Recommendation on Live-Streaming Platforms: Dynamic Availability and Repeat Consumption

Jérémie Rappaz
EPFL
Lausanne, Switzerland
jeremie.rappaz@epfl.ch

Julian McAuley
University of California, San Diego
CA, USA
jmcauley@eng.ucsd.edu

Karl Aberer
EPFL
Lausanne, Switzerland
karl.aberer@epfl.ch

ABSTRACT

Live-streaming platforms broadcast user-generated video in real-time. Recommendation on these platforms shares similarities with traditional settings, such as a large volume of heterogeneous content and highly skewed interaction distributions. However, several challenges must be overcome to adapt recommendation algorithms to live-streaming platforms: first, content *availability* is dynamic which restricts users to choose from only a subset of items at any given time; during training and inference we must carefully handle this factor in order to properly account for such signals, where 'non-interactions' reflect availability as much as implicit preference. Streamers are also fundamentally different from 'items' in traditional settings: repeat consumption of specific channels plays a significant role, though the content itself is fundamentally ephemeral.

In this work, we study recommendation in this setting of a dynamically evolving set of available items. We propose *LiveRec*, a self-attentive model that personalizes item ranking based on both historical interactions and current availability. We also show that carefully modelling repeat consumption plays a significant role in model performance. To validate our approach, and to inspire further research on this setting, we release a dataset containing 475M user interactions on *Twitch* over a 43-day period. We evaluate our approach on a recommendation task and show our method to outperform various strong baselines in ranking the currently available content.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; *Social networks*; *Multimedia streaming*.

KEYWORDS

recommender systems, ranking methods, datasets, live-streaming, repeat consumption

ACM Reference Format:
Jérémie Rappaz, Julian McAuley, and Karl Aberer. 2021. Recommendation on Live-Streaming Platforms: Dynamic Availability and Repeat Consumption. In *Fifteenth ACM Conference on Recommender Systems (RecSys '21), September 27-October 1, 2021, Amsterdam, Netherlands.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3460231.3474267

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
RecSys '21, September 27-October 1, 2021, Amsterdam, Netherlands
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8458-2/21/09…$15.00
https://doi.org/10.1145/3460231.3474267

1 INTRODUCTION

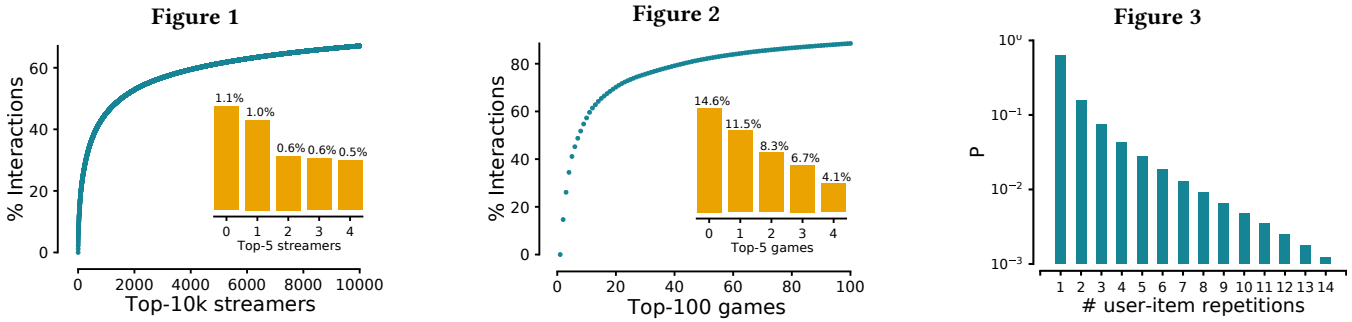
Video streaming platforms, such as *Twitch* or *Youtube Live*, are increasingly becoming a major part of people's daily lives. As of February 2020, Twitch reported 3 million broadcasters monthly and 15 million daily active users.[1] The increasing volume of concurrent broadcasts, the growing audience, as well as the long-tail of niche content, suggest the need for systems designed specifically for such platforms.

On live-streaming platforms, content creators broadcast video in real-time on their respective channels. The broadcast of real-time content[2] implies that videos can only be consumed at specific points in time. This dynamically evolving availability of streams presents challenges for traditional methods. Implicit feedback methods, trained to capture preference signals from positive interactions, make an underlying assumption that positive observations outrank those items the user never interacted with; when availability is dynamic this assumption no longer holds. Therefore, the explicit modelling of availability signals is required to distinguish between 'non-interaction' resulting from implicit preferences and from unavailable items.

A live-streaming model should also account for repeat consumption that represents a large portion of observed interactions, since users repeatedly consume content produced by the same streamers. This contrasts with typical recommendation domains (e.g. movies, e-commerce) that generally assume one-off user-item interactions. At the same time, it contrasts with existing lines of work on repeat consumption [1], since the content of channels is dynamic and differs at each new interaction.

In this work, we introduce the task of live-streaming recommendation with temporally evolving availability. We first provide preliminary experiments that demonstrate why existing methods are unsuited to this setting and, in particular, why naive sampling strategies are insufficient to capture user preferences. In light of these experiments, we introduce a self-attentive model, *LiveRec*, that learns to recommend live-streaming channels under availability constraints. Our model first selects candidates from the pool of available items at a specific point in time. These candidates are fed into a self-attention block that parametrizes relationships among available items. Historical interactions are modelled using a sequence encoder that is both used to select candidates and compute the final predictions, thus making the approach fully end-to-end.

[1] https://en.wikipedia.org/wiki/Twitch_(service)
[2] On some platforms, the content is also available on-demand, but we only consider the strict live-streaming setting in this work.



Left: CDF percentage of the top-1k streamers (items) in the dataset. The top-5 most popular streamers account for around 4% of the dataset. **Center:** CDF percentage of the top-100 games in the dataset. **Right:** Count of individual user-item pairs in the dataset.

In order to validate our approach, we introduce a large dataset of interactions from Twitch containing the consumption of logged-in users over a 43 day period. We identify key characteristics that differentiate our data from traditional settings, such as the high prevalence of recurring consumption, that we incorporate in our approach. To the best of our knowledge, this is the first publicly available dataset of content consumption detailing individual users' watching habits. We show that strong sequential baselines don't apply straightforwardly to this setting, and that our adaptations are necessary to achieve substantially better performance. We conclude with an in-depth analysis of the results and a discussion of the dynamics of live-streaming platforms.

