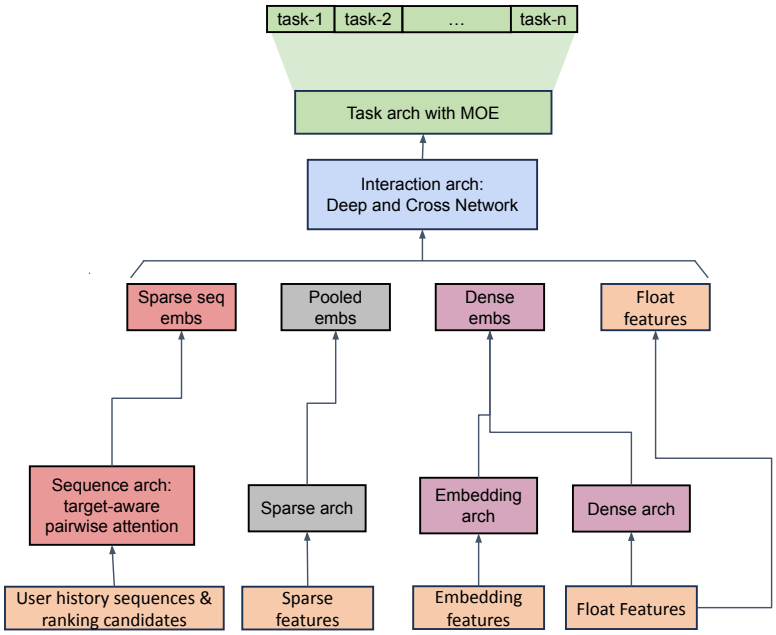


Figure 9. A high level architecture of a baseline DLRM ranking model ("DLRM (DIN+DCN)" in Table 7) that utilizes major published work including DIN (Zhou et al., 2018), DCN (Wang et al., 2021), and MMoE (Ma et al., 2018).

Metric Name	Selection Type		
	Greedy	Weighted	Random
Main Engagement Metric (NE)	0.495	0.494	0.495
Main Consumption Metric (NE)	0.792	0.789	0.791

Table 13. Comparison of subsequence selection methods for *Stochastic Length* on model quality, measured by Normalized Entropy (NE).

F. Stochastic Length

F.1. Subsequence Selection

In Equation (4), we select a subsequence of length L from the full user history in order to increase sparsity. Our empirical results indicate that careful design of the subsequence selection technique can improve model quality. We compute a metric $f_i = t_n - t_i$ which corresponds to the amount of time elapsed since the user interacted with item x_i. We conduct offline experiments with the following subsequence selection methods:

- **Greedy Selection** – Selects L items with smallest values of f_i from S
- **Random Selection** – Selects L items from S randomly
- **Feature-Weighted Selection** – Selects L items from S according to a weighted distribution $1 - f_{n,i}/(\sum_{j=1}^{L} f_{j,i})$

During our offline experiments, the feature-weighted subsequence selection method resulted in the best model quality, as shown in Table 13.

F.2. Impact of Stochastic Length on Sequence Sparsity

In Table 3, we show the impact of Stochastic Length on sequence sparsity for a representative industry-scale configuration with 30-day user engagement history. The sequence sparsity is defined as one minus the ratio of the average sequence length of all samples divided by the maximum sequence length. To better characterize the computational cost of sparse attentions, we also define *s2*, which is defined as one minus the sparsity of the attention matrix. For reference, we present the results for 60-day and 90-day user engagement history in Table 14 and Table 15, respectively.

Alpha	Max Sequence Length							
	1,024		2,048		4,096		8,192	
	sparsity	s2	sparsity	s2	sparsity	s2	sparsity	s2
1.6	71.5%	89.4%	75.8%	92.3%	79.4%	94.7%	83.8%	97.3%
1.7	57.3%	77.6%	60.6%	79.8%	67.3%	86.6%	74.5%	93.3%
1.8	37.5%	56.2%	42.6%	62.1%	51.9%	74.2%	62.6%	85.5%
1.9	15.0%	25.2%	17.7%	29.0%	29.6%	47.5%	57.8%	80.9%
2.0	1.2%	1.7%	2.5%	3.5%	18.9%	30.8%	57.6%	80.6%

Table 14. Impact of *Stochastic Length* (SL) on sequence sparsity, over a 60d user engagement history.

