Sequential Consumption-Aware Ranking Model for Recommendations at Twitch

Jian Vora **Edgar Chen** **Nikita Mishra** **Saad Ali**
{jianfvor, chenedga, mishniki, saali}@twitch.tv

Abstract

Sequential models to encode past user behavior have shown great progress in web-scale personalized recommendation systems. While most of these models are trained for next-item prediction for end-to-end recommendations, we explore the use of these sequential models in the ranking stage of a multi-stage recommendation system. Sequential models capture short-term user preferences which complement batched features learned over a longer time period. In particular, we explore various design choices in encoding user interaction history as a plug-and-play module in any deep-learning ranking system and introduce a novel way of incorporating time watched in the sequential model in addition to the user interaction sequence. We experiment with Twitch (a community driven live streaming video platform) user-interaction data, present various ablation results, and show a 9% improvement in offline metrics.

1 Introduction

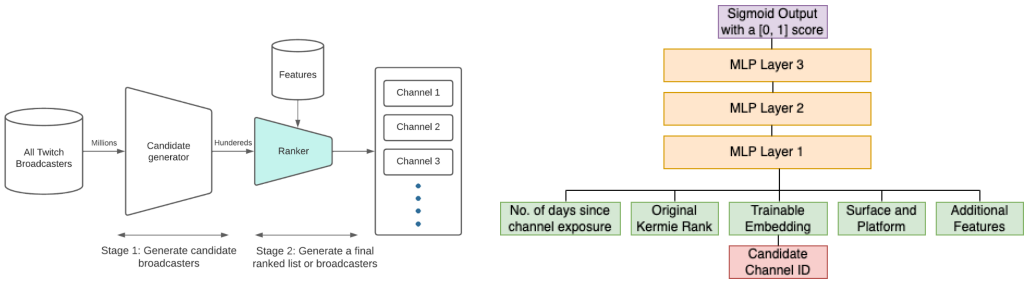
Industrial scale recommendation-systems are trained on large amounts of user interaction data and capture user behaviour mainly through implicit feedback to provide users a personalized experience. These systems are crucial for businesses as they drive growth by boosting engagement. Until recently, the most successful industry-scale recommendation models involve some combination of continuous features and embeddings in a MLP-like structure [14, 10]. With the advent of the transformer architecture, we have seen significant improvement in the modeling sequences [12]. Many improvements in natural language processing (NLP) ranging from sentiment classification, entity extraction, summarization, etc., have come from scaling these transformer architectures to train on large amounts of data (large language models) [6, 2, 1].

Progress in sequential modeling in NLP has led to improvements in recommendation systems as well, even in production-scale systems, as demonstrated by the successes of SASRec [7] and BERT4Rec [11]. These approaches consider recommendations as a sequence modeling problem. That is, given a sequence of past user interactions (e.g. clicks, page visits, video watches), what is the next most likely item that a user will engage with. One particular design that has recently been used is to employ sequential model along with an MLP-like model. In this design, sequential model can be responsive to dynamic user behavior, allowing them to capture short-term behavior changes, while other parts of the model leverage long-term features to understand a user's long-term static preferences. Generally transformers are preferred over other models such as RNNs, CNNs, MLPs for sequence modelling for recommendations, as they leverage the self-attention mechanism to capture the *dependency* among items in users' interaction sequences.

1.1 Twitch Recommendation System

Twitch uses a two-staged recommendation system, similar to many other industrial scale recommendation systems such as YouTube [5, 4], Alibaba [3, 13], and Pinterest [17, 16]. The first stage is a **retrieval** stage that filters relevant items for a large $(300,000)$ catalog of items down to $10-100$ items, followed by a **ranking** stage that takes in the candidates generated by the retrieval stage and ranks them, i.e., decides the order in which they are shown to the user. Figure 1a illustrates a visual description of the two-stage system used by Twitch.

Preprint. Under review.



(a) Two-staged recommendation system

(b) Deep ranker architecture in production

Figure 1: Conceptual flow of items in Twitch Recommender System

The candidate generator filters down relevant items (also interchangeably called channels, streamers, or broadcasters) from a pool of approximately 300K channels, and generates a list of hundreds of channels for the ranking stage. The candidate generator is optimized for a high recall that learns an embedding for each user and channel. Channel candidates for a given user are generated by choosing the channels that maximize the inner product with the user embedding.

