Learnings from Building the User Intent Embedding Store towards Job Personalization at LinkedIn

Nagaraj Kota, Venkatesh Duppada, Ashvini Jindal, Mohit Wadhwa
LinkedIn AI, India
{nkota,venkd,ajindal,mwadhwa}@linkedin.com

ABSTRACT

Search and Discovery process of a job-seeker towards realizing the dream opportunity can be very complex. Given the dynamic nature of job postings, churn-rate of skills, and gaps in intent matches, professionals often find it hard to discover the right opportunity. Most often, they need guidance on the right search queries to issue or next-steps in the job-seeking funnel to reach a target job-apply.

In this work, we experiment with large-scale job-sessions dataset from LinkedIn, to understand and represent user's job-seeking behavior. In particular using *action sequences* unified from various search and discovery channels, we pre-train language models, e.g. BERT (Bidirectional Encoder Representations from Transformers) to model user's session activities. We further fine-tune the BERT based contextual session embeddings towards predicting entities from target sessions, in an eXtreme Multi-Label (XML) classification setting. We hypothesize that XML fine-tuning task enables dense-representation, and predicted entities to be used efficiently in multiple downstream tasks of job-search query recommendation, job-search ranking, job-recommendation retrieval, and job-notification expansion, as shown in experiments. We demonstrate significant improvements in accuracy across tasks leading to reduced time to reach a given job-apply, as well as increase in total job-applies in the system. To the best of our knowledge, this is the first work to efficiently model cross-channel activities data at scale using self-attention mechanisms, leading to statistically significant improvement in job-seeker experience. We also share metrics impact and learning from deploying these models in production towards job-seeker satisfaction.

KEYWORDS

Personalization, Query Recommendation, Extreme Classification, Self-Attention, BERT, Transformer, Search and Recommendation

1 INTRODUCTION

Large-scale economic graph[1] of entities, where professionals, jobs, skills, companies, etc. are interconnected, provide a rich representation of the global digital economy. Online professional social networks such as LinkedIn, aims to create an economic opportunity by making this economic graph universally explorable and actionable. Jobs marketplace is one such key vertical in the graph, where job-seekers connect to their dream opportunity through various *channels* involving search, recommendation, alerts, and navigation.

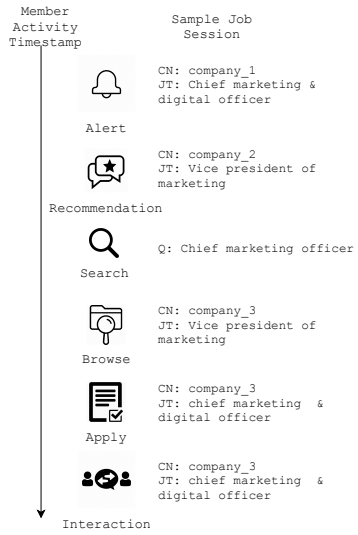

Figure 1: An illustration of a job-seeking funnel, where the user interacts with multiple channels to finally apply for a job.

We define *job-seeking funnel* of a user as collection of job sessions, each consisting of search queries and actions such as *view*, *save*, *dismiss*, or *apply* on a job-posting, which could originate from multiple heterogeneous channels. We also note that a funnel is successful when job-seeker hears back from the job-poster for the applies (considered as partial success) that were done. Figure 1 illustrates a sample job session, where the job-seeker following various search & discovery channels applied for a job title(JT): *Chief marketing & digital officer* at a company(CN): *Company_3*. Moreover, the activity funnel was successful, since job-poster viewed the candidate profile and interacted in a short window leading to an *Interaction* event. We do not consider sparse delayed feedback on 'hired' status in this work, which are typically many months away.

Potential applications of using such a large collection of job-seeking activity funnels are diverse, where one can personalize the professional connection recommendation, feed of network updates, and also recommend relevant courses for acquiring required skills - all tailored towards realizing the dream job with entities of interest. In this work we focus on personalizing search and discovery channels within the jobs-marketplace. However, our method is also applicable in domains such as e-commerce, news, and music, where understanding diverse user actions, intents in activity funnel (e.g. purchase) can lead to relevant contextual personalized

[1] https://economicgraph.linkedin.com/

KDD'21 IRS Workshop, August 14–18, 2021, Singapore

experiences. Traditional approaches to personalization include engineering through a) learning user, item specific model parameters ex. GLMix [34] b) aggregate statistics of interacted entities ex. user-jobTitle clicks, c) representation learning to yield user, and items in same semantic space, optimized for single, or multiple tasks [19]. We argue that these approaches either lack in 1) transferability of semantics across tasks, 2) model-capacity to understand finer intents in funnel. Also, these approaches are hard to extend and scale, if we were to augment user's activities from other verticals (LinkedIn feed, learning) related to their job-seeking behavior. For example, a user may be more interested in reading news about a specific company or people that she is following, which can help to realize her dream job. Also, user and item semantics can change over time. Ex. during COVID, there was a shift towards *remote, work-from-home* jobs, and many new job-postings with healthcare related skills. Users too, upon acquiring new skills, want to shift domain, or type/role of jobs.