2 RELATED WORK

In this section, we first review relevant lines of work from the literature on streaming platforms. Then, we describe relevant lines of work in the field of recommendation with an emphasis on sequential methods.

Streaming Platforms have been studied from a social dynamics perspective. The work from Hamilton et al. [6] investigates user motivations in joining live streaming channels. They conclude that, similar to on-demand video services, users are interested in a particular type of content, but also engage with the interactive characteristics of the service. Hilvert-Bruce et al. [10] report that compared to mass media, motivations of viewers on Twitch have a stronger social and community basis. They also suggest that viewers preferring small channels are more motivated by social engagement than users preferring large channels. Kaytoue et al. [13] characterize audience dynamics on the Twitch platform. Pires et al. [20] highlight the difficulty of identifying popular segments early on and show that there is no trivial solution to this problem. Nascimento et al. [19] identify several characteristic behaviors, such as the large audience drop at the end of a stream, investigate spectators assiduity and characterize the volume of comments in stream chats.

Temporal dynamics play a key role in the analysis of content consumption and have been studied in different scenarios. Early attempts focus on the modelling of long-term preference drifts [14, 36]. More recent approaches model evolving trends within the community of users in the context of fashion recommendation [7]. Another line of work focus on learning and inferring from temporally ordered data streams [2]. Several lines of work study the impact absolute position in the sequence and relative time intervals on performance [17, 38]. The CTA model [32] captures both temporal and contextual information by incorporating users' browsing activity. Wang et al. [29] incorporate various temporal patterns of repeat consumption using Hawkes processes. Wan et al. [28] model complementarity, compatibility and loyalty towards products in the grocery shopping domain. They propose AdaLoyal, a learning algorithm that explicitly account for users' must-buy purchases in addition to their overall preferences and needs. Anderson et al. [1] analyze the repeat consumption patterns on different social platforms. They propose a hybrid model that predicts user choice based on a combination of recency and quality. In this study, we consider repeat consumption of the same channel broadcasting new content, which differs from what past work has considered.

Recommendation systems model relationships between users and items [11, 15, 23] for explicit [16] and implicit feedback settings [21]. Sequential approaches infer user preferences from sequences of interactions. Recently, neural approaches have become popular for they high expressivity. Various research lines have attempted linear [8], recurrent [9], convolutional and graph approaches [31, 34, 37]. More recently, self-attention mechanisms, inspired by the field of natural language processing [27], have been investigated in the context of recommendation [3, 12, 25, 30, 35]. They generally do not model users explicitly and only learn from sequences of items. In Section 5.2, we discuss ranking refinement techniques that have been explored in the context of entity linking [4, 33].

3 DATA

In this section, we describe our data collection on Twitch in July 2019 over a 43-day period and give general statistics about the resulting dataset. In order to discover available channels, we queried the public Twitch API in rounds to list all available streams. The number of live streams ranged from around 20k to 75k for a single round during data collection. In each round, we also queried each available stream to get a list of connected users and the currently played game. In order to have sufficient time to query each stream in a single round, we set a 10 minute interval between each round. The

Figure 4

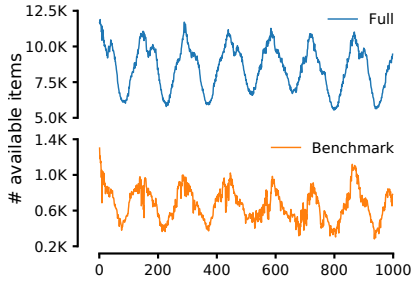


Figure 5

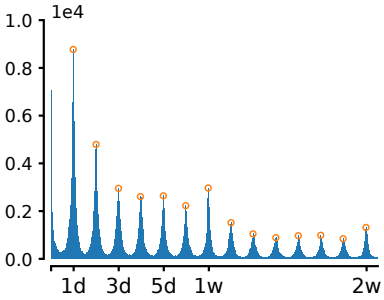


Table 1

	Bench.	Full
#Users	100k	15.5M
#Streamers	162.6k	465k
#Interactions	3M	474.7M
Watch time [h]	800k	124M
Density	9.2e-5	1.9e-5

Left: Number of available items for the first 1000 time steps of 10 minutes. **Center:** Time interval distribution (normalized) of repeated user-item interactions. **Right:** Datasets statistics. Watch time is estimated by considering periods of 5 minutes (half the duration of a round).

final dataset was collected over 6,148 rounds. Our dataset will be released in two formats: the *full* version, that contains all collected data and is more suited to data analysis tasks; and a *benchmark* version, that contains all interactions from 100k uniformly sampled users and that is used to compare the performance of different methods. All performance evaluations reported in this work use the *benchmark* version. Statistics before and after pre-processing can be found in Table 1. Both anonymized versions shall be made available at publication time. Both datasets exhibit skewed interaction distributions over items. For example, in the *full* version of the dataset, the most popular streamer drives more than 1% of the total interactions (see Fig. 1) and the most popular game drives 14.6% of the total interactions (see Fig. 2). Datasets contain absolute timestamps but we only consider relative time intervals in this work, thus mitigating time zones related side effects.

Repeat consumption is a common scenario in our dataset (see Fig. 3). Since Twitch is a social platform, content providers aim to grow and retain their respective audiences. We measure time intervals between any two interactions for the same user with the same streamer. We observe both daily and weekly dynamics, as seen in Fig. 5. We also observe short-term repeated interactions to be prevalent in our dataset.

4 WHAT IS DIFFERENT IN LIVE-STREAMING RECOMMENDATION?

Live-streaming differs from traditional scenarios in terms of the semantics of both positive and negative interactions. In this setting, negative examples can reflect either latent preferences, or simply the unavailability of a particular item (streamer) at interaction time. Additionally, several items appearing in a user history could become available simultaneously, which requires one to rank positive interactions among themselves. This differs from traditional settings where positive interactions included in the training set are discarded from the testing set. In this section, we present preliminary experiments that demonstrate the limitations of existing methods in this particular setting. Then, we propose an availability-aware sampling strategy that accounts for these observations.