In the ranking stage, given C candidates from the candidate generation stage, the task is to 'rank' the channels. We do this by scoring each candidate (which indicates the probability the user will watch the channel for more than one minute) and then sorting the channels based on the scores assigned to them. We find in practice that our candidate generator is near the optimal point of tradeoff between training times (8+ hours) and performance, with older models showing performance degradations due to the fast-moving nature of the Twitch ecosystem. In contrast, the ranking stage is more scalable than the candidate generator model because it operates over fewer channels, allowing it to use more detailed features and computationally expensive feature processing stages if needed, given the same training and inference time budgets.

In this work, we show that leveraging sequential user interaction information leads to significant boost (9% improvement in offline metrics) in the performance for Twitch's ranking model. The current ranker model in production uses MLP layers to score each candidate channel based on a set of features as shown in Figure 1b, however, it is agnostic to the sequence of past user interactions. The sequence of user interactions provide a useful signal to capture the changing user preferences which help one guide the order in which channels are shown to the user. Additionally, in NLP tasks, all tokens are treated to be equally important, but video watch sequences have extra signals of interaction quality (minutes spent on the stream, chat, follow actions), which can leveraged to further improve performance.

Mentioned below are the contributions of this work:

- Explore and validate various design choices for integrating sequential data in the existing Twitch production ranker, and how they affect downstream performance for Twitch data.
- Incorporate a novel way of using channel watch duration in the user history, which *implicitly* guides the model to attend to more watched channels in the history.
- A study of the tradeoff between increased computational costs (pre-processing, training, and inference) and better performance as a result of using sequential data.

2 Related Work

2.1 Sequential Data for End-to-End Recommendations

Sequential models such as RNNs, transformers trained on user histories are capable of replacing end-to-end recommendation systems (their vocabulary is all the items in the catalog). At inference time, these models predict the next item given some sequence of past interactions, i.e., all these models are causal at inference time. During training time, we can classify models into the following:

1. **Unidirectional Models:** Unidirectional Models such as SASRec [7] or its extensions [15] are trained to predict the next item given a sequence of *past* interactions. Training data for such a model is generated by having the actual history as the input data and the same history shifted by one step in the future as the target. Advantages of training such a model is that it

aligns best with the recommendation problem but on the other hand it might not have an overall understanding of what interaction trajectories might look like.

2. **Bidirectional Models:** Bidirectional Models such as BERT4Rec [11] on the other hand are trained to predict masked items in a user interaction sequence by looking at *both* the past and the future interactions. They are trained using the masked language modeling (MLM) loss [6] and have typically shown better performance than unidirectional models [11]. At inference time, an additional mask token is appended to the history which is predicted and that serves the next item prediction task.
3. **Pre-trained Language Models:** The above two class of models were trained entirely on user interaction data either using the MLM loss or the next item predicition loss. Large Language Models (LLMs) are trained using the MLM loss on corpora of natural language but encode very useful priors and have show out-of-the-box improved performance using either few-shot prompting [9] or finetuning [8]. However, the amount of priors that can be extracted about Twitch channels, especially on smaller and newer streamers, is limited.

2.2 Sequential Data for Ranking

The setup of ranking problem in recommendations is as follows: given a user u, we need to score each channel id c_i in the set of candidate channel ids C. This score which indicates the affinity of the user u to candidate c_i. Earlier ranking systems used various user and channel features to train a machine learning model (XGBoost, MLP layers) to predict the score. Recent works such as Alibaba's Behavior Sequence Transformer (BST) [3] and Pinterest's TransACT [16] have shown that using user interaction history as a feature leads to a significant boost in the ranker's performance even at industrial scale. They use transformers to get sequence representations which are concatenated with other long-term features based on which the final model is trained end-to-end.

Both [3] and [16] are closest to our work, but in the subsequent sections, we highlight some key differences from both the works which are important for Twitch specific ranking use-case, both in the pre-processing and modelling aspect. Another key difference is our integration of sequential minutes-watched data in addition to pageview histories which motivates the use of having more meta-data to help for sequential modelling in recommendations in general.

3 Sequential Deep Ranker

In this section, we describe the data features used in the model, pre-processing steps, the overall sequential deep ranker architecture, and the final loss function over which the model is trained.