In this work, we delve deeper into modeling user intent and interests by leveraging the actions in job-seeking funnel through semi-supervised approaches at scale. Treating user's job-seeking funnel as a document, we explore the well-known strategy of unsupervised pre-training and supervised fine-tuning for various downstream search and discovery tasks. We argue that, text understanding of activities can adapt to newer job postings (new item-ids), as the text have similar content and semantics. It can also adapt to shift in job-supply marketplace, since we model the job-seeking behavior based on subwords of the actions. While we can deeply personalize the experiences for warm-users using their rich content in the job-seeking funnel, we can infer intent vector for cold users and not signed-in users based on their initial set of interactions as well. Specifically, our contributions are as follows:

- We summarize job-seeking funnel through a low dimensional dense representation using BERT (Bidirectional Encoder Representations from Transformers) [13] pre-training. We highlight certain design choices and large-scale challenges in BERT pre-training unique to our application. We further formulate an eXtreme Multi-Label (XML) classification setting, where the contextual representations from pre-trained model are fine-tuned to predict entities, that are relevant to users' trail of activities.
- We demonstrate transferability, and utility of such fine-tuned user embeddings, and predicted entities in diverse downstream tasks: 1) guided query recommendation, 2) job-search ranking, 3) candidate selection in job-recommendation, and 4) audience expansion in job-notifications.
- We A/B test the proposed self-attention models in production through a unified architecture involving daily inference flows. We demonstrate significant increase in job-applications leading to confirmed hires (estimated at +5%), along with job-seeker satisfaction through various quality metrics. All these are enabled through personalization across diverse tasks, leveraging the core job-seeking funnel representation.

The remainder of the paper is organized as follows. Next section discusses related work. Overall framework with job-session dataset, and modeling strategies are discussed in Section 3. Search and discovery downstream tasks, and serving architecture are discussed in Section 4. Model-fitting configurations, and experiments are presented in Section 5, with insights from production deployment in Section 6. Finally, we conclude the work with future directions.

2 RELATED WORK

There have been several studies on the impact of short/long-term user's action history on Search, Query Auto-Complete (QAC) personalization. Long history, modelled as a bag of n-grams from past query sequences in a LambdaMART framework, shown to be a better predictor of intent in QAC [26], where as short history is more useful as the session evolves in personalizing the SERP [5, 30]. We differentiate the work from generic web-search query recommendation task in that we may not have historical query sequences, but have additional context available from non-search channels. Recent work [2, 3, 11] models cross-task data to exploit the sequential dependency among the individual actions to rank both documents and queries. But these works leverage only the search task sessions for downstream tasks. However, in a practical setting, both search & recommender systems co-exist, and our work exploits on sequence of actions from various channels. Our work is also related to query inference from web documents [16], where a Transformer decoder is used to generate queries using a graph-augmented sequence to attention architecture. In our work, we focus on predicting queries that would likely to lead to success in a job-seeking funnel. Hence training data characteristics and end optimization differ from the above work.

Global and local attention mechanisms proposed in [21], are also adopted in query recommendation and forms a strong baseline as noted in [3]. Hierarchical RNN architectures [10, 12, 27] are also well studied incorporating session and user level behaviors to model the short/long-term search history. [35] explores self-attention for modeling behavior interactions in multiple semantic spaces for recommendation tasks. Compared to these works, we explore architectures based-off self-attention for personalization of diverse search and discovery tasks. We focus on user intent modeling through language models that scales to hundreds of millions of users, and are engineered as context vector for diverse job-seeking tasks. We also provide details on business impact, and learning from multiple deployments in production.