4.1 Defining Items

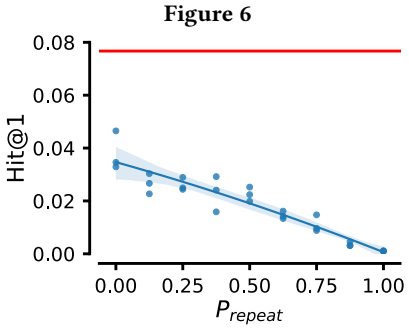
On a streaming platform, items could have different definitions: one could define items as *streams*, that are unique segments during which a streamer broadcasts uninterruptedly. The main problem of this case is the fact that streams only happen at a single point in time which lets the model learn in extremely sparse regimes. Alternatively, one could define items as *streamers*, that typically broadcast content multiple times. The main problem of this case is the fact that the model has to learn from multiple occurrences of the same user-item interaction, even if a streamer constantly broadcasts new content and evolves over time. In this work, we consider items representing streamers and discuss on how to account for repeat consumption during training and evaluation.

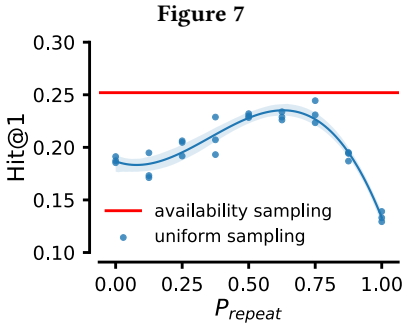
4.2 Preliminary Experiment: Repeat Consumption

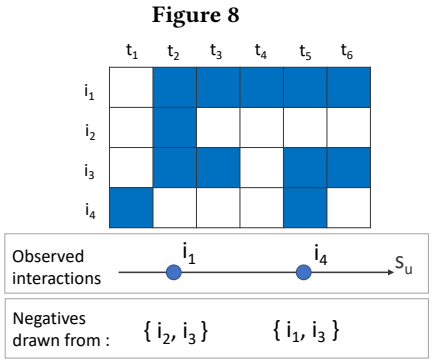
When splitting our dataset in the temporal dimension,[3] around 65% of user-item interactions observed in the testing period are also present in the training set. In such a setting, a model shall not only learn to accurately predict novel interactions but also to balance between novel and repeated interactions. Without yet considering extra features for distinguishing among repeated interactions (e.g. time, content), a simple model could balance interactions by their frequency, by favoring interactions occurring more often during training. A natural way to incorporate repetition information into a model is to develop a sampling strategy accounting for item frequencies.

We demonstrate the challenge of balancing between novel and repeat interactions through a preliminary experiment: we train a simple matrix factorization model with 20 latent dimensions using a ranking criterion [22] in a non-sequential setting. During training, for a user u and a positive item i, we sample negative examples from the pool of items consumed by u in the training set with a probability P_{repeat} and from a uniform distribution over all items with probability $(1 - P_{repeat})$, excluding i in both cases. During testing, we evaluate the model on its capacity to rank interactions with new streamers (Fig. 6), as well as its capacity to rank streamers appearing in the training sequence of the considered user (Fig. 7). We evaluate

[3]Considering the same setting as the experiments in Section 6, the last 250 rounds of 10 minutes (4%) of the dataset are withheld for testing.







Left: Performance for novel interactions, i.e. user-item pairs never seen among training sequences. **Center:** Performance for repeat interactions, i.e. user-item pairs present in training sequences. Uniform sampling experiments are repeated 3 times for each value of P_{repeat} and shown with a polynomial fit (95% CI). **Right:** Illustration of the availability matrix from which availability sampling is performed.

the model with the *Hit@1* metric by ranking all available items at the time of the last interaction of each user in the testing set. The best overall performance (0.157) is obtained at $P_{repeat} = 0.5$. This score represents a relative improvement of **16.1%** over a uniform sampling strategy (0.135). This observation shows the importance of accounting for repeat consumption in the considered setting. We also observe that our sampling strategy deteriorates predictions for novel items as P_{repeat} increases, which demonstrates the difficulty of balancing between novelty and repetition.

4.3 Preliminary Experiment: Availability

Availability signals are important to capture the meaning of 'non-interactions'; most recommendation systems relying on implicit feedback assume that the choice of item i over item j only reflects an implicit preference for i, since they also assume all items being available at all times. In a live-streaming setting, non-interaction with item j can reflect a preference for item i as much as item j simply being unavailable at interaction time. Similar to repeat consumption, availability signals can be accounted for through sampling, as we demonstrate in an additional experiment: we precompute availability from observed interactions as a matrix of size $n \times t_{max}$, where n is the total number of items and t_{max} is the total number of time steps of 10 minutes in our dataset (see in Fig. 8). Instead of sampling negative examples from all items, we sample a negative example j from the pool of available items at interaction time, excluding i. As such, we provide to the model additional context in which the interaction occurred. We also mitigate the impact of repeat interactions since multiple observations of the same user-item pair generally occur in different contexts. We evaluate this strategy on temporally disjoint training and testing sets, in order to avoid the model learning the availability matrix directly. We observed this strategy to provide **21.3%** (0.190) of relative improvement over the best performing sampling strategy presented in the last section. We select this strategy in all further experiments.

5 METHODS

Until now, we discussed ways to incorporate repeat consumption and availability information *implicitly*, through different sampling strategies. In this section, we discuss how to account for those signals by specific changes in model architecture.

5.1 Sequence Encoder

Let $s = (s_{p_1}, s_{p_2}, \ldots, s_{p_\ell})$ be a fixed-length user sequence, where s_p represents the p-th item entry. Sequences shorter than ℓ are padded with a null token and sequences longer than ℓ are cropped. The first key component of our model is a sequence encoder that converts each step s_p into a hidden representation vector $\mathbf{h}_{s_p}$. We use vector $\mathbf{h}_{s_{p-1}}$, that represents all previous entries in the sequence, for predicting the p-th item entry. We use an existing encoding technique, SASREC [12], even though our framework is not tied to this specific encoding scheme. First, we embed each item in s using an embedding matrix $\mathbf{M}$ of size $\mathbb{R}^{n \times h}$ where n is the number of items and h is the dimensionality of our latent space. We also encode positions in the sequence with an embedding matrix $\mathbf{P}$ of size $\mathbb{R}^{\ell \times h}$ that we learn during training. The resulting matrix $\hat{\mathbf{E}}$ is computed as the sum of item embeddings and positional embeddings:

$$\hat{\mathbf{E}} = \begin{bmatrix} \mathbf{M}_{s1} + \mathbf{P}_1 \\ \mathbf{M}_{s2} + \mathbf{P}_2 \\ \cdots \\ \mathbf{M}_{sl} + \mathbf{P}_2 \end{bmatrix}$$

Then, we pass the embeddings $\hat{\mathbf{E}}$ into two *query-key-value* self-attention layers in order to learn a relationship between elements of the sequence. Similar to the original implementation [12], we use layer normalization and apply a causality mask that prevents the model from learning from future interactions.