Alpha	Max Sequence Length							
	1,024		2,048		4,096		8,192	
	sparsity	s2	sparsity	s2	sparsity	s2	sparsity	s2
1.6	68.0%	85.0%	74.6%	90.8%	78.6%	93.5%	83.5%	97.3%
1.7	56.3%	76.1%	61.2%	80.6%	67.5%	87.0%	74.3%	93.3%
1.8	38.9%	58.3%	42.0%	61.3%	50.4%	72.4%	61.0%	84.4%
1.9	16.2%	27.3%	17.3%	28.6%	27.2%	44.4%	54.3%	77.8%
2.0	0.9%	1.2%	1.6%	2.1%	13.5%	22.5%	54.0%	77.4%

Table 15. Impact of *Stochastic Length* (SL) on sequence sparsity, over a 90d user engagement history.

F.3. Comparisons Against Sequence Length Extrapolation Techniques

We conduct additional studies to verify that *Stochastic Length* is competitive against existing techniques for sequence length extrapolation used in language modeling. Many existing methods perform sequence length extrapolation through modifications of RoPE (Su et al., 2023). To compare against existing methods, we train an HSTU variant (HSTU-RoPE) with no relative attention bias and rotary embeddings.

We evaluate the following sequence length extrapolation methods on HSTU-RoPE:

- **Zero-Shot** - Apply NTK-Aware RoPE (Peng et al., 2024) before directly evaluating the model with no finetuning;
- **Fine-tune** - Finetune the model for 1000 steps after applying NTK-by-parts (Peng et al., 2024).

We evaluate the following sequence length extrapolation methods on HSTU (includes relative attention bias, no rotary embeddings):

- **Zero-Shot** - Clamp the relative position bias according to the maximum training sequence length, directly evaluate the model (Raffel et al., 2020; Press et al., 2022);
- **Fine-tune** - Clamp the relative position bias according to the maximum training sequence length, fine-tune the model for 1000 steps before evaluating the model.

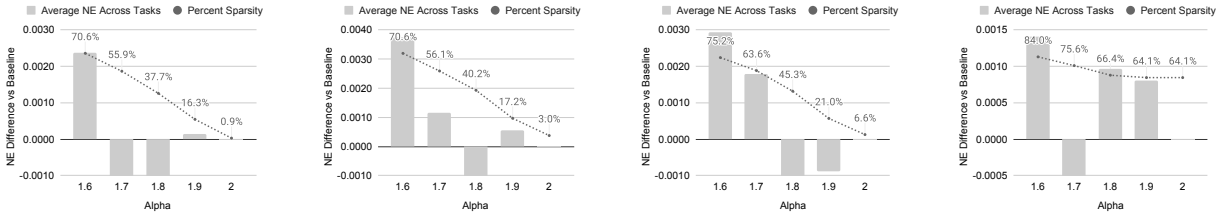


Figure 10. Impact of *Stochastic Length* (SL) on ranking model metrics. Left to right: $n = [1024, 2048, 4096, 8192]$ (n is after interleaving algorithm as discussed in Section 2.2 to enable target-aware cross attention in causal-masked settings).

Evaluation Strategy	Average NE Difference vs Full Sequence Baseline		
	Model Type	2048 / 52% Sparsity	4096 / 75% Sparsity
Zero-shot	HSTU (Raffel et al., 2020)	6.46%	10.35%
	HSTU-RoPE (Peng et al., 2024)	7.51%	11.27%
Fine-tune	HSTU (Raffel et al., 2020)	1.92%	2.21%
	HSTU-RoPE (Peng et al., 2024)	1.61%	2.19%
Stochastic Length (SL)	HSTU	**0.098**%	**0.64**%

Table 16. Comparisons of *Stochastic Length* (SL) vs existing Length Extrapolation methods.