3.1 Data Features & Pre-Processing

For each user u and candidate c, we are given S static batch features $\{b_i\}_{i=1}^{S}$ which are a mix of numerical and categorical variables, as shown on the left side of Figure 2. Some examples of these features in the Twitch context include number of days since channel exposure to user and days since channel visit, user-category minutes watched in the past two hours, and surface of the user. An extensive list of features that are used is shared in Appendix A. Apart from the static features, we also have user interaction histories. For each user u and a candidate channel id c, we extract the following two sequences from the user's watch history:

1. $\mathcal{P}_{u,c} = (p_u^{(1)}, p_u^{(2)}, \ldots, p_u^{(N-1)}, c)$ which denotes the sequence of channel pageviews (i.e. channel ids of channels owning that page) of the user u prior to the time at which the candidate channel c was presented to the user.
2. $\mathcal{T}_{u,c} = (t_u^{(1)}, t_u^{(2)}, \ldots t_u^{(N-1)}, t_c)$ which denotes the minutes watched by the user on each of the pageviews/channels corresponding to the same position in $\mathcal{P}_{u,c}$. In other words, there is a one-on-one correspondence between the elements of $\mathcal{P}_{u,c}$ and $\mathcal{T}_{u,c}$.

We choose the pageview event as the token of the sequence as it captures the user's recent history and interests, and gives sequences of desirable length that a transformer's token limit can handle well, around the length of a paragraph or two (100-200 tokens) for heavier users. Appending minute-watched to the pageview events adds further information about the user's engagement with the channel of each pageview.

We use a transformer based architecture (described in Section 3.2) which requires sequences to have fixed lengths. We fix the maximum length of interaction sequences to be $N = 100$ which is a hyperparameter. Shorter sequences are padded with a special `[PAD]` token while longer sequences

are truncated and only the N most recent interactions are taken. Twitch pageview histories have many pageviews which are repeated consecutively multiple times in the history. This can due to the user watching the stream on different platforms, tabs, or even refreshing the page as that counts as a new pageview. To remove this bias from the dataset (as it does not capture the user preference), we removed all consecutive pageviews of the channel id if they occured within 60 minutes of one another. The pageview with the maximum minutes watched was kept in the dataset as a representative.

For model training, the target variable is a binary variable which is 1 *iff* the user has watched the candidate channel for more than a minute, else it is 0. The task for the model is to predict the probability of a minute-watch for a new candidate channel given both the batch and sequential features. Thus, the model takes in all the features and predicts a score in $[0, 1]$ (ensured by a sigmoid activation at the final logit) for each candidate channel. The model was trained end-to-end using the binary cross entropy (BCE) loss on the target variable.

There also exists a `[NA]` token for the minutes watched sequence where data is not available for that pageview in the rare cases where there is a data quality issue. We append the candidate channel into the pageview sequence $\mathcal{P}_{u,c}$ as done in [16] and append the `[NA]` token to the minutes watched sequence $\mathcal{T}_{u,c}$. We found this design choice to perform better than appending the candidate embeddings to all items in the history as done in [3].

3.2 Model Architecture

We now go over some key features of the sequential deep ranker architecture, described in Figure 2.

Embedding Layer: There are two separate embedding tables for the pageviews and minutes watched sequences. Using these tables, each channel id is mapped to a 50 dimensional embedding and each minute watched (as a string) is mapped to a 10 dimensional embedding. Both of these embeddings are concatenated and passed into the transformer block. We have also experimented with passing the candidate channel embedding into the MLP layer as well (which the current deep ranker production model takes in) and discuss the results in Section 4.

Transformer Layers: The embeddings are passed into 2 transformer layers with 4 attention heads with an hidden embedding dimension of 64 and feed-forward dimension of 32. We found that not adding positional embeddings benefits the performance as also shown in [16]. We hypothesize that this may due to a changing meaning of a difference in position depending on how much time has elapsed between two interactions— when we compare interactions in positions 2 and 3 they can be anywhere from one minute to one week apart. Adding positional embeddings based on the age of the interaction is a potential future work.

MLP Layers: We used 3 MLP layers similar to the current production deep ranker model. MLP layers take in both the batch and sequential features, where hidden layer has a dimension of 150 and the last layer predicted a single scalar which was fed into a sigmoid activation.

