NxtPost: User to Post Recommendations in Facebook Groups

Kaushik Rangadurai, Yiqun Liu, Siddarth Malreddy, Xiaoyi Liu,
Piyush Maheshwari, Vishwanath Sangale, Fedor Borisyuk
Meta Platforms Inc.

ABSTRACT

In this paper, we present *NxtPost*, a deployed user-to-post content-based sequential recommender system for Facebook Groups. Inspired by recent advances in NLP, we have adapted a Transformer-based model to the domain of sequential recommendation. We explore causal masked multi-head attention that optimizes both short and long-term user interests. From a user's past activities validated by defined safety process[1], *NxtPost* seeks to learn a representation for the user's dynamic content preference and to predict the next post user may be interested in. In contrast to previous Transformer-based methods, we do not assume that the recommendable posts have a fixed corpus. Accordingly, we use an external item/token embedding to extend a sequence-based approach to a large vocabulary. We achieve **49%** abs. improvement in offline evaluation. As a result of *NxtPost* deployment, 0.6% more users are meeting new people, engaging with the community, sharing knowledge and getting support. The paper shares our experience in developing a personalized sequential recommender system, lessons deploying the model for cold start users, how to deal with freshness, and tuning strategies to reach higher efficiency in online A/B experiments.

CCS CONCEPTS

• **Information systems** → **Retrieval models and ranking**.

KEYWORDS

Sequential Recommender system, User History Modeling, Session-based Recommender system, Recommender system

ACM Reference Format:
Kaushik Rangadurai, Yiqun Liu, Siddarth Malreddy, Xiaoyi Liu,, Piyush Maheshwari, Vishwanath Sangale, Fedor Borisyuk. 2022. NxtPost: User to Post Recommendations in Facebook Groups. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3534678.3539042

1 INTRODUCTION

Facebook Groups[2] is a global platform that enables individuals with common interests to form communities and share their experiences (Fig. 1). Hundreds of millions of people engage on the platform each day [18]. Some groups may encompass conversations of diverse and general interests, whereas others cater to specific topics such as gaming, parenting, social learning, etc. Group members interact and share ideas by posting or commenting on a variety of content such as photos, videos, web links, and text.

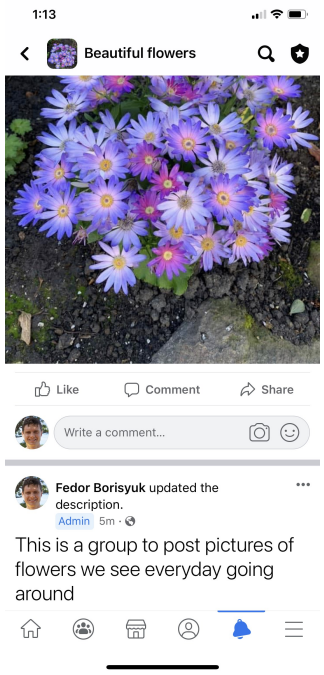

Figure 1: A screenshot of a Facebook Group about flowers.

To foster greater connection between individuals and communities, we are interested in developing a personalized recommender system for Facebook Groups. In particular, we aim to recommend publicly visible group posts to Facebook users for their enjoyment based on their respective content preferences. We formulate the objective as a sequential recommendation problem wherein a user's historical activity patterns are used in conjunction with static user features to predict the next group post that may likely interest the user. An activity history comprises dynamic content interactions such as likes, reactions, comments, and reshares. Examples of static user features include predicted language and home country. We call this recommender system *NxtPost*.

There were several challenges in building *NxtPost*:

- *Cardinality of posts:* Billions of posts are created each day with hundreds of millions of them engaged daily [2]. Therefore the cardinality of items to recommend pose a challenge in comparison to other production recommendation systems. While some try to solve the problem by recommending the

[1]Integrity violating posts filtered out from the data according to safety procedures [1].
[2]https://www.facebook.com/groups

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539042

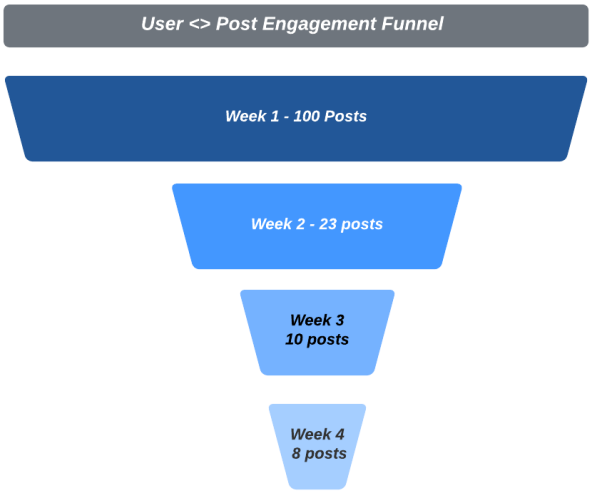

Figure 2: Posts have a short shelf-life. Out of 100% of posts engaged on the first week only 23% continue to be engaged a week after and only 10% of them 2 weeks after.

top N most engaged posts, the cardinality is still in the order of hundred millions.