In Table 16, we report the NE difference between models with induced data sparsity during training (Stochastic Length, zero-shot, fine-tuning) and models trained on the full data. We define the sparsity for zero-shot and fine-tuning techniques to be the average sequence length during training divided by the max sequence length during evaluation. All zero-shot and fine-tuned models are trained on 1024 sequence length data and are evaluated against 2048 and 4096 sequence length data. In order to find an appropriate *Stochastic Length* baseline for these techniques, we select *Stochastic Length* settings which result in the same data sparsity metrics.

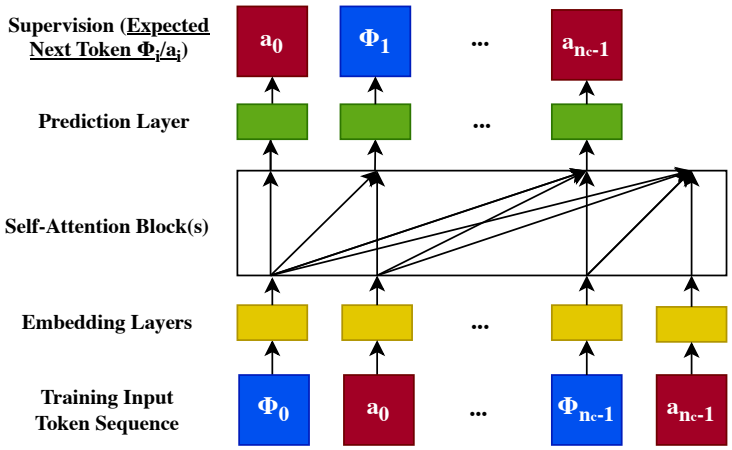
We believe that zero-shot and fine-tuning approaches to sequence length extrapolation are not well-suited for recommendation scenarios that deal with high cardinality ids. Empirically, we observe that *Stochastic Length* significantly outperforms fine-tuning and zero-shot approaches. We believe that this could be due to our large vocabulary size. Zero-shot and fine-tuning approaches fail to learn good representations for older ids, which could hurt their ability to fully leverage the information contained in longer sequences.

G. Sparse Grouped GEMMs and Fused Relative Attention Bias

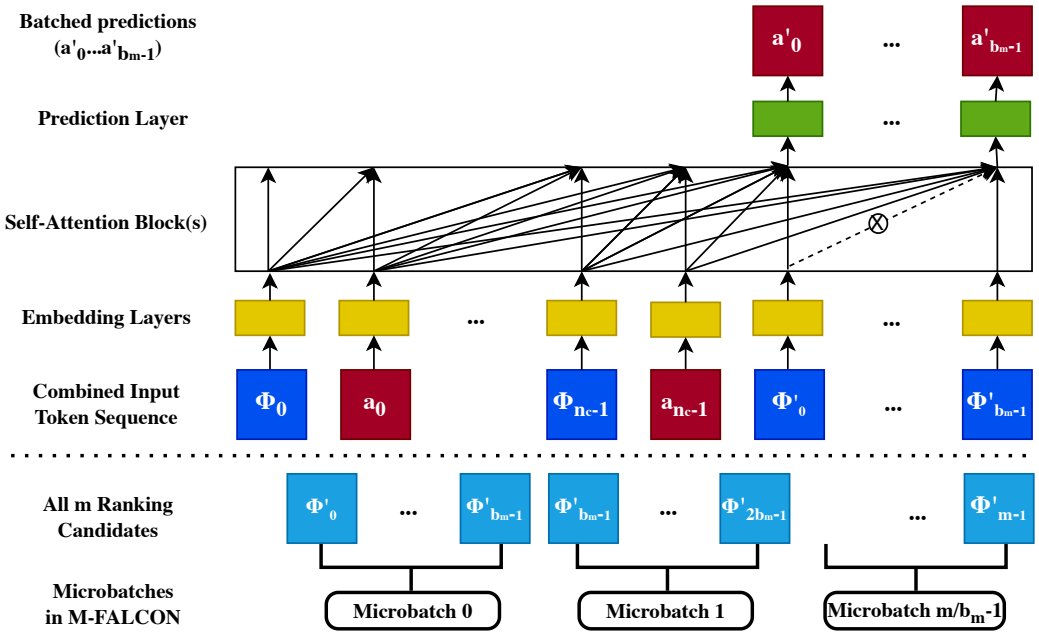
We provide additional information about the efficient HSTU attention kernel that was introduced in Section 3.2. Our approach builds upon Memory-efficient Attention (Rabe & Staats, 2021) and FlashAttention (Dao et al., 2022), and is a memory-efficient self-attention mechanism that divides the input into blocks and avoids materializing the large $h \times N \times N$ intermediate attention tensors for the backward pass. By exploiting the sparsity of input sequences, we can reformulate the attention computation as a group of back-to-back GEMMs with different shapes. We implement efficient GPU kernels to accelerate this computation. The construction of the relative attention bias is also a bottleneck due to memory accesses. To address this issue, we have fused the relative bias construction and the grouped GEMMs into a single GPU kernel and managed to accumulate gradients using GPU's fast shared memory in the backward pass. Although our algorithm requires recomputing attention and relative bias in the backward pass, it is significantly faster and uses less memory than the standard approach used in Transformers.