In the context of online advertising, Hidden Markov models (HMMs) are explored to model purchase funnel to assign users to stages [1]. Our work is also closely related to activity funnel modeling in look-alike campaigns, where Bi-directional GRUs with local, and global attention mechanisms are explored to predict user's conversion and automatic tagging of activity funnel into different stages [36]. Sequential recommender systems literature is also related to our work, where the methods predict the next item that the user likely to take an action, exploiting *relevant* information from user's action history. Self attention mechanisms [18][20] are used effectively to predict the next best action, based on the homogeneous item sequences. Additional metadata, leveraging user micro-behavior are shown to further improve the precision of recommenders [15, 23, 28] as well. We differ from the next item prediction literature in that, our focus is to get an efficient user activity representation that can be transferred to various downstream tasks. We also work with heterogeneous action sequences

JOB_SEARCH hedge fund VIEW *c_arrow_search_partners* rotational analyst VIEW c_ijc_associates,_inc fundamental analyst JOB_SEARCH hedge fund analyst VIEW c_voya_financial analyst/associate, hedge fund investor relations JOB_SEARCH equity analyst APPLY ***c_onewire_holdings,_llc analyst, special situations at carlson capital l p**** VIEW c_the_blackstone_group gso - consultant, analyst

JOB_SEARCH consumer analyst VIEW c_niagara_partners,_inc buy side JOB_SEARCH equity research analyst APPLY ***c_softbank_investment_advisers associate - cfo office**** VIEW c_investment_firm investment analyst - hedge funds / private equity VIEW c_affirm capital markets director

Figure 2: An example activity document with two job-sessions consisting of queries, action_types, and job-postings (company, and job title). *indicates an applied job-posting

involving queries, items, item annotations, and micro-behaviors, and show the novelty of textual understanding of activities at scale.

3 ACTIVITY UNDERSTANDING FRAMEWORK

In this section, we emphasize on job-seeking funnel characteristics and how self-attention architectures are adapted to exploit the sequence contexts towards intent understanding.

3.1 Overview

We denote a user i's job-seeking funnel or activity document A_i, as a collection of multiple job-sessions, $A_i = \bigcup_j S_{i,j}$, where each session $S_{i,j}$ (jth session of user i), consists of arbitrarily ordered 1) job posting that was acted upon (with types: viewed, applied, or saved), and 2) search queries made as users engage in search & discovery experience. We treat 30-mins of idle time as a criterion for session boundary. We represent the space of user's explicit queries as $Q = \{q_1, q_2, \ldots, q_L\}$, and the interacted documents as $D = \{d_1, d_2, \ldots, d_M\}$. Each job document d_k is represented as a set of action_type, company name, and job title, denoted as $d_k = (at_k, cn_k, ttl_k)$. Hence, we represent a user i's job-session $S_{i,j}$ as

$$S_{i,j} = \bigcup_k \begin{cases} q_{i,j,k}, & \text{if } type(i,j,k) = query \\ d_{i,j,k}, & \text{otherwise} \end{cases} \quad (1)$$

Note that, one can further extend the granularity of action_types at_k to {short_view, long_view, interaction, etc.}. Such finer micro behaviors [15] have been shown to improve performance in various other recommendation settings as well. Below we define the two sequence context variants that we study in this work.

Query sequences per user i are constructed using in-session or across-sessions query histories, from the activity document A_i. This subset of the data is similar to many of the web search query recommendation tasks.

Action sequence is same as A_i and captures the intent through queries, action_types, company, and job titles as shown in Figure 2. To the best of our knowledge, we have not seen such cross-channel user trails being leveraged for diverse search and discovery tasks.

3.2 Data Characteristics

A job-seeking funnel similar to that of an e-commerce purchase funnel, is a sequence of job-related activities that are triggered from various search & discovery channels. We construct a 1.4B job-sessions dataset, from these trail of activities over 13-months. Figure 3 provides a distribution of sessions in number (left) and share of action_types (right) at various session lengths. In order to capture the impact of search on job-applies within session, we slice the metrics by `srch_apply_srchApply` to denote if a session

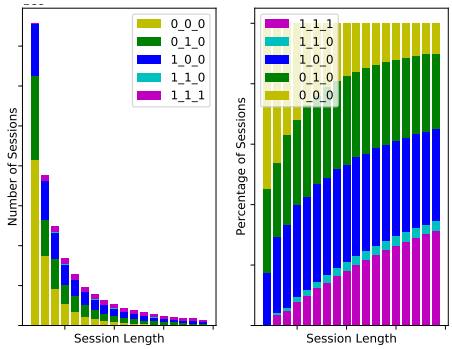

Figure 3: Distribution of job-sessions against session length, sliced by if a session has 1) any search event, 2) any job-apply, 3) job-apply post a search event. e.g. 0_1_0 indicate sessions with a job-apply action, but no search queries.