5.2 Modelling Availability

We seek a method that learns from the set of available items at the time of an interaction. Considering a user interacting with item i at time t, we learn from a set H_t containing all available items at t. Attention-based methods are typically of quadratic complexity; computing an attention function over the set H_t would generally be impractical, due to the large number of concurrently available items on a streaming platform. Therefore, we restrict this computation to a limited set of candidate items. Candidates are dynamically

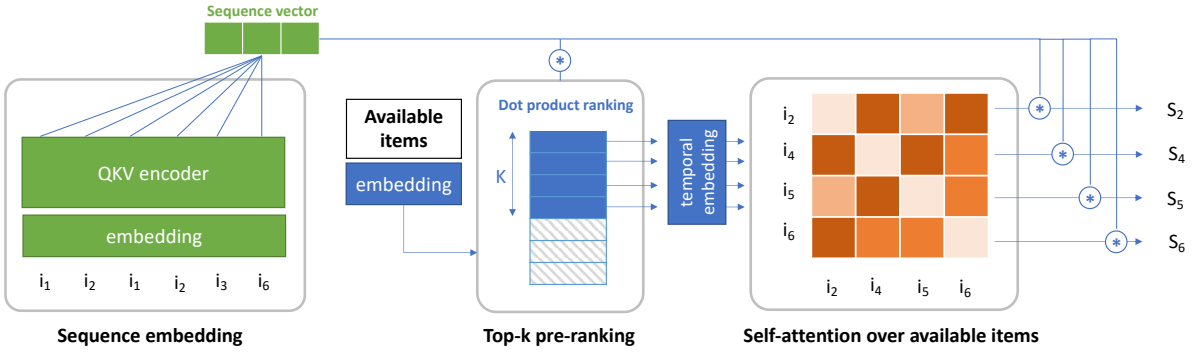


Figure 9: Illustration of the *LiveRec* architecture. First, the model encodes an input sequence of user interactions (left). Then, the model ranks, for each step of the sequence, all available items at the time of the interaction (center). Last, the model draws a dependency between a top-k selection of available items using a self-attention mechanism (right).

retrieved by computing a relevance score $s(\cdot)$ for each available item in H_t. For predicting the p-th item entry, this score is obtained from a dot product operation between the sequence embedding and the item embedding vector.

$$s(i, p, t) = \mathbf{h_{p-1}} \cdot \mathbf{M_i}, \qquad \mathbf{i} \in \mathbf{H_t}$$

Then, we sort all items in H_t by their score $s(\cdot)$, select the top-k highest scoring items and represent them as a sequence $(r_1, r_2, \ldots, r_k)$. We use matrix $\mathbf{M}$ to embed each of the top-k elements into a matrix $\mathbf{M}_{\text{av}} \in \mathbb{R}^{k \times h}$. Our objective is to learn an attention function $f : \mathbb{R}^{k \times h} \mapsto \mathbb{R}^{k \times h}$ over $\mathbf{M}_{\text{av}}$, in order to draw a dependency between each of the k-selected items. We adopt the widely used *query-key-value* form of self-attention and use the matrix $\mathbf{M}_{\text{av}}$ as input for queries, keys and values.

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}})\mathbf{V},$$

$$\mathbf{Q} = \mathbf{K} = \mathbf{V} = \mathbf{M}_{\text{av}} = \begin{bmatrix} \mathbf{M}_{r_1} \\ \mathbf{M}_{r_2} \\ \cdots \\ \mathbf{M}_{r_k} \end{bmatrix}$$

The factor $\sqrt{d}$ is introduced to avoid overly large values of the inner product $\mathbf{Q}\mathbf{K}^T$ [27]. We use layer normalization, residual connections and dropout similar to SASREC [12]. We experimented with this approach with one and two layers of attention (see section 6). Because the absolute position in the sequence $(r_1, r_2, \ldots, r_k)$ is irrelevant to the task, we do not encode positions and do not use any masking in this attention stage.

5.3 Modelling Repeat Consumption

We design an encoding scheme that distinguishes between novel and repeated interactions. In the repeat case, our scheme also encodes recency: for an item $i \in H_t$, we retrieve the time t_j of its last occurrence prior to t. We define $\mathbf{q_i} \in \{0, 1\}^{z+1}$, a one-hot encoded vector that represents a mapping from the time interval $|t - t_j|$ to a bucket index in the range $[1, z]$. Based on our observations in Section 3, we divide time intervals into buckets of 24 hours and clip the maximum time interval to 20 days. We keep index zero of $\mathbf{q_i}$ for representing novel item entries with no previous occurrences in the sequence. We embed time intervals represented by $\mathbf{q_i}$ using an embedding matrix $\mathbf{T} \in \mathbb{R}^{(z+1) \times h}$. Then, we combine the corresponding time interval embedding vector to the candidate item vector using element-wise addition (see Fig. 10). We use this new representation for the selection of candidates and the attention stage.

$$s(i, p, t) = \mathbf{h_{p-1}} \cdot (\mathbf{M_i} + \mathbf{q_i}\mathbf{T}), \quad \mathbf{i} \in \mathbf{H_t}$$

$$\mathbf{Q} = \mathbf{K} = \mathbf{V} = \mathbf{M}_{\text{av+rep}} = \begin{bmatrix} \mathbf{M}_{r_1} + \mathbf{q}_{\mathbf{r}_1}\mathbf{T} \\ \mathbf{M}_{r_2} + \mathbf{q}_{\mathbf{r}_2}\mathbf{T} \\ \cdots \\ \mathbf{M}_{r_k} + \mathbf{q}_{\mathbf{r}_k}\mathbf{T} \end{bmatrix}$$

Time interval embeddings are added prior to the self-attention stage for the model to learn from temporal dependencies among candidates. The self-attention module outputs, for each candidate item, a new representation that encodes a relationship with other candidates. We compute the final score $\hat{x}_{u,i,t}$ for each of the k candidates by computing a dot product between $\mathbf{h_{p-1}}$ and each of the k rows of matrix $\mathbf{O} = \text{Attention}(\mathbf{M}_{\text{av+rep}}, \mathbf{M}_{\text{av+rep}}, \mathbf{M}_{\text{av+rep}}), \mathbf{O} \in \mathbb{R}^{k \times h}$.