H. Microbatched-Fast Attention Leveraging Cacheable OperatioNs (M-FALCON)

In this section, we provide a detailed description of the M-FALCON algorithm discussed in Section 3.4. We give pseudocode for M-FALCON in Algorithm 1. M-FALCON introduces three key ideas.



(a) GR's ranking model training (with $n = 2n_c$ tokens), in causal autoregressive settings.



(b) GR's ranking model inference utilizing the M-FALCON algorithm.

Figure 11. Illustration of the M-FALCON algorithm. Top: model training in GR's target-aware formulation. Bottom: model inference with m candidates $\Phi'_0, \ldots, \Phi'_{m-1}$, divided into $\lceil m/b_m \rceil$ microbatches, where we show model inference for the first microbatch $\Phi'_0, \ldots, \Phi'_{b_m-1}$ (with $2n_c + b_m$ total tokens after $\Phi_0, a_0, \ldots, \Phi_{n_c-1}, a_{n_c-1}$ are taken into account) above the dotted line. Note that the self-attention algorithm is modified such that Φ'_i cannot attend to Φ'_j when $i \neq j$ – this is highlighted with "×" in the figure.

Batched inference can be applied to target-aware causal autoregressive settings. The ranking task in GR is formulated in a target-aware fashion as discussed in Section 2.2. Common wisdom suggests that in a target-aware setting, we need to perform inference for one item at a time, with a cost of $O(mn^2d)$ for m candidates and a sequence length of n. Here we show that this is not the optimal solution; even with vanilla Transformers, we can modify the attention mask used in self-attention to batch such operations ("batched inference") and reduce cost to $O((n+m)^2d) = O(n^2d)$.

An illustration is provided in Figure 11. Here, both Figure 11 (a) and (b) involve an attention mask matrix for causal autoregressive settings. The key difference is that Figure 11 (a) uses a standard lower triangular matrix of size $2n_c$ for causal

training, whereas Figure 11 (b) modifies a lower triangular matrix of size $2n_c + b_m$ by setting entries for (i, j)s where $i, j \geq 2n_c, i \neq j$ to False or $-\infty$ to prevent target positions $\Phi'_0, \ldots, \Phi'_{b_m-1}$ from attending to each other. It is easy to see that by doing so, the output of the self-attention block for Φ'_i, a'_i, only depends on $\Phi_0, a_0, \ldots, \Phi_{n_c-1}, a_{n_c-1}$, but not on Φ'_j $(i \neq j)$. In other words, by making a forward pass over $(2n_c + b_m)$ tokens using the modified attention mask, we can now obtain the same results for the last b_m tokens as if we've made b_m separate forward passes over $(2n_c + 1)$ tokens, with Φ'_i placed at the $2n_c$-th (0-based) position during the i-th forward pass utilizing a standard causal attention mask.