1) has any search event 2) has any job-apply 3) has a job-apply post a search event. We observe a significant share of short sessions with length<=2 and no search queries (0_*_*), that are mainly driven by the transactional push job-notifications, and job-recommender systems. However, job-applies from the segment (0_1_0) indicates satisfied seeker intent, which otherwise are missed out, if had modelled user intent using only the search-engagement channel. From the right subplot in Figure 3, one can observe that at higher session lengths (>10), >50% of sessions have at least one search action (1_*_*) in them. Also, job-apply post search activities (1_1_1), among the search sessions are between 40-50%, much higher (+2x) compared to the lower session-length (passive) region. Another observation is that of increase in share of search and job-applies post search activities, as the session length increases. This is mainly driven by the active job-seekers who use search channel often and contribute to significant number of job applications in the system. This is also observed in online advertising [36], where unaware users transition to high intent / purchase through various intermediate stages of aware -> interest -> consideration. Above observations show that there is a huge value in exploiting the context across channels.

3.3 Unsupervised Pre-training

Treating job-seeking funnel as a *document* of variable length text, with each job-session analogous to a sentence, we employ *Bidirectional Encoder Representations from Transformers (BERT)* [13] pre-training to learn the contexual representations. We experiment with a corpus of 1.4B job-sessions that consists of 70B word tokens, which is significantly higher than BookCorpus(800M), and Wikipedia(2.5B). Following the original BERT work, we use Byte Pair Encoding (BPE) [25] encoding mechanism and build a subword vocabulary of size 100K. We note that certain action types like VIEW are very common in the session, and there are many repetitive user actions across sessions, ex. company entities. Hence, along with *full word masking*, we do *selective masking* to not mask frequent action-types. We keep the same masking (random 15%) strategy as BERT with a dupe factor of 3 and optimize Masked Language Model (MLM) objective only, as we observed that Next Sentence Prediction (NSP) task is relatively simpler in our case. Note that MLM objective aims to predict entities at random in a session, enforcing BERT to accurately *predict user trajectories*. Further design

choices w.r.t training, architecture, masking mechanisms, and pooling are discussed in Section 5.1, as the job activity semantics are very different when compared to natural language sentences from BookCorpus/Wikipedia.

3.4 Supervised Fine-tuning to Predict User Intents at Scale

We consider a variety of downstream search and discovery tasks as described in Section 4, which requires an aggregated representation of user's job-sessions. From an engineering design perspective, it is desired to incorporate the aggregated representation without further fine-tuning. Also, we do not want to maintain channel specific, or task-specific versions of user embedding in production. We also note that the usual supervised ranking task's data that maximizes P(action | user, jobPosting, context) are not adequate here. It *breaks the sequence*, and are also not aligned to predict the various intents in the job-seeking funnel. Hence we predict *entities* of interest from the held-out 20% job-sessions, in an eXtreme Multi-Label (XML) setting[17]. Predicted entities or labels are explicit actions from users, for e.g. search queries, or applied (company, job-titles). For fine-tuning, we consider recent 1-chunk (512 sub-words) from user's job-session activities as input to the BERT pre-trained model to yield [CLS] token (and other pooling variations') embedding, which we treat as user representation or user-activity vector. We add a *sigmoid* classification layer on top of the derived embedding to rank the entities of interest per user activity document. This enables BERT model parameters to be updated end-to-end during backpropagation. Above fine-tuning strategy enables multiple usecases at scale and leads to significant improvement over the pre-trained model, as shown in experiments.

4 SEARCH AND DISCOVERY PERSONALIZATION USECASES

In this section, we provide details on diverse downstream tasks, and how activity representations are leveraged in personalizing user experiences.

4.1 Guided Query Recommendation

We consider personalizing job query suggestions in *cards* that are shown on LinkedIn jobs home page [2] as 'suggested job searches', or as soon as the user clicks the search box 'try searching for'. '*Suggested job searches*' card is targeted towards users to drive more searches, converting passive intent to active interests. These suggestions are also aimed at bridging the gap in discovery process, that many job-seekers did not know how to issue the relevant queries to reach the right job-postings. 'Try searching for' card serves dual purposes, where we personalize the suggestions based on past search queries issued and/or using much richer activity funnel beyond search.

Task definition: Given a user activity document A_i, *predict a subset* τ *of queries in* Q that relevant to the sequence of activities. Note that, we extend the task definition in [3] to include cases, where query history may not be present, and the trail can be arbitrarily long with interactions from multiple channels. In fact, $\sim$ 16% of the job-seekers never issued a search query, but instead engaged in other recommendation channels.