- *Volatility of posts:* We observed that most posts have a short shelf-life, meaning that there is little overlap between engaged posts week over week. This is in contrast to most modern recommender systems which deal with relatively fixed recommendation set.
- *Many types of engagement across multiple surfaces:* There are many forms of user engagement on Facebook that are explicit signals of relevance such as likes/reacts, shares, views, clicks, leaving a comment, or even liking a comment or commenting on a comment. These user engagements also occur across different devices and surfaces/tabs within Facebook. Techniques that use engagement as a target signal have to contend with how to deal with different sources of relevance that are not directly comparable.

To demonstrate the issue of volatility of posts we show how post engagement declines over time in Fig. 2. Posts have a temporary nature; out of 100% of posts engaged in a given week, only 23% continue to be engaged the following week and only 10% continue to be engaged two weeks later. We will next describe our model *NxtPost* and how it addresses each of these aforementioned challenges.

This paper presents *NxtPost*, a recommendation system that recommends posts to a user by predicting which post would likely come next in the sequence of the posts recently engaged by the user. We filter out integrity violating posts from the data according to defined safety procedures [1]. Within *NxtPost* we extend the idea of modeling sequences of words to the more general notion of modeling sequences of complex objects—in the form of posts—which contain text and multi-media. *NxtPost* has demonstrated gains in production applications by introducing several modeling techniques described section § 3. In order to deal with a large vocabulary, we've removed the learnable token embeddings in the Transformer layer and replaced it with a pre-trained item embedding. We've also removed the classification layer and used a Two Tower architecture instead. We've also explored causal attention and 2 losses to optimize both the short and long-term user interests. We achieve over 49% absolute offline metrics improvement in comparison to previous state-of-the-art modeling approaches. We share our modeling techniques in § 3, ablation studies in § 4, and production deployment experience and tuning tricks to achieve higher performance in online A/B experiments in § 6.

NxtPost has been deployed to Facebook Groups; it powers user-to-post embedding-based retrieval. *NxtPost* operates at Facebook Groups scale with hundreds of millions of users [18] consuming results of *NxtPost* recommender system. As a result of *NxtPost* deployment, 0.6% more people are meeting new people, engaging with the community, sharing knowledge and getting support.

2 RELATED WORK

Personalized Content Recommender Systems. Collaborative filtering (CF) is a well-researched technique that has been adopted by a number of large-scale consumer applications [5, 13–15, 19, 20]. CF-based recommender systems traditionally apply transductive learning on a ⟨user, content⟩ bipartite graph to identify content of interest to each user. However, if large number of new nodes are continuously added to the graph, the effectiveness of a CF-based system to suggest fresh and relevant content is markedly reduced by challenges related to cold-start [21]. Thus, we opt for a content-based approach to empower a user-to-content recommendation system for Facebook Groups.

Graph Convolutional Network (GCN) has attracted industry attention in recent years [12]. Ying et al. [27] used graph structure to aggregate content representations for related-content recommendations and demonstrated improvement in both offline and online settings. A GCN-based system typically derives node representation from a weighted bag of neighboring embeddings. By contrast, in *NxtPost* we take into consideration the position of each post in the user's interaction history and learn the weight of each post through attention. Further analysis on the correlation between future engagement actions and post positions is provided in § 3.3.

Transformers have been explored in the domain of Sequential Recommendation. Kang and McAuley [11] used causal attention that mimics a language model to learn the user representation while Sun et al. [22] used masked language model to learn a user representation. However, both these techniques assumed that the number of items in the Transformer layer were limited and doesn't change quickly. By contrast, *NxtPost* uses combines a TwoTower approach with causal language modeling techniques to deal with a large dataset size at Facebook.

Sequential Modeling. Two-tower neural networks have been widely used to model the semantic relevance between heterogeneous data types [3, 8, 17, 25]. Yi et al. [26] used recently watched videos as an input feature to the user tower, but the viewer history was represented by the average video ID embeddings instead of the watch sequence. Tang et al. [23] explored sequence encoding and showed improvement to recommendation quality by incorporating encoders of different temporal ranges. For *NxtPost*, we provide an ablation study on sequence length in § 4.1.

There have been some recent works on sequence modeling inspired by BERT. Sun et al. [22] proposed a BERT-like pre-training approach by randomly masking some items in the input sequences and then predicting the IDs of those masked items based on their surrounding context. In this paper, we share our experience in developing Transformer-based user-to-content models. In contrast to existing works, we extend the transformer usage for user-to-content prediction to unlimited vocabulary. Zhai [29] proposed to use transformers with retrieval losses [9], which improves performance of recommendation system further. We have observed improvements in model performance by adopting retrieval losses in our implementation (§3.2). In lieu of content IDs, we construct input features from pre-trained content embeddings and apply transformers to prepare user representations. We provide an ablation study of different model configurations in comparison with prior works (§ 4). We report significant improvement in key metrics with *NxtPost* over existing approaches.

3 MODELING

In this section, we describe the model architecture of *NxtPost* and also explain how we collect training data and optimize the model. Central part of *NxtPost* is a transformer encoder architecture with causal multi-head attention and support to input pre-trained item embeddings. We go over the details of these techniques and explain how we translate it to the domain of sequential recommendation.