Microbatching scales batched inference to large candidate sets. Ranking stage may need to deal with a large number of ranking candidates, up to tens of thousands (Wang et al., 2020). We can divide the overall m candidates into $\lceil m/b_m \rceil$ microbatches of size b_m such that $O(b_m) = O(n)$, which retains the $O((n + m)^2 d) = O(n^2 d)$ running time previously discussed for most practical recommender settings, up to tens of thousands of candidates.

Encoder-level caching enables compute sharing within and *across* requests. Finally, KV caching (Pope et al., 2022) can be applied both within and across requests. For instance, for the HSTU model presented in this work (Section 3), $K(X)$ and $V(X)$ are fully cachable across microbatches within and/or across requests. For a cached forward pass, we only need to compute $U(X)$, $Q(X)$, $K(X)$, and $V(X)$ for the last b_m tokens, while $K(X)$ and $V(X)$ for the sequentialized user history containing n tokens can be cached and reused. $f_2(\text{Norm}(A(X)V(X)) \odot U(X))$ similarly only needs to be recomputed for the b_m candidates. This reduces the cached forward pass's computational complexity to $O(b_m d^2 + b_m n d)$, which significantly improves upon $O((n + b_m)d^2 + (n + b_m)^2 d)$ by a factor of 2-4 even when $b_m = n$.

Algorithm 1 M-FALCON Algorithm.

Input: Merged token series $x_0, x_1, \ldots, x_{n-1}$ (can be e.g., $(\Phi_0, a_0, \ldots, \Phi_{n_c-1}, a_{n_c-1})$ where $n = 2n_c$); m ranking candidates $\Phi'_0, \ldots, \Phi'_{m-1}$; a b-layer h-heads self-attention model trained in causal autoregressive settings (e.g., HSTU or Transformers) $f(X, cacheStates, attnMask) \to (X', updatedCacheStates)$ where $X, X' \in \mathbb{R}^{N \times d}$, $attnMask \in \mathbb{R}^{N \times N}$, and $cachedStates, updatedCacheStates \in \mathbb{R}^{b \times h \times N \times d_{qk}} \times \mathbb{R}^{b \times h \times N \times d_{qk}}$ (due to caching $K(X)$s and $V(X)$s across b layers); microbatch size b_m, where we assume m is a multiple of b_m for simplicity.
Output: Predictions for all m ranking candidates, $(a'_0, \ldots, a'_{m-1})$.
$numMicrobatches = (m + b_m - 1)//b_m$
$attnMask = L_{n+b_m}$ {L_{n+b_m} represents a lower triangular matrix. Lower triangular entries are 0s, the rest are $-\infty$.}
$attnMask[i, j] = -\infty$ for $i, j \geq n, i \neq j$ {This prevents the last b_m entries from attending to each other.}
$(a'_0, a'_1, \ldots, a'_{b_m-1}), kvCache \leftarrow f(embLayer((x_0, x_1, \ldots, x_{n-1}, \Phi'_0, \ldots, \Phi'_{b_m-1})), \varnothing, attnMask)$
$predictions = (a'_0, a'_1, \ldots, a'_{b_m-1})$
$i = 1$
while $i < numMicrobatches$ **do**
  $(a'_{b_m i}, a'_{b_m i+1}, a'_{b_m(i+1)-1}), \_ \leftarrow f(embLayer((x_0, x_1, \ldots, x_{n-1}, \Phi'_{b_m i}, \ldots, \Phi'_{b_m(i+1)-1})), kvCache, attnMask)$
  $predictions \leftarrow predictions + (a'_{b_m i}, a'_{b_m i+1}, \ldots, a'_{b_m(i+1)-1})$
  $i \leftarrow i + 1$
end while
return $predictions$

Algorithm 1 is illustrated in Figure 11 to help with understanding. We remark that M-FALCON is not only applicable to HSTUs and GRs, but also broadly applicable as an inference optimization algorithm for other target-aware causal autoregressive settings based on self-attention architectures.

H.1. Evaluation of Inference Throughput: Generative Recommenders (GRs) w/ M-FALCON vs DLRMs

As discussed in Section 3.4, M-FALCON handles b_m candidates in parallel to amortize computation costs across all m candidates at inference time. To understand our design, we compare the throughput (i.e., the number of candidates scored per second, QPS) of GRs and DLRMs based on the same hardware setups.