[2]https://www.linkedin.com/jobs/

eXtreme Multi-Label Classification, Slice: Given a user activity document representation, we want to predict the queries that happened in the past or in a held-out sessions data, or both. In a typical job-seeking system, potential number of queries to be predicted can easily run over $> 10M$. Hence we formulate the *query recommendation* task as an eXtreme Multi-Label (XML) classification problem. There exists a rich body of literature on extreme classification[3], and we leverage state-of-the-art linear classifier, Slice [17], that can scale to $\approx$ 100 million labels with inference time under logarithmic in number of labels. We denote L, N, D as number of labels, training samples, and feature dimension respectively. Slice uses Hierarchical Navigable Small World Graph (HNSW) [22] for sampling $O(\frac{N}{L} logL)$ hard negatives per label. Along with the explicit positives, we train L 1-vs-all classifiers using the user-activity vectors obtained through representation learning, in parallel. We note that overall complexity for training L classifiers is $O(NDlogL)$. Prediction time is $O(DlogL)$ due to an efficient label candidate selection using Approximate Nearest Neighbor Search (ANNS) over HNSW index[22].

We also study BERT fine-tuning in an XML setting that are trained end to end. Compared to the Slice training, where feature extraction and learning linear classifiers are de-coupled, fine-tuning enables BERT model parameters to be updated end-to-end during backpropagation as described in Section 3.4.

4.2 Learning to Rank in Job Search

Search channel contributes to majority of the job applications in the system, primarily driven by the active job-seekers. They issue queries related to entities of interest (title, skill, company etc.), and engage with job postings towards their dream job. Passive job-seekers also engage with search through their broader intent queries. It is critical to personalize the Search Engine Results Page (SERP) that are relevant, not just to the issued queries, but also to user's profile, onsite behavior, and context information ex. location, recency. Learning to rank approaches are widely used in the industry for SERP re-ranking, in that items with higher relevance are ranked higher, maximizing certain utilities. We leverage TF-ranking library [24] based out of Tensorflow for solving this large-scale personalized ranking problem. Production baseline is a deep and wide neural architecture with query, jobPostings represented through Transformer layers [29] in the *deep* part, and many {query, user, document, context} features along with its *sparse* interactions are incorporated in the wide part. We extend the deep part of the architecture with fine-tuned user-activity vector in a separate column (frozen), followed by fully-connected layers for late interaction with other features. We note that only the fully-connected layer weights are learnt during the back-propogation and activity-vector remains the same across usecases. While fine-tuned activity vector captures semantics across channels, they are further adapted to optimize Search objectives.

4.3 Audience Expansion in Job Notifications

Push notifications through various channels e.g. App, E-mail are critical in retaining users, who care for freshness of the items recommended. Job notifications system, where users are notified as soon a relevant job is posted, enables immediate reach of relevant

[3]http://manikvarma.org/downloads/XC/XMLRepository.html

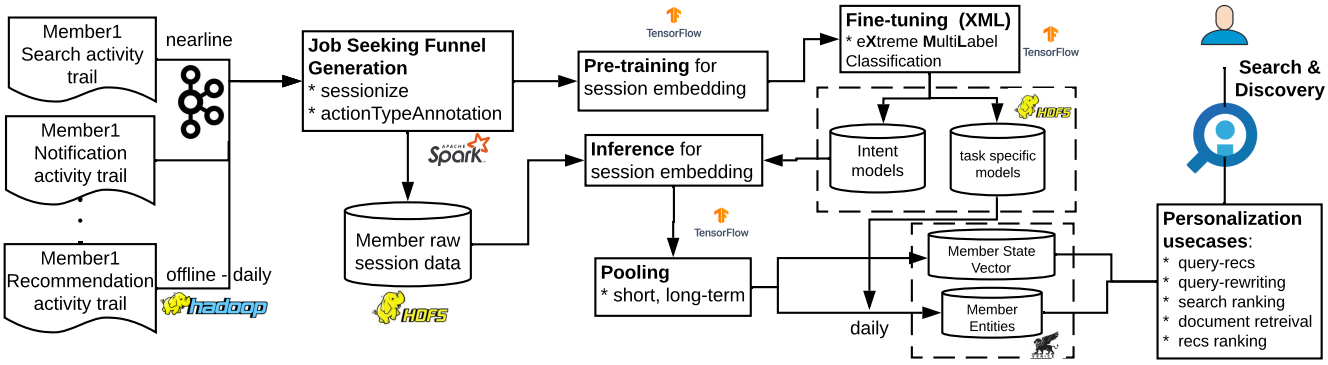

Figure 4: High-level unified architecture for Search and Discovery Personalization