3.1 *NxtPost* Model Architecture

NxtPost is a Two Tower / Dual Encoder architecture with in-batch negatives. It has a learn-able user tower and a fixed (pre-trained) post tower and has in-batch negatives. Before we get into the details of the model architecture, let us first understand what is a Two Tower architecture and how it works. In order to explain how two tower architectures work, we'll explain what happens during training in a single batch of batch size B. In the TwoTower model, we apply the user tower obtain user embeddings tensor of size [B, D], and similarly we also obtain the post embeddings tensor of the same size. We then L2-normalize the embeddings and perform a matrix multiplication of the normalized embedding matrices. This will give us the logits matrix of shape [B, B]. This matrix represents user post similarity for every possible pair within a batch, with every row belonging to a user and every column belonging to a post. We treat this as a multi-class classification problem, where the number of classes is B, and ground-truth class indices lie in the main diagonal. We use multi-class cross-entropy loss to optimize our network. The main advantages of a TwoTower model over a classification model is that the number of learnable parameters in a TwoTower model is independent of the number of items/tokens in the vocabulary whereas a classification model depends on the number of number of items and in fact the final linear projection layer from the embedding dimension to the number of items in the vocabulary is often the bottleneck.

The model architecture is depicted in Figure 3. Central part of the model is a Transformer encoder with causal masking. An important change that we make to the Transformer layer is to remove a learnable token embedding layer and replace it with pre-trained token embeddings. The reason for this is that, we've a large vocabulary (order of billions) and having an embedding layer of this magnitude is not feasible. We've also removed the classification layer and replaced it with a TwoTower architecture for the same reason. We've 2 losses - optimizing for users short-term and long-term interests. With the help of the causal masking, we make the hidden representation of the Transformer match at step t match with the post embedding at step (t+1). This helps learn the users short-term interests. In order to model the user's long-term interests, we take the final hidden representation and match it with multiple posts in the future. We use the final hidden representation from the Transformer layer as user's representation. We go over the various components of the model architecture in the sections below.

3.1.1 Post/Item Embeddings. For the post embeddings, we have features for a post that include text, multimedia such as images and videos, and additional metadata such as the poster's country and detected language. We use one shared 6-layer XLM-R [4] encoder for all the textual fields. For each post there is a variable number of images attached to it. We use pre-trained image embeddings (Borisyuk et al. [2]) for each image in the post we apply a shared MLP layer and use deep sets (Zaheer et al. [28]) fusion to combined the set of image embeddings into a fixed size representation. We then fuse the representations from different feature channels with learned attention weights to get the document's final embedding representation. For video representation we used pre-trained video embeddings based on Wang et al. [24]. We train this model as a post to post similarity task using a Two Tower architecture. The architecture of the post tower is based on Liu et al. [16] and hence we'll not revisit this in detail here. The post embeddings are then fed as token embeddings to the Transformer encoder of the user tower, which we describe in the next section.

3.1.2 User Tower. For the user tower, we feed in a sequence of posts through a Transformer encoder layer with causal attention mask. While transformers usually take in a sequence of token ids and learn the token embeddings as part of the transformer layer, we feed the pre-trained post embeddings from the previous section. This is to deal with the large vocabulary of posts and the volatile nature of the relevance of posts. Along with the post embeddings, we also concatenate others feature like the time since current and treat the concatenated embedding as the embedding of the post. Additionally, we sum the post embeddings along with learned position embeddings and learned user action (like/comment/reshare) embeddings (see Fig. 4) and feed this as input to the transformer layer.

3.1.3 Short Term User Interests. In order to model the user's short term interests, we take advantage of causal masking in Transformer encoder layer. For every time step t, we take the hidden representation of the Transformer encoder layer at a time step t and match it with the post embedding at time step (t+1). We use all the posts that belong to other users in the history as negatives and optimize using a Cross-entropy loss. The reason we need a causal encoder for this is that it avoids peeking into the future. By doing this, we're able to provide all prefixes of the history as training data to the model. This also makes the model more robust as the model is now optimizing

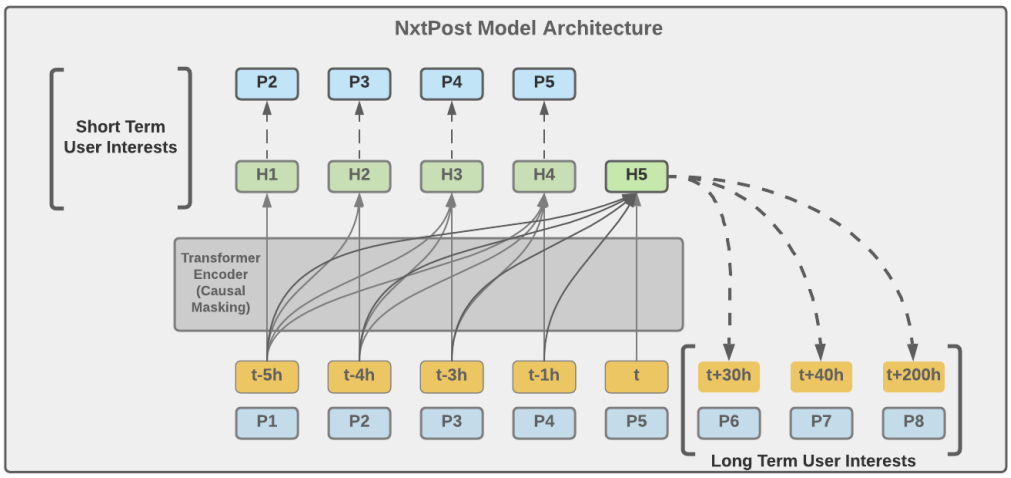

Figure 3: Architecture of *NxtPost*, optimizing for both users short-term and long-term interests.

for numerous (history, label) pairs in a single batch rather than just 1 slice of it.