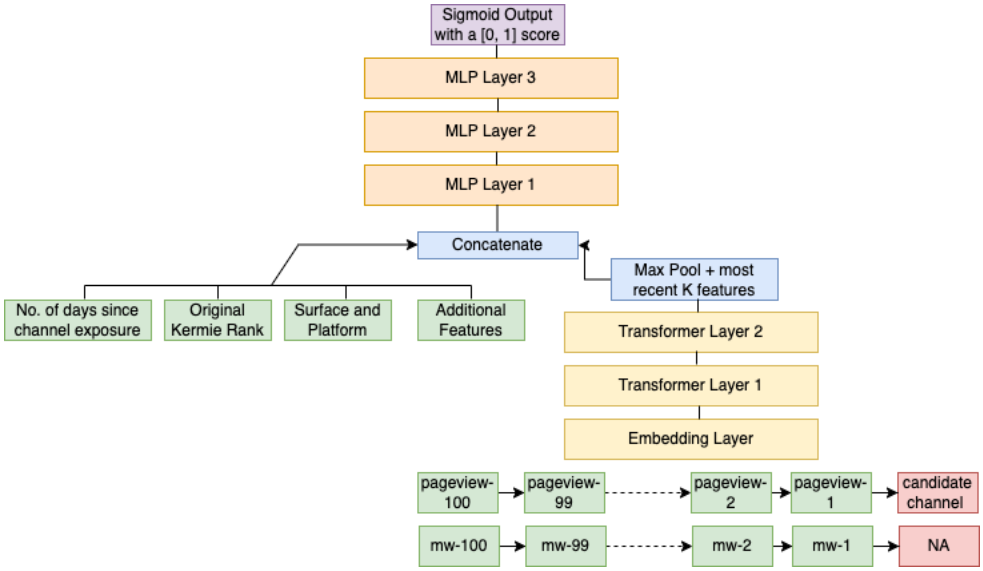


Figure 2: Sequential Deep Ranker Architecture. Pageviews are the channel IDs while mw stands for minutes watched of that channel ID in that interaction.

Fusing Batch and Sequential Features: The output of the transformer block is of the shape $B \times L \times D$, where B is the batch-size, L is the maximum sequence length, and D is the transform hidden-embedding dimensionality. We need to merge this with other ranker features of the shape $B \times S$. For this purpose, we take the max-pool of the transformer features on the time dimension. We also take the $K = 10$ most recent item feature outputs from the transformer and flatten them to concatenate with the batch features. This ensures that we pass both the summarization of the history (max-pool) and features of the K most recent interactions as well. Thus, a tensor of shape $B \times (S + (K + 1)D)$ is the input to the MLP layers.

4 Results

We trained the above ranking model offline on Twitch user interaction data over a week. For evaluation, we used the data for the next day and all data of the past week was used as the sequence data to generate recommendations. We don't have any online results yet, but we plan to bring the model to an A/B test in Q4 2023. In the final dataset, we had 765k unique users, 166k unique channels, and a total of 11.8M training datapoints. Around 2.7M of these datapoints had a trivial sequence length of 1, i.e., only the candidate channel id in the history. The maximum sequence length was 100. For this, the mean and median sequence lengths was 38.89 and 25 respectively before padding the sequences. This indicates that the interaction sequence lengths are heavy-tailed, i.e., most users do not have a lot of interaction with the platform but some have a high amount of interaction.

METRIC NAME	DR	SDR WITHOUT TIME	SDR WITH TIME
PRECISION@1	0.2678	+8.5%	**+9.2%**
MAP@1	0.2641	+8.4%	**+8.9%**
NDCG@6	0.4253	+3.6%	**+4.0%**
MAP@6	0.3775	+4.8%	**+5.1%**

Table 1: Comparison of ranking performance with and without the addition of minutes watched data.

For evaluation, we use a variety of metrics such as precision@k, ndcg@k, and map@k metrics which are typically used to evaluate ranking systems (higher is better). All the results were based on offline experiments. We shall refer to the deep ranker as DR and sequential deep ranker as SDR henceforth.

Relevance of Minutes Watched in the Sequential Model: Table 1 illustrates that using sequential data improves the performance significantly and that addition of minutes watched data in the model further improves the performance by $\sim 0.6\%$. This is a good takeway for many other recommendation problems which use sequential data as additional metadata differentiates the task from NLP tasks.

Varying Maximum Sequence Length: We also experimented with varying maximum sequence lengths of $25, 50$ and 100. We see that the performance increases with adding more history in the model as demonstrated in Table 2. We see that the gains start to flatten out and increasing the sequence lengths gives us gains only up to some extent. On the other hand, we also compared how the computational costs change which gives one a tradeoff between performance and complexity (these are computed relative to the costs of the deep ranker model). We see that costs increase the most in the pre-processing and training stages, and not in the inference stage which just involves a forward pass through the transformer layers. This is also because the GPU is underutilized in the DR, which means that the runtime is only increased mildly by the SDR in the GPU stages as the bottlenecks are mostly data-related.