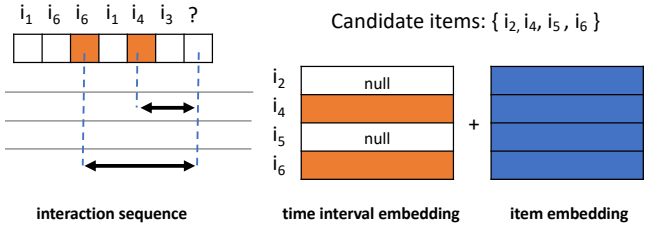


Figure 10: Illustration of the time interval embedding module. Items i_2 and i_5 are novel entries. Embeddings for items i_4 and i_6 are based on the time difference with their last respective occurrences.

5.4 Training

Following existing sequential recommendation approaches, we compute predictions using mini-batches of sequences. During training,

the model predicts the next item of each sequence step in a single forward pass. We train the model to maximize the difference in score between positive and negative instances through negative sampling. Instead of sampling items uniformly, and according to our observations in Section 4, we draw negative examples from a set $H_{i,t}$ for a positive item i at time t.

$$H_{i,t} = \{j \in A_t \wedge j \neq i\}$$

Given a positive i and a negative items j that fulfill the constraint described above, we adopt the following cross entropy loss to train the model.

$$-\sum_{s} \sum_{(i,t) \in s} \log(\sigma(\hat{x}_{u,i,t})) + \sum_{j \in H_{i,t}} \log(1 - \sigma(\hat{x}_{u,j,t}))$$

During training, we use vector $\mathbf{h_{p-1}}$ for both selecting candidates and predicting the next entry in the sequence which makes the approach fully *end-to-end*.

6 EXPERIMENTS

6.1 Evaluation

In order to avoid learning from future interactions, we split the dataset in three distinct time intervals. We withhold 250 time steps for validation and 250 time steps for testing, both by splitting from the end of the dataset. We evaluate all approaches using the metrics *hit@1*, *hit@10* and *NDCG@10* on the last interaction of each sequence s in the testing period.

Additionally, we break down this score into a *Hit-new* score and a *Hit-rep* score in Table 2. By doing so, we evaluate the model on its capacity to repeat an interaction from the input sequence, as well as its capacity to recommend serendipitous content. A testing interaction falls into *rep* if the item appears in the testing input sequence, and in *new* otherwise. For a sequence length of 16, the percentage of repeat consumption is equal to 51%.

Our experiments are conducted with different variants of our model: *LiveRec + rep* is a variant that only uses time interval embeddings on top of a sequence encoder. *LiveRec + av* is a variant including self-attention over a top-k selection of items. *LiveRec + rep + av* is our final model, as described in Section 5, using both self-attention and time interval embedding. *LiveRec + rep* does not use any candidate selection strategy and ranks all available items H_t instead of a top-k selection.

6.2 Baselines

In this section, we compare various baselines with our approach in order to evaluate existing methods in a live-streaming setting (see Section 4) and demonstrate the benefits of the proposed architecture.

- **REP** is a simple model that predicts a score equal to the number of appearances of an item in the input sequence. *REP* is a strong predictor of repeated interactions but it is unable to recommend new items.
- **POP** this model gives a score equal to the popularity of an item in the training set. It does not consider the interaction sequence to compute predictions.
- **MF-BPR [21]** is a matrix factorization model trained with a ranking criterion. This model does not account for sequential or temporal dynamics.
- **FPMC [22]** is a sequential recommendation method that models transitions in terms of the last entry in the sequence. The model is trained using a BPR loss [21].
- **SASREC [12]** is a self-attentive recommendation method. Multiple query-key-value attention layers are stacked together to capture relevant information from the interaction sequences. Positional encoding is used to help the model encoding temporal information. We referred to as *SASREC - uniform* a model trained with negative samples drawn uniformly over all items.
- **BERT4REC [25]** is a recent adaptation of the BERT language model. We modified a publicly available implementation[4] in order to run the model using the same experimental setting. Similar to the original training environment, we define the masking probability ρ and predict masked items only. In order to be fair towards other approaches, we implement a *masked* cross-entropy loss that only backpropagates over available items.

6.3 Experimental Setting

We train all models until the score does not improve for 10 epochs on the validation set and store the model at each epoch. Then, we test with the checkpoint having the highest (validation) *hit@1* and evaluate performance on the testing set. We consider ℓ_2-regularization in the range $\ell_2 = \{0.0001, 0.001, 0.01, 0.1, 1.0\}$. All models are implemented in Pytorch and trained using the Adam optimizer with a learning rate of 0.0005. All attention-based approaches are trained with a fixed dimensionality of 128. Other methods are trained with a dimensionality in the range $\{16, 32, 64, 128\}$. Batch size is fixed to 100. For BERT4REC, we consider ρ in the range $\{0.25, 0.5, 0.75\}$ and results are reported with $\rho = 0.25$. We filter out users with fewer than 5 interactions and streamers with fewer than 3 interactions. If the input testing sequence is shorter than the maximum length l, we append interactions from the validation and the training set. *All code and data shall be released at publication time* [5].

6.4 Overall Performance Comparison

In this section, we discuss the results obtained by various architecture in Table 2.

We first notice that *REP*, which only predicts future interactions by repeating elements from the input sequence, provides a reasonably competitive score. Since repeated interactions account for more than 50% of the testing data, *REP* represents an effective strategy for recommending content without a parametrized method, despite its inability to recommend new content. This result also shows the importance of measuring the two metrics *Hit-new* and *Hit-rep* individually.