3.1.4 Long Term User Interests. In order to model the user's long term interests, we take the final hidden representation from the transformer encoder with causal masking. We then match it with every post embedding at various time steps from (t+1) to (t+m) where m is the maximum number of labels we're considering. By doing this we're achieving two things - (i) we're able to model multiple user interests and (ii) we're also able to capture user's long-term interests. We use all the labels from other users as negatives and use Cross-Entropy loss to optimize the problem. The total loss is equal to the weighted combination of the short-term interest loss and the long-term interest loss.

3.2 Transformer Encoder Layer

We use a transformer encoder layer to encode the context sequence which has 4 building blocks: the embedding layer, multi-head attention, position-wise feed-forward network and a pooling layer at the end.

3.2.1 Embedding Layer. Traditionally a transformer layer has three embeddings: token embeddings, position embeddings and segment embeddings which are learned as part of the training process. From Fig. 5 we observe that last recent posts capture the most similarity with the post that would be engaged by the user in the future and hence we decided to keep the position embeddings. However, we needed to make changes to the token and segment embeddings. The token embeddings learn an embedding for every token in the language vocabulary, usually on the order of 10^5s. Since posts have a short shelf-life and new posts are constantly being created it is impractical to learn an embedding for each post as part of the model. Instead of learning an embedding for each post we instead plug-in an external content-based post encoder that is trained on content similarity—two posts will have similar embeddings if their contents are similar. While segment embeddings make sense for natural

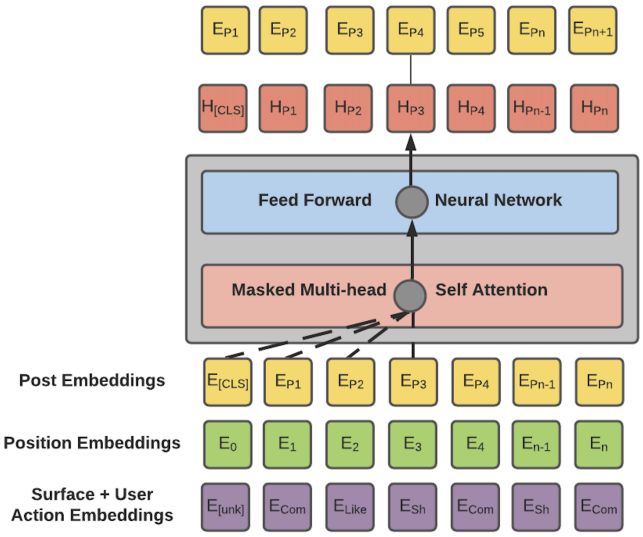

Figure 4: Causal pre-training - Given a prior set of posts, the model is required to predict the next post, which in this case is P_4. In a given batch of size B, we make upto $B * (maxsequencelength - 1)$ predictions in a batch.

language sentences (where each sentence belongs to a separate segment), we have replaced it with a combination of surface and user action embeddings. Intuitively, different user actions have a different weight (for example, commenting on a post might have a higher weight or importance than just viewing a post) and we learn an embedding for each of them. All the 3 embeddings are summed and passed to a transformer encoder layer. We've also added the [CLS] placeholder denotes the start of the sequence and its embedding is learned as part of the model.

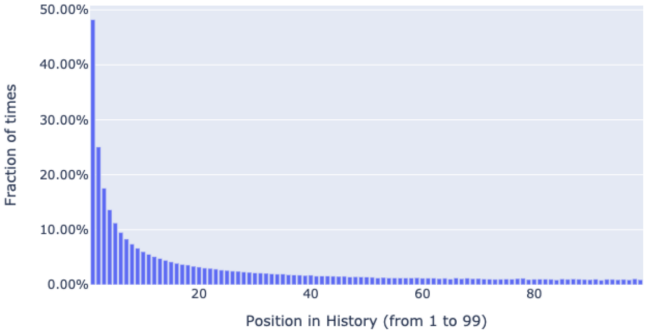

Figure 5: The sequence length data analysis. User's next engaged post in the future is mostly similar to several recently engaged posts from history.

Figure 6: Are some actions more predictive than others? In short, yes. Accuracy of predicting the next event in the sequence is much lower for comment click/like and post share than for other actions such as post like/comment/click and time spent (to a lesser extent).

3.2.2 Multi-Head Attention. At the heart of the transformer context encoder is the multi-head attention. Attention was first popularized in sequence modeling and has since been widely adopted as it can capture the dependencies between items without regard to how close/far away they are from each other. In Multi-Head attention, we project the queries, keys and values h times with different learned linear projections. On each of these projected versions of queries, keys and values, we perform attention in parallel yielding d_v dimensional output values. These are then concatenated and once again projected resulting in the final values.