METRIC NAME	SDR(MAX-LEN=25)	SDR(MAX-LEN=50)	SDR(MAX-LEN=100)
PRECISION@1	0.2793	+1.6%	**+1.9%**
MAP@1	0.2763	+1.5%	**+1.8%**
NDCG@6	0.4287	+0.7%	**+0.9%**
MAP@6	0.3713	+0.9%	**+1.1%**
PREPROCESSING	+60.3%	+61.2%	+62.8%
TRAINING	+21.4%	+24.6%	+28.3%
INFERENCE	+2.3%	+3.1%	+4.8%

Table 2: Comparision of the ranking performance with varying sequence lengths in SDR along with the tradeoff in computational costs (preprocessing, training, and inference).

Passing Candidate Channel Embeddings in the MLP: The original deep ranker used to take in channel embeddings as a feature in the MLP layers. We experiment if this is needed or if the transformer feature for the candidate channel id is sufficient. We find that in fact explicitly adding the channel embedding in the MLP layers slightly deteriorates performance by up to 0.5%. This is interesting in itself that embeddings trained by using them in different parts of the network might point to some optimization problems which leads to inferior performance.

5 Conclusion and Future Work

Overall, we explored the use of sequential user interactions for rankings at Twitch which shows very promising performance in offline metrics. By using the powerful capability of capturing sequential relations, we show the power of the Transformer in modeling user behavior sequences for recommendation by extensive experiments. The learnings from our work can also be used for ranking many other entities such as videos, products, reviews on different platforms of Amazon. The immediate next steps would be to try an A/B test to see if online minutes watched increase. On the algorithmic side, some avenues of future work would be to incorporate an MLM loss for training the transformer along with the minute-watch target. Exploring the use of candidate generator embeddings (the retrieval stage) instead of training them from scratch could also help in improving performance and significantly lowering the training cost as most of the trainable parameters come from the embeddings. Using additional sequential data such as search queries, follow sequence, chat, and spend (which are typically sparse, but provide useful information) can be potential future work.

6 Customer Problem Statement

This work helps our recommender systems understand customers' short term interests, driving increased hours watched and engagement on the site. It uses GDPR-compliant data and should not disadvantage any users or streamers, although users with longer recent watch histories will benefit more. There are fallbacks in place such that if the SDR (or current DR) fails to run, users will see slightly degraded recommendations.

References

[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[2] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

[3] Qiwei Chen, Huan Zhao, Wei Li, Pipei Huang, and Wenwu Ou. 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*, pages 1–4, 2019.

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

[5] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, NY, USA, 2016.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

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

[8] Jinming Li, Wentao Zhang, Tian Wang, Guanglei Xiong, Alan Lu, and Gerard Medioni. Gpt4rec: A generative framework for personalized recommendation and user interests interpretation. *arXiv preprint arXiv:2304.03879*, 2023.

[9] Junling Liu, Chao Liu, Renjie Lv, Kang Zhou, and Yan Zhang. Is chatgpt a good recommender? a preliminary study. *arXiv preprint arXiv:2304.10149*, 2023.

[10] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091*, 2019.

[11] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1441–1450, 2019.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[13] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 839–848, 2018.

[14] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*, pages 1–7. 2017.

[15] Liwei Wu, Shuqing Li, Cho-Jui Hsieh, and James Sharpnack. Temporal collaborative ranking via personalized transformer. *arXiv preprint arXiv:1908.05435*, 2019.

[16] Xue Xia, Pong Eksombatchai, Nikil Pancha, Dhruvil Deven Badani, Po-Wei Wang, Neng Gu, Saurabh Vishwas Joshi, Nazanin Farahpour, Zhiyuan Zhang, and Andrew Zhai. Transact: Transformer-based realtime user action model for recommendation at pinterest. *arXiv preprint arXiv:2306.00248*, 2023.

[17] Jiajing Xu, Andrew Zhai, and Charles Rosenberg. Rethinking personalized ranking at pinterest: An end-to-end approach. In *Proceedings of the 16th ACM Conference on Recommender Systems*, pages 502–505, 2022.

A List of Features Used in the Deep Ranker

The following features are currently used in the deep ranker model running in production:

- User ID
- Candidate Channel ID
- Surface
- Platform (Web, iOS, Android, TV)
- Minutes watched in the same category in the last 2 hours
- Original Candidate Generator Rank
- No. of days since channel exposure
- No. of days since click on channel
- No. of exposures of channel in the last 30 days
- No. of clicks on channel in the last 30 days