candidates (users) for job-posters. Similar to many recommenders, this system recommends suitable candidate(s) for a posted job in a two-step process. First the relevant job-seekers are *retrieved* based on job-postings' (query-side) metadata, entities ex. skills, title, location. Matched candidates are then re-ranked by learning to rank methods optimizing for certain utilities. We note that, entity-based exact matches are precise, but are limited in recall, leading to low liquidity in certain segment of jobPostings. We visualized a random sample of user's activity vector from BERT fine-tuned model. For a given user with title:*'Senior Counsel IP & Licensing'*, top-ranked neighboring user's title were *{(principal partner), (partner, patent lawyer), (deputy general counsel - business, finance), (senior corporate counsel)}*. This shows an evidence of many of the contexts including job-seniority, job-function are implicitly captured in a low-dim representation of a job-activity funnel and we were able to accurately capture similar users in the job-seeking funnel. Hence, we **expand** the top-K exact matches (anchor-set) from the entity based retrieval to include candidates that are similar in their job-seeking journey. To that end, we index the user-activity vectors using HNSW [22] method for efficient search of nearest neighbors. We denote this system as Scalable Approximate Nearest Neighbor Search (SANNS). HNSW index is built offline and served/queried online with REST APIs. Specific choices in indexing the vectors, and querying mechanisms are described in the experiments.

4.4 Candidate Selection in Job Recommenders

Traditional candidate selection models leverage top-K user attributes, or entities to form boolean clauses to retrieve relevant job documents. These entities mainly come from user's profile text, and onsite behavior. Compared to the embedding based retrieval, this strategy is explainable, and one can tweak the clauses for desired precision, and recall. We extend the retrieval boolean clauses with predicted entities from the *guided query recommendation* task, as these are personalized based on user's action sequences as described in Section 4.1. We also leverage SANNS method in an offline setup, based on applied sequences observed per job posting. Using the initial-K applied members for a given job, we leverage HNSW index and offline query mechanism to find similar job-seekers. These expanded members per job are further re-ranked based on similarity scores and are attributed to the job that was considered in context. We then aggregate job attributes (like titles, skills) for a given member across all the expanded jobs, and extend/re-rank the retrieval boolean clauses. Both these schemes introduce personalized clauses in candidate selection, derived from user-activity vectors.

4.5 Unified Serving Architecture

Figure 4 provides a high level architecture for personalizing Search and Discovery tasks using the activity understanding framework. First, we merge user actions from different channels {search, alerts, recommenders } and create sessions with boundary idle timeout of 30mins between any two actions. We annotate actions with their types e.g. VIEW, APPLY, JOB_SEARCH as shown in Figure 2. Though these actions are near real-time streamed through Kafka[4] events, we ETL with hourly cadence for all the actions to unify. Job-sessions per user are persisted on HDFS, with BERT pre-training, and fine-tuning under XML setting, were done using Tensorflow on LinkedIn's GPU cluster. Fine-tuned intent models are then used during the *daily* inference for activity vector extraction. Task-specific models (such as Slice) are leveraged to predict entities of interest using user-activity vectors. We persist user-activity vector and entities in Venice[5] key-value store. We observe that discriminative models capture long-term intent across sessions and an addition of query or an action does not impact the suggestions much. Hence, currently we deployed it in a daily cadence, although we are looking to update the dataflows/inference to Samza for near realtime capability. Except for the usecase in Section 4.3, all others fetch the required artifacts: activity vector, or bag of entities for a given user, through point lookup off Venice store. Also, online search & discovery usecases are abstracted from mode of update in the user store, which could be nearline, or hourly, or daily.

5 EXPERIMENTS

In this section, we evaluate activity document representation models with its pre-training, and fine-tuning objectives. We also share the insights from incorporating the user-activity vectors in various personalization usecases. Key metrics that we report from the online user A/B tests include first-order click through rates (**CTR**), second-order **job-applications**, and the end goal of increased **confirmed-hires**. Note that, measurement of actual confirmed-hires would require running A/B tests with much larger lookback window due to delayed feedbacks. Also, the ground-truth data about job-poster reaching out to job-seeker, interviewing, and getting-hired are sparsely available through various applicant tracking systems (ATS).