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T/\sqrt{d_k})V \tag{1}$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2}$$

$$\text{MultiHead}(Q, K, V) = \text{concat}(head_1, head_2, ..., head_h)W^o \tag{3}$$

3.2.3 Position Wise Feed Forward Layer. Besides the multi-head attention, we apply a feed-forward network to each position separately and identically. This consists of 2 linear layers with a ReLU activation between them.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{4}$$

3.2.4 Pooling layer. The final layer is a pooling layer, which takes a hidden representation at every position, pools them and provides a fixed representation of the entire sequence. We have explored mean pool, sum pool and attention pooling but found mean pool to work best.

$$\begin{aligned} &\varphi = \{\varphi_i\}_{i=1}^N && \text{representations of N channels} \\ &w = \text{Softmax}(\text{proj}_N(\varphi_1 \parallel ... \parallel \varphi_N)) && w \text{ is N channel weights} \\ &f = \sum_{i=1}^{N} w_i \dot{\varphi}_i && \text{final tower representation} \end{aligned} \tag{5}$$

We used PyTorch and several downstream libraries such as Facebook's PyText, Fairseq, and the Multimodal Framework (MMF) to implement the model. For all of our experiments, we used a dropout of 0.2, gradient clipping of 1.0, 2 transformer encoder layers, sequence length of 100, learning rate of 7×10^{-4} with a batch size of 1024 and Adam optimizer.

3.3 Training

Training data. We collected user actions on the posts across pages and groups in the Facebook ecosystem as positive examples, where the data is first de-identified and aggregated before training. We filter out integrity violating posts from the training data according to defined safety procedures [1]. In order to find the correlation of user action to the probability that the user would engage with this post in the future, we analyzed many user contexts. More specifically, we split this context randomly and calculated the cosine similarity of the post embedding in the context with the target post. We then sliced it by the user action of the context post to Figure 6. We find that like, comment and post click are more correlated with the similarity of future posts than actions like comment click, comment like and comment react.

In our training setup, the dataset is positive (user, list of posts) pairs where in-batch negatives are used for negative examples. We have sampled batch negatives using techniques described in Liu et al. [16]. We've cleaned the data by taking the following measures: (1) filter-out posts with less than N interactions, (2) always consider only the history of user sequence and never look into the future. We use a scaled multi-class cross-entropy loss (Eq. 6) to optimize our network, where p_i denotes a post, and u_i denotes a user, cos denotes cosine similarity and s denotes scale. The idea of having a scale times cosine is also mentioned in Deng et al. [6]. During training, we found that having a *scale* or temperature parameter is important for loss to converge. In our use case, we choose scale between 15 and 20

$$\text{loss}_i = -\log \frac{\exp(s \cdot \cos\{u_i, p_i\})}{\sum_{j=1}^{B} \exp(s \cdot \cos\{u_i, p_j\})} \tag{6}$$

3.4 Evaluation

Before running A/B tests online on live traffic, we first evaluate our model candidates offline and select the best one based on metrics such as batch Recall@K (also called Hits@k).

Batch Hits@K: This metric measures whether diagonal element $\cos(q_i, d_i)$ is among the top K scores of the row $\cos(q_i, d_j)$, $j \in [1, B]$. This metric is easy to compute during training and is the closest metric that the model optimizes, enabling us to iterate fast on modeling ideas. For evaluation, we hold out one day of data in the future respective to when the training data was collected and use it in the target post prediction task. In prediction we only consider the history of user sequence prior to the event of engaging with the target post and never look into the future.

KNN Hits@K or Hits@K: While the batch metrics help us iterate fast, a more representative metric is where for a given user, we perform a KNN search on the entire post corpora. We then average this metric over the entire set of users. While this metric is more computationally expensive, it is more indicative of online performance.

4 ABLATION STUDIES

We performed offline ablation studies to confirm the efficiency of every step. Our baseline model [3] uses deep & wide architecture used with id features, where every item is assigned unique identifier and its embedding is trained as part of the model. We trained a transformer architecture described in § 3 with two separate sequences for pages and group posts, and static user features. It achieves +49% absolute improvement in metrics. Transformers bring substantial improvements into model quality. We were able to replace prior id based models in production. Additionally because the model is trained with pre-trained content embeddings and access to a large vocabulary, it can be applied to any new content which users enjoys over time.

We then added a CLS token - as a way to incorporate users with no prior engagement and we observed a lift in our offline metrics. We then switched over to causal mask. While this didn't give us any offline improvement, it enabled us to work on the future improvements to model a users short-term and long-term interests.

We targeted user's long-term interests by making the user embedding similar to multiple item embeddings from user's engagement future. By doing this, we target user's multiple interests and also make the model work consistently over a period of time.

With all techniques together we are able to achieve over 49% absolute improvement over the wide and deep model baseline. Biggest wins comes from Two Tower Transformer and modeling user's long-term interests. We observe that *NxtPost* system of a large vocabulary transformer based approach is able to distill dynamic nature of user history, retrieve engaging recommendations and out perform prior state of the art approaches. You can read our offline ablation study in the table 1.

4.1 Varying Sequence Length and Encoder Layers

To understand the model performance we analysed the sequence length required to improve model performance. We note that the longer the sequence length the higher to computational costs of the model. As shown in Fig. 7 beyond a sequence length of around 50-100 we begin to see diminishing returns. We have use similar sequence length for groups and page posts. We did a data analysis to understand, which sequence length would be the effective considering the user history. We have used a content understanding model for comparing a post in the history to the target post clicked in the future.

Technique	Batch Hits@1
Wide & dense baseline [3]	0.15
Two Tower Transformer (TTT)	0.44 (+29%)
Two Tower Transformer (TTT) + CLS	0.46 (+31%)
Above with Causal Mask	0.46 (+31%)
Above + Long-term	0.60 (+45%)
Above + Short-Term	0.62 (+47%)
Above + Relative Time Feature	0.64 (+49%)

Table 1: Ablation study of different model configuration given same training and evaluation data.