As shown in Figure 12 and Figure 13, GRs' throughput scales in a sublinear way based on the number of ranking-stage candidates (m), up to a certain region – $m = 2048$ in our case study – due to *batched inference* enabling cost amortization. This confirms the importance of batched inference in causal autoregressive settings. Due to attention complexity scaling as $O((n + b_m)^2)$, leveraging multiple microbatches by itself improves throughput. Caching further eliminates redundant linear and attention computations on top of microbatching. The two combined resulted in up to 1.99x additional speedups relative

to the $b_m = m = 1024$ baseline using a single microbatch, as shown in Figure 13. Overall, with the efficient HSTU encoder design and utilizing M-FALCON, HSTU-based Generative Recommenders *outperform* DLRMs in terms of throughput on a large-scale production setup by up to 2.99x, despite GRs being *285x more complex in terms of FLOPs*.

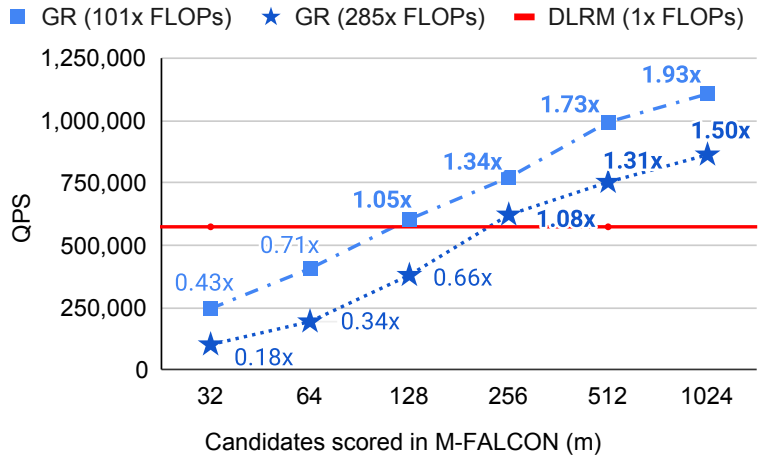


Figure 12. End-to-end inference throughtput: DLRMs vs GRs (w/ M-FALCON) in large-scale industrial settings. Note that this figure is the same as Figure 6, and is reproduced here to facilitate reading.

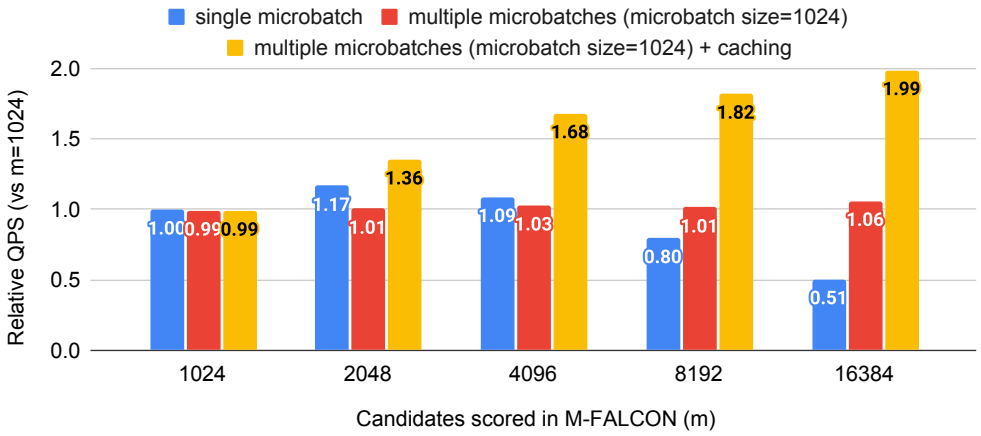


Figure 13. End-to-end inference throughtput: M-FALCON throughput scaling, on top of the 285x FLOPs GR model, in large batch settings where m (total number of ranking candidates) ranges from 1024 to 16384, and $b_m = 1024$.