[4]https://kafka.apache.org/
[5]https://engineering.linkedin.com/blog/2017/02/building-venice-with-apache-helix

Table 1: LinkedIn job-sessions dataset statistics

	#docs	avg. sessions per doc	total #tokens	unique #tokens
Pre-training	85M	16.47	70B	4M
	#examples	#labels	avg. labels per point	avg. points per label
Fine-tuning	train:16M, test:2M	500K	16.16	790.0

Hence, we model confirmed-hires using a predictive model that accurately estimates the probability of a job-application resulting in confirmed hire. We denote this metric as predicted confirmed-hires (**PCH**). Due to sensitivity and business reasons, we will not be able to provide further details on the features, and algorithmic details of the PCH models.

Towards *reproducibility* of the main findings in this work, and its relevance to broader recommendation challenges with new-items, we also conduct experiments on publicly available Microsoft News Recommendation dataset, **MIND**[32]. This challenge involves textual understanding of user activities at scale, in-order to rank documents that rarely occurs in the training data (12.5%). We detail pre-training, supervised fine-tuning task, learning to rank setting, and offline experiment results in Appendix A. Due to privacy, legal, and business restrictions, we will not be able to release the LinkedIn job-sessions dataset or the code.

5.1 Job-seeking Funnel Representation Learning

For pre-training the BERT language models from scratch, we collect a sample of job-seeking activities on LinkedIn from Jan-2019 to Jan-2020. Table 1 provides statistics of this large-scale dataset, that consists of 1.4B job-sessions from ~85M users.

Unsupervised Pre-training: As noted in Section 3.3, masked job-sessions are given as input to BERT pre-training to optimize Masked Language Model (MLM) loss. We input sessions in sequence similar to that of original BERT work, in that they are seperated by [SEP], and maximum sequence length (or max_positional_embedding) across two-segments (i.e. two-sessions) are 512 subword tokens. We experiment with various BERT configurations as noted in Table 2. With asynchronous parameter server distribution-strategy, we optimize the model weights using TensorFlow on in-house TonY[6] cluster using 30 V100 GPU workers. We set #parameter-servers:4, batch_size:32, and optimize using AdamWeightDecayOptimizer[13] with learning_rate:5e-4. With total_training_steps:50M, which is 2 epochs of the duped (3-factor) masked job-sessions corpus, we set warmup_steps to 20% of the train_steps per worker. Based on the trade-off in validation loss, training time, and serving latency, we chose 12-layer with 128 hidden-units and 2 attention-heads as the final BERT configuration as marked in bold in Table 2.

For aggregated representation of job-sessions per user (i.e. user-activity vector), we consider recent 5-chunks (i.e. 5*512 subword tokens) from the activity document. Each chunk is given as input to extract various contextual representations. We experimented with 1) [CLS], and 2) mean-token (REDUCE_MEAN) embeddings from a) last layer, b) concatenation of last-4 layers. We found that mean-token embeddings from **last-4** layers outperform all the variants in finetuning task (described below) with precision@k +2x, over the commonly used last layer's [CLS] variant. We then mean-pool the recent-5 chunks' embedding to get user-activity vector. This simple, yet practical scheme enables us to summarize very long-range contexts when using BERT models. We are also exploring Longformer[4] to directly model the long-range context as well.

[6]https://github.com/linkedin/TonY

Table 2: BERT pre-training configs. Highlighted row was optimal w.r.t. training time, serving latency, and validation loss.

#Layers	Hidden Size	#Attention Heads	#Training Params	Training Time (days)
4	256	4	30M	5
6	512	8	70M	12
8	128	2	14.5M	2
12	**128**	**2**	**15.4M**	2.5
12	256	4	35M	5.5

To quantify the gains from BERT based contextual embeddings, we also train global word vectors using **fastText**[6] to obtain session (sentence) embeddings that serves as another feature extraction baseline. We use the same BERT pre-training corpus with a vocab of 1.3M size consisting of *top* company phrases, and other unigrams. We learn the word representations using a $skipgram$ model with $softmax$ loss, predicting the target word using a context window of 5. Various configurations with word vector dimensions $(100, 300, 512)$ and learning rate $(0.05, 0.08, 0.1)$ are tried with 5 epochs. Of these, best performing variation with lowest test-loss was $(dim = 300, lr = 0.08)$. We derive session (sentence) embedding and mean-pool over all sessions to yield a 300d user activity vector.