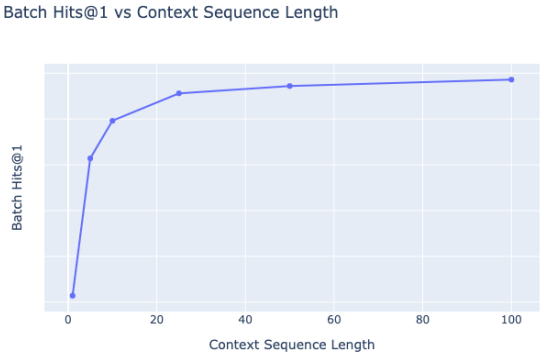

Figure 7: Changes in Hits@1 given the sequence length. Experimented with Two Tower Transformer variant from Table 1.

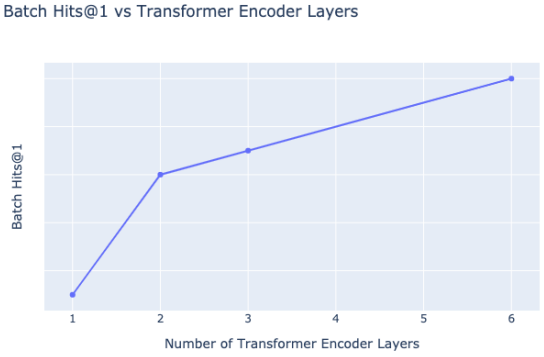

Figure 8: Changes in Hits@1 with the number of transformer encoder layers.

Technique	Batch Hits@1
NxtPost Model	baseline
NxtPost Model without engagement feature	-1.6%

Table 2: Ablation study of the relative time of engagement feature.

Technique	Batch Hits@1
NxtPost Model	baseline
NxtPost model with 500 sampled negatives	-5.3%
NxtPost model with 1000 sampled negatives	-3.1%

Table 3: Ablation study of the number of negatives for both the short-term and long-term losses.

We tuned number of layers in the encoder and choose two layers (Fig. 8), because computational load is increasing more rapidly for number of layers in comparison to improvement in Hits@1 we are getting. We tuned both batch length and number of encoder layers based on *Two Tower Transformer* variant from Table 1.

4.2 Using relative time of engagement as a feature

We observed that adding the relative time of engagement is a useful feature to capture user behavior. Some users prefer to engage with a lot of posts in a short period of time whereas some users engage with posts after more than a week. By incorporating this feature into the model, we have learnt the following lessons from our offline study - (i) the model gives less preferences to older engagement regardless of the position in the history. (ii) we observe that the model pays more attention to posts, where user signaled more long term interest such that visiting the group and consuming posts across multiple user sessions. We've summarized our offline ablation analysis in the table 2.

4.3 Varying Number of Negatives

Fetching in-batch negatives in a standard TwoTower model is a standard task - For the user i, item at index i is the positive and all the other items in the batch are negatives. However, we've 2 tasks and both have more than 1 positives and hence we describe our process of fetching negatives. For modeling short-term interests, we label it as a multi-label multi-class problem. For a given user, we could have upto N positives (where N is the maximum sequence length in the Transformer). We use all the posts from other users as the negative pool. For modeling long-term interests, we've multiple labels per user as positive. We again model this as a multi-label multi-class problem and use all the label posts from other users as negatives.

In this section, we explore an option of uniformly sampling from the pool and use only a fraction of them as negatives. From the negative pool, we uniformly sample (500, 1000) posts and use them as negatives. However, we found that using the entire pool always work best for us. We've summarized our results in Table 3.

Technique	Batch Hits@1
NxtPost Model	baseline
Sequence with popular posts	+3.2%
Personalized Model	+6.1%

Table 4: Offline metrics only for cold-start and marginal users.

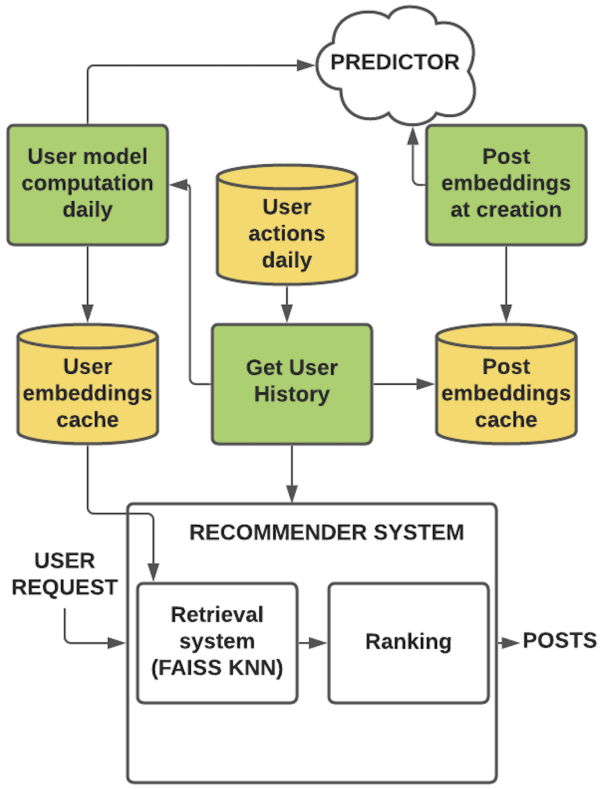

Figure 9: System architecture of *NxtPost*.