LiveRec rep + av provides the best overall score. However, we notice that *SASREC* leads to a higher *Hit@1-new* and that *LiveRec + av* leads to a higher *Hit@10-new*. This result shows that there is still margin for improvement in balancing between novel and repeated

[4] https://github.com/jaywonchung/BERT4Rec-VAE-Pytorch
[5] https://github.com/JRappaz/liverec

Model	H@1	H@1-new	H@1-rep.	H@10	H@10-new	H@10-rep.	NDCG@10
POP	0.0317	0.0237	0.0387	0.1350	0.1006	0.1650	0.0754
REP	0.3698	0.0166	0.6776	0.5347	0.0424	0.9637	0.4630
MF-BPR	0.0363	0.0279	0.0436	0.1537	0.1182	0.1848	0.0879
FPMC	0.0690	0.0372	0.0968	0.2515	0.1662	0.3258	0.1529
SASREC - uniform	0.1994	0.0721	0.3103	0.5827	0.3888	0.7517	0.3733
SASREC	0.3004	**0.1180**	0.4593	0.7221	0.5156	0.9021	0.5014
BERT4REC	0.3517	0.1018	0.5694	0.7089	0.4668	0.9199	0.5237
LiveRec + rep	0.3655	0.0686	0.6241	0.7581	0.4907	0.9912	0.5615
LiveRec + av	0.3357	0.1067	0.5352	0.7363	**0.5222**	0.9229	0.5303
LiveRec + rep + av	**0.4122**	0.0920	**0.6913**	**0.7655**	0.4998	**0.9970**	**0.5893**

Table 2: Results for all considered approaches. The best performing method in each column is boldfaced.

interactions. Compared to the sequence encoder alone (*SASREC*), the additional modelling of repeat consumption (*rep*) and availability (*av*) lead to a relative improvement of **21.7%** and **11.8%**, respectively. The combination of the two leads to a relative improvement of **37.2%** over the sequence encoder alone (*SASREC*). The introduction of the time interval embedding (*rep*) encourages the model to favor repeated interactions which hurts novelty (*Hit@1-new*). However, incorporating self-attention (*rep + av*) over available items mitigates this effect. Compared to *REP*, *LiveRec* scores a higher *Hit@1-rep*, which suggests a better capacity to rank multiple repeat consumption options than a simple frequency-based ranking.

BERT4REC provides higher performances than *SASREC*. This observation in accordance with the original paper [25]. We notice that the gain in performance comes primarily from a better capacity to model repeat consumption. We also emphasize that masking unavailable items in the loss function is critical to obtain this result. In order to compare results with our observations from Section 4, we trained *SASREC* with two different sampling strategies: uniform sampling and availability sampling. We observe a significant improvement by sampling negative interactions from the set of currently available items, in accordance with our preliminary results.

6.5 Analysis

In light of the results presented in Table 2, we seek to further elaborate on the influence of various factors on performance through the following questions.

Question 1: *What is the influence of the number of candidates k on performance?*

The number of candidates k is a critical parameter of our approach since only candidate items are considered in the final ranking. Therefore, this parameter should be sufficiently large to cover a broad spectrum of candidate items, the ranking of which will be refined by the self-attention module. We obtain the best results with $k = 128$ [6] (see Fig. 12). We also experiment with 1 and 2 attention layers. A single layer module performs better for small values of k but fails to scale to a larger number of candidates. We hypothesize this phenomenon to be due to the self-attention module being only *distantly* personalized: it is trained on a selection of candidate items instead of raw interactions, a complex mapping that could be successfully captured with multiple layers of attention.

Question 2: *What is the influence of sequence length on performance?*

Increasing sequence lengths monotonically increases performance for our model (see Fig. 13). This gain diminishes, as we increase sequence length, since the average sequence length of the training dataset is equal to 28 and sequences shorter than this maximum length are padded. The results presented in Table 2 are consistent over all sequence lengths in Fig. 13. However, as we increase sequence length, we also increase the percentage of repeated interactions between users and streamers. For simplicity, we perform the full evaluation, presented in Table 2, at around 50% of repeated interactions and leave as future work a more in-depth analysis of the impact of this ratio on performance.

Question 3: *What is the popularity distribution of recommended items with LiveRec?*

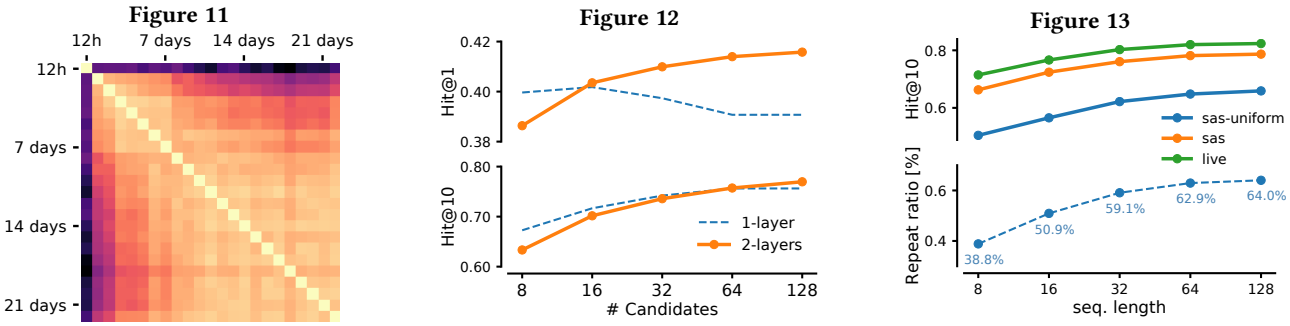
Mitigating popularity bias has become a concern in the recommender systems literature [5, 18, 24]; recommendation algorithms are known for recommending popular items frequently while ignoring items in the long tail. This situation could be problematic in a live-streaming setting, where the interaction distribution over items is already extremely skewed (see Fig. 1). Therefore, we investigate the relationship between the introduction of the self-attention mechanism presented in Section 5.2 and the popularity of recommended items.

In general, the introduction of the attention mechanism leads to a decrease in popularity of the recommended content. Specifically, we observe a reduction in the average popularity of recommended items of **17.1%** with the introduction of self-attention (*av*) and **20.3%** with time interval embedding (*av + rep*), compared to the sequence encoder alone (*SASREC*). As one can observe in Fig. 15, the introduction of *rep + av* leads to a popularity distribution of recommended items that matches the observed distribution of interactions more closely.