4.4 Dealing with Cold-start and Marginal Users

As *NxtPost* is a sequential model, it doesn't perform well for cold-start (users with no past engagement) and marginal (users with just a handful of past engagement) users. In order to deal with this, we took a couple of approaches - backfilling user engagement with popular posts and using a personalized model to backfill engagement from other similar users. In the first approach, we got the most popular posts and used this as a user history. In order to improve relevance, we only selected posts having the same attributes (like location, language) as the user. This helps the cold-start users. In the second approach, we built a user-user similarity model based on engagement and backfilled the history of marginal users from other similar users. This helps in improving the metrics for marginal users and we've summarized our results in Table 4. Note that this evaluation dataset only contains marginal and cold-start users and is different from the rest of the evaluation dataset.

Technique	Online results
Two Tower Transformer (TTT)	+0.18%
Two Tower Transformer + long-range	+0.5%
Two Tower Transformer + short & long-range	+0.6%

Table 5: Online A/B testing results based on model variants in Table 1 for relative improvement in number of groups users who are meeting new people, engaging with the community, sharing knowledge and getting support.

5 SYSTEM ARCHITECTURE

We illustrate the system architecture in Fig. 9. *NxtPost* is deployed in production and is designed to operate on products created in real-time. Model inferences are computed at cloud of machines called Predictor [7]. Predictor provides functionality to deploy the model, and API to call the model with a set of input features. Predictor is a cloud service and can scale accordingly.

5.1 Serving Post Embeddings

Upon the creation and update of a post, an asynchronous call will be made to Predictor to generate its post embedding, and the embedding vector will be updated to recommendations index in real-time to prepare the post for retrieval and ranking. In other words, post embeddings are already pre-computed and indexed when search queries are issued, which makes the computation of user to post similarity across many post candidates tractable. For purposes of user history inference post side embeddings are also stored to distributed file system. We access post embeddings from distributed file system when we rerun the user embeddings refresh inference job.

5.2 Serving User Embeddings

We designed re-computation of user embeddings daily for the users who have fresh engagements with a content posts in Facebook ecosystem. We collect engaged posts ids, and join it with embeddings from the embedding cache in distributed file system and provide most recent user history to User Tower model, which re-compute fresh user embeddings. We filter out integrity violating posts from the post ids list according to defined safety procedures [1]. After re-computation we refresh the user embeddings in key-value distributed memory lookup store.

5.3 Recommendation candidates retrieval

At Facebook, our systems need to handle a large QPS to serve our large user base. Queries to recommendations system are requested in real time, and given user id, we retrieve its pre-computed embedding from key-value store in milliseconds. Given user embeddings within *NxtPost* resulted posts are retrieved from recommendations index in less than several hundred milliseconds. Our recommendation retrieval engine uses [10] to compress the embedding space to speed up the k-nearest neighbor computations. Results are then passed back to several stages of ranking.

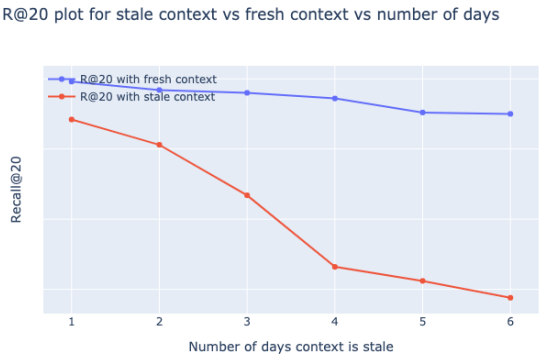

Figure 10: We find that Recall@20 drops by 3% if the history is not updated for a day, 5% if they are not updated for 2 days and up to 20% if they're not updated for 6 days.

6 DEPLOYMENT LESSONS

We deployed *NxtPost* on Facebook Groups recommendation system powering billions of queries per day. In our first iteration, we deployed the vanilla *Two Tower Transformer* (Table 1), which achieves 0.44 Hits@1 in the batch. We observed relative increase of 0.18% in number of users, who used groups to meeting new people, engaging with the community, sharing knowledge and getting support. In the next iteration, we improved the model substantially by learning user's long-term interests with a relative improvement of 0.5%. We further improved upon this to 0.6% by learning both the short-term and long-term interests. We learnt several important deployment lessons while during development of *NxtPost*.

6.1 User History Freshness

In our online experiments, we have enabled daily refresh of user embeddings by updating their context. With this setup, it naturally begs the question - Is it necessary to update the user context daily? What is the loss in metric by not doing this? We try to answer these questions analytically through both online and offline experiments. In our offline experiments, we compare Recall@20 when the context is fresh with the context being stale.

We observed that if we do not refresh the user embeddings daily then both offline (Fig. 10) and online metrics regress and go down day by day. This is intuitive because user's interests change over time and also user can not click *like* on the same post twice, so showing the fresh content based on what user interacted with recently is important to continue to entertain the user.

6.2 Model performance over time

During the training data collection process, it is important that the (user context, target post) pair are not restricted to a small time period (like a day or a week). The reason for this is that posts might be biased towards a set of particular topics and we don't want the model to be biased towards these topics. In order to test our hypothesis, we trained our model with the target post coming from a short time period and ran offline KNN evaluation over the

next 7 days. We observed that the KNN Hits@20 drops by around 30% over a period of 7 days.

We solve this problem by modeling user's long term interests. The main intuition behind modeling user's long-term interests is that the user embedding is learned to match with their interests over the next few days. After introducing the long term interests, we observed that the KNN Hits@20 drops by only 2.3% at the 7 day period, as opposed to 30% before.

6.3 Targeting and Precision

It is important for recommendation system to stays highly engaged. We measure system efficiency by click-through rate (CTR), which means that ratio of *number of user actions to number of user impressions of recommended posts* need to stay constant or improve over time of system development. To make sure *NxtPost*'s User to post embeddings improved or do not regress click-through rate we implemented filtering of returned results by a threshold. This helps to move from -6.2% in CTR regression to neutral CTR movement keeping the system more efficient by recommending more related content to users based on their interests.

7 CONCLUSION

In this paper we presented Sequence based User to Post recommendation system called *NxtPost*. We proposed to extend Transformer Based Sequence model to support pre-trained item embeddings, a large vocabulary and use two tower architecture instead of a classifier. We use a causal multi-head attention to learn both user's short-term and long-term interests. With all of the techniques explored we are able to increase model performance by over 49% absolute in comparison to baseline. We observed that as a result of *NxtPost* deployment, 0.6% more users are meeting new people, engaging with the community, sharing knowledge and getting support.

In the future iterations of the *NxtPost* we plan to explore near-real time model inference. We are also looking to explore Graph Neural Networks and bringing in other entities like groups, pages, search queries and hashtags.

8 ACKNOWLEDGEMENTS

The authors would like to thank Shaoliang Nie, Ignacio Arranz, Liang Tan, Hamed Firooz, Nikhil Garg, Qing Xu, Jun Mei, Sheng Song, Zhen Li and others who contributed, supported and collaborated with us.

REFERENCES

[1] Tom Alison. 2021. Changes to Keep Facebook Groups Safe. https://about.fb.com/news/2021/03/changes-to-keep-facebook-groups-safe/
[2] Fedor Borisyuk, Siddarth Malreddy, Jun Mei, Yiqun Liu, Xiaoyi Liu, Piyush Maheshwari, Anthony Bell, and Kaushik Rangadurai. 2021. VisRel: Media Search at Scale. In *KDD*.
[3] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & Deep Learning for Recommender Systems. In *RecSys*.
[4] Alexis Conneau, Kartikay Khandel wal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised Cross-lingual Representation Learning at Scale. In *ACL*.
[5] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *RecSys*.
[6] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. 2019. ArcFace: Additive Angular Margin Loss for Deep Face Recognition. arXiv:cs.CV/1801.07698
[7] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang. 2018. Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. In *HPCA*.
[8] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *CIKM*.
[9] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2014. On Using Very Large Target Vocabulary for Neural Machine Translation. *CoRR* (2014).
[10] J. Johnson, M. Douze, and H. Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* (2019).
[11] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. *2018 IEEE International Conference on Data Mining (ICDM)* (2018), 197–206.
[12] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[13] Dmitry Kislyuk, Yuchen Liu, David Liu, Eric Tzeng, and Yushi Jing. 2015. Human Curation and Convnets: Powering Item-to-Item Recommendations on Pinterest.
[14] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. 2019. Pytorch-BigGraph: A Large Scale Graph Embedding System. In *MLSys*.
[15] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing* (2003).
[16] Yiqun Liu, Kaushik Rangadurai, Yunzhong He, Siddarth Malreddy, Xunlong Gui, Xiaoyi Liu, and Fedor Borisyuk. 2021. Que2Search: Fast and Accurate Query and Document Understanding for Search at Facebook. In *KDD*.
[17] Xichuan Niu, Bofang Li, Chenliang Li, Rong Xiao, Haochuan Sun, Hongbo Deng, and Zhenzhong Chen. 2020. A Dual Heterogeneous Graph Attention Network to Improve Long-Tail Performance for Shop Search in E-Commerce. In *KDD*.
[18] Facebook News Room. 2019. The next step in Facebook's AI hardware infrastructure. https://about.fb.com/news/2019/04/f8-2019-day-1/.
[19] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. 2007. Restricted Boltzmann Machines for Collaborative Filtering. In *ICML*.
[20] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-Based Collaborative Filtering Recommendation Algorithms. In *WWW*.
[21] Martin Saveski and Amin Mantrach. 2014. Item Cold-Start Recommendations: Learning Local Collective Embeddings. In *RecSys*.
[22] 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 *CIKM*.
[23] Jiaxi Tang, Francois Belletti, Sagar Jain, Minmin Chen, Alex Beutel, Can Xu, and Ed H. Chi. 2019. Towards Neural Mixture Recommender for Long Range Dependent User Sequences. In *WWW*.
[24] Weiyao Wang, Du Tran, and Matt Feiszli. 2019. What Makes Training Multi-Modal Networks Hard? *CVPR* (2019).
[25] Ji Yang, Xinyang Yi, Derek Zhiyuan Cheng, Lichan Hong, Yang Li, Simon Xiaoming Wang, Taibai Xu, and Ed H. Chi. 2020. Mixed Negative Sampling for Learning Two-Tower Neural Networks in Recommendations. In *WWW*.
[26] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed Chi. 2019. Sampling-Bias-Corrected Neural Modeling for Large Corpus Item Recommendations. In *RecSys*.
[27] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *KDD*.
[28] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. 2017. Deep Sets. *CoRR* (2017).
[29] Andrew Zhai. 2021. Representation Learning for Recommender Systems. In *KDD '21 OARS workshop*.