Question 4: *What types of dynamics are captured by time interval embeddings?*

In order to characterize temporal patterns captured by *LiveRec*, we compute the similarity matrix between embedding vectors after

[6] In our case, the maximum value that can fit on a single GPU



Left: Cosine similarity between time interval embeddings. **Center:** Performance with different numbers of pre-ranking candidates with 1 and 2 layers of attention. **Right:** Impact of sequence length on performance (top) and percentage of repeat interactions (bottom).

a full training phase (see Fig. 10). First, in accordance with our observations in Section 3, the first bucket, representing a time interval of less than 12 hours, is different from all other vectors. This observation suggests the dynamics of repeated interactions during the same day to be governed by distinct temporal dynamics (e.g. users returning to the same stream after a few minutes). Second, we observe a visible pattern for intervals of less than one week. This observation is in keeping with the weekly dynamics observed in Section 3. Finally, vectors representing time intervals of more than one week become increasingly similar as time intervals increase, which suggests that the model captures a unified representation of long-term repeat patterns.

Question 5: *What types of dynamics are captured by the attention module over available items ?.*

We compare item embeddings, learned from user sequences, and attention weights (*av*), learned from the context of available items. We embed the top-1k most popular items using matrix $\mathbf{M}$ and provide those items as (*query*, *key*, *value*) inputs of the self-attention module. The attention weights between *query* and *key* items are shown in Fig. 14 (left). In order to observe the semantic relatedness of those items, we also show a 2D projection (t-SNE [26]) of their embedding vectors and select 4 clusters (center). We observe each cluster of items exhibiting a distinct patterns of attention: attention weights are consistent within clusters but strongly vary from one cluster to another. We also observe the model being able to give attention weight to *query* and *key* items with low content similarity (i.e. belonging to different clusters), which shows the ability of the model to learn this relationship between semantically unrelated items. Considering that the attention module (*av*) is only *distantly* supervised, since it only learns from a candidate selection of items, our observations suggest that it captures a different, and more global availability context compared to the sequence embedding module.

7 DISCUSSION AND FUTURE WORK

The growing audiences of live-streaming platforms emphasize the need for efficient retrieval and recommendation methods. In this work, we focused our modelling efforts on a dataset of limited size. Scaling our approach, especially to a production environment, would require extra considerations. First, the fast retrieval of available items represents a barrier to large scale studies. For the training on the *benchmark* dataset, we maintained availability sets in memory but this approach might reach its limits with larger applications and would require a more scalable design for retrieving and sampling available items. Second, the additional attention module (*av*) increases the complexity of the model, even though the module has the same architecture as the sequence encoder, making the additional complexity a constant factor. For example, introducing *av* in the model[7] reduces the training speed from 8.5 it/s to 5 it/s. In order to account for this observation, one must carefully select sequence length, as well as the number of candidate items, in order to balance between accuracy and training time. Finally, scaling the approach to larger datasets might require more complex training strategies. For example, semi-supervised learning could be employed to first learn item representations on a large dataset, before fine-tuning a more costly approach on a smaller subset of data.

To the best of our knowledge, our work is the first attempt at modelling view dynamics on live-streaming platforms and many challenges remain. First, content-based methods have not yet been investigated. For example, visual features could potentially be exploited to characterize the various segments of a stream. Second, we believe that user dynamics could be further exploited. In this work, we focused on sequential methods that model item-to-item relationships. The dual scenario, the modelling of user-to-user interactions, remains unexplored. In particular, we believe that the modelling of users currently watching a stream could help improving performance. Since the number of users is, by an order of magnitude, higher than the number of streamers, future research should design efficient methods to learn from user interactions. Third, during our analysis, we noticed bursts of activity around specific channels. We believe that the traditionally static notion of item popularity could be adapted to a dynamic setting. For example, the number of concurrent users watching a stream could be exploited at inference time. Fourth, while we focus on *SASREC* for encoding historical data, our framework is not tied to any specific method for learning from interaction sequences. Future research could swap the encoder with a different encoding method (e.g. graph-based) to capture a richer

[7] For a sequence length of 16 and a 16 candidate items.

Figure 14

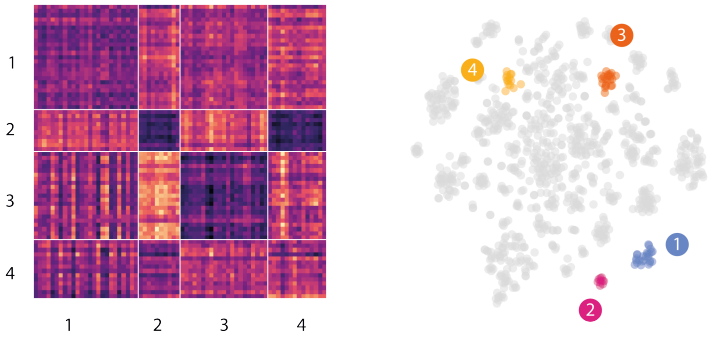
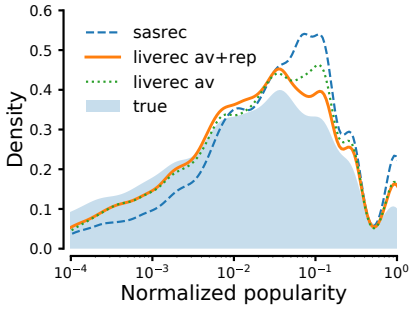


Figure 15



Left: Example of self-attention over available items: the top-1k most popular items are projected using t-SNE and are used as an input for the attention module; the attention weights are displayed after the softmax operation. The four selected clusters shown in the projection correspond to the rows (queries) and columns (keys) of the weight matrix. **Right:** Normalized popularity distribution of recommender items, alongside with the true popularity of consumed items.

context. Finally, many social aspects on live-streaming platforms remain to be explored. Our dataset could help to better understand the social dynamics taking place on a streaming platform. Going beyond the live-streaming setting, we believe that our approach could be leveraged for other types of applications, such as digital TV, and in settings where items only remain available for a limited period of time, such as the front page of a news website.

8 CONCLUSION

In this work, we introduced live-streaming recommendation, a scenario in which items are not always available for users to consume. We showed that a sampling strategy simulating this evolving availability is crucial to capture user preferences. Moreover, we described how to incorporate the notion of availability into our model architecture by performing an explicit comparison among available items. In order to account for the large number of concurrent broadcasts, we made this process efficient by comparing only on a subset of candidate streams. We also investigated repeated user interactions with streamers and proposed a way to model this phenomenon with time interval embeddings, which we show to improve performance. With the release of a large dataset of user interactions on Twitch, the various improvement over existing methods, our study paves the way for new research in live-streaming recommendation.

REFERENCES

[1] Ashton Anderson, Ravi Kumar, Andrew Tomkins, and Sergei Vassilvitskii. 2014. The dynamics of repeat consumption. In *Proceedings of the 23rd international conference on World wide web*. 419–430.

[2] Shiyu Chang, Yang Zhang, Jiliang Tang, Dawei Yin, Yi Chang, Mark A Hasegawa-Johnson, and Thomas S Huang. 2017. Streaming recommender systems. In *Proceedings of the 26th international conference on world wide web*. 381–389.

[3] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. 2017. Attentive collaborative filtering: Multimedia recommendation with item-and component-level attention. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. 335–344.

[4] Nicola De Cao, Ledell Wu, Kashyap Popat, Mikel Artetxe, Naman Goyal, Mikhail Plekhanov, Luke Zettlemoyer, Nicola Cancedda, Sebastian Riedel, and Fabio Petroni. 2021. Multilingual Autoregressive Entity Linking. *arXiv preprint arXiv:2103.12528* (2021).

[5] Priyanka Gupta, Diksha Garg, Pankaj Malhotra, Lovekesh Vig, and Gautam Shroff. 2019. NISER: Normalized item and session representations to handle popularity bias. *arXiv preprint arXiv:1909.04276* (2019).

[6] William A Hamilton, Oliver Garretson, and Andruid Kerne. 2014. Streaming on twitch: fostering participatory communities of play within live mixed media. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*. ACM, 1315–1324.

[7] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*. 507–517.

[8] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[9] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).

[10] Zorah Hilvert-Bruce, James T Neill, Max Sjöblom, and Juho Hamari. 2018. Social motivations of live-streaming viewer engagement on Twitch. *Computers in Human Behavior* 84 (2018), 58–67.

[11] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*. Ieee, 263–272.

[12] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 197–206.

[13] Mehdi Kaytoue, Arlei Silva, Loïc Cerf, Wagner Meira Jr, and Chedy Raïssi. 2012. Watch me playing, i am a professional: a first study on video game live streaming. In *Proceedings of the 21st International Conference on World Wide Web*. ACM, 1181–1188.

[14] Yehuda Koren. 2009. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 447–456.

[15] Yehuda Koren and Robert Bell. 2015. Advances in collaborative filtering. *Recommender systems handbook* (2015), 77–118.

[16] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[17] Jiacheng Li, Yujie Wang, and Julian McAuley. 2020. Time interval aware self-attention for sequential recommendation. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 322–330.

[18] Masoud Mansoury, Himan Abdollahpouri, Mykola Pechenizkiy, Bamshad Mobasher, and Robin Burke. 2020. Feedback loop and bias amplification in recommender systems. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2145–2148.

[19] Gustavo Nascimento, Manoel Ribeiro, Loïc Cerf, Natália Cesário, Mehdi Kaytoue, Chedy Raïssi, Thiago Vasconcelos, and Wagner Meira. 2014. Modeling and

analyzing the video game live-streaming community. In *2014 9th Latin American Web Congress*. IEEE, 1–9.

[20] Karine Pires and Gwendal Simon. 2015. YouTube live and Twitch: a tour of user-generated live streaming systems. In *Proceedings of the 6th ACM multimedia systems conference*. ACM, 225–230.

[21] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618* (2012).

[22] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*. 811–820.

[23] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to recommender systems handbook. In *Recommender systems handbook*. Springer, 1–35.

[24] Ana-Andreea Stoica, Christopher Riederer, and Augustin Chaintreau. 2018. Algorithmic Glass Ceiling in Social Networks: The effects of social recommendations on network diversity. In *Proceedings of the 2018 World Wide Web Conference*. 923–932.

[25] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*. 1441–1450.

[26] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762* (2017).

[28] Mengting Wan, Di Wang, Jie Liu, Paul Bennett, and Julian McAuley. 2018. Representing and recommending shopping baskets with complementarity, compatibility and loyalty. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 1133–1142.

[29] Chenyang Wang, Min Zhang, Weizhi Ma, Yiqun Liu, and Shaoping Ma. 2019. Modeling item-specific temporal dynamics of repeat consumption for recommender systems. In *The World Wide Web Conference*. 1977–1987.

[30] Shoujin Wang, Liang Hu, Longbing Cao, Xiaoshui Huang, Defu Lian, and Wei Liu. 2018. Attention-based transactional context embedding for next-item recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[31] Wen Wang, Wei Zhang, Shukai Liu, Qi Liu, Bo Zhang, Leyu Lin, and Hongyuan Zha. 2020. Beyond clicks: Modeling multi-relational item graph for session-based target behavior prediction. In *Proceedings of The Web Conference 2020*. 3056–3062.

[32] Jibang Wu, Renqin Cai, and Hongning Wang. 2020. Déjà vu: A contextualized temporal attention mechanism for sequential recommendation. In *Proceedings of The Web Conference 2020*. 2199–2209.

[33] Ledell Wu, Fabio Petroni, Martin Josifoski, Sebastian Riedel, and Luke Zettlemoyer. 2020. Zero-shot Entity Linking with Dense Entity Retrieval. In *EMNLP*.

[34] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. 2019. Session-based recommendation with graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 346–353.

[35] Jun Xiao, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, and Tat-Seng Chua. 2017. Attentional factorization machines: Learning the weight of feature interactions via attention networks. *arXiv preprint arXiv:1708.04617* (2017).

[36] Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff Schneider, and Jaime G Carbonell. 2010. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of the 2010 SIAM international conference on data mining*. SIAM, 211–222.

[37] Chengfeng Xu, Pengpeng Zhao, Yanchi Liu, Victor S Sheng, Jiajie Xu, Fuzhen Zhuang, Junhua Fang, and Xiaofang Zhou. 2019. Graph Contextualized Self-Attention Network for Session-based Recommendation.. In *IJCAI*, Vol. 19. 3940–3946.

[38] Wenwen Ye, Shuaiqiang Wang, Xu Chen, Xuepeng Wang, Zheng Qin, and Dawei Yin. 2020. Time matters: Sequential recommendation with complex temporal information. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1459–1468.