Supervised fine-tuning, datasets/evaluation: We evaluate user-activity vectors in predicting entities of interests under XML framework, as mentioned in Section 3.4. We split the sequence of sessions from a user activity document into 80% context, and 20% target. We construct two datasets: **D1**, where entities (search queries, and applied job-titles, if any) from the 20% target are made to predict. **D2**: where entities from 100% of sessions are made to predict. D2 statistics are provided in Table 1. Note that we use the term label, entity, query interchangingly. Also across the two datasets, user-activity vector as context, is derived from 80% of the sessions only. D2 has other practical benefits with prediction of top-K ranked explicit entities that one can set an alert on or for targeting of sponsored campaigns. In our offline experiments, best method among D1 and D2 remains the same, although precision@k are worser by ≈ 40% in D1, which is relatively a difficult task.

We report *Precision@k* by ranking the predicted entities that follows the actual action sequences. However, in many cases, predicted entity (e.g. solutions architect) and actual ground-truth (e.g. technical consultant) are related, which leads to no-match in precision@k definition. Hence we also define a new metric, Semantic-Similarity@k to capture relatedness, as

$$\frac{1}{N}\sum_{i}\frac{1}{k}\sum_{j=1}^{k}\max_{e_i}(cosine(\hat{e}_{ij}, e_i)) \tag{2}$$

where for user i, $\hat{e}_{ij}$ denotes entity embedding of j'th predicted entity by model and e_i are set of all ground truth entity embeddings of user i. N is the total users in the evaluation set. We note that the entity embeddings are obtained as phrase-vectors through fastText[6], trained on a query sequences dataset. We do not impose ordering constraints among the *target* entities, as they are heterogeneous in its downstream utility. E.g. query that lead to many discovery

Table 3: Comparison of discriminative query suggestion models

Variant	Precision			Semantic Similarity			SC
	@1	@3	@5	@1	@3	@5	
CNN+e2e	0.525	0.411	0.348	0.73	0.68	0.65	0.097
fastText+e2e	0.575	0.442	0.368	0.76	0.70	0.67	0.158
BERT+e2e	**0.80**	**0.655**	**0.546**	**0.86**	**0.81**	**0.77**	0.122
fastText+Slice	0.418	0.27	0.21	0.73	0.69	0.67	0.699
BERT_PT+Slice	0.403	0.27	0.207	0.73	0.68	0.66	0.760
BERT_FT+Slice	0.696	0.50	0.384	**0.84**	**0.78**	**0.75**	**0.766**

sessions, vs job apply vs abandonment. Business may care for one or multiple of these goals requiring multi-objective optimization[9] formulations, which is beyond the scope of this work. Hence we do not report the usual ranking metrics such as Normalized Discounted Cumulative Gain (NDCG), Mean Reciprocal Rank (MRR) for query recommendation task.

Baseline models, trained end to end: We further experiment with models that are optimized end to end to predict relevant entities, using the global word vectors (fastText) as input representation. We consider 1) a neural network with 2-layers of fully-connected layers of 64-hidden units each with batch normalization, and relu activation, 2) CNN model with recent 2048 word tokens of the activity document as input. Word tokens are initialized with 300d fastText vectors learnt from the job-sessions corpus. We use 128 filters, each with [2,3,4,5] word n-grams followed by max-pooling and a fully connected layer with 512-hidden units. We denote these baselines as **fastText+e2e**, and **CNN+e2e** respectively. **CARS**[3], **LostNet**[11] as a baseline did not scale to our dataset D2, but we provide a note on results from **ACG** [12], and **MNSRF**[2]. Since the supervised fine-tuning task on dataset D2, is similar to *guided query recommendation*, we report the overall results in Section 5.2

5.2 Personalized Query Recommendation

User-activity vector from BERT pre-trained model, denoted as BERT_PT, is obtained by the concatenation of 128d vectors (mean token-embeddings) from last-4 layers to yield a 512-dimensional vector. A Slice model is trained using 16-million 512d BERT_PT activity vectors to predict relevant labels from **500K** set that were derived from top occurring entities in job-search logs. We denote this modeling strategy as **Bert_PT+Slice**. The parameters used to train the slice model are M:100, efC:300, efS:300, num_nbrs:300, classifier_threshold:1e-6. For more details on these parameters, we refer the reader to [17]. Similarly, we train another Slice model on 300d fastText activity vector to yield **fastText+Slice**. We also study fine-tuning BERT_PT in an eXtreme Multi-Label (XML) classification setting. We add a classification layer on top of the derived user-activity vector from BERT_PT and fine-tune the network in end-to-end manner along with BERT parameters. We denote this method as **BERT+e2e** (refers to BERT fine-tuning end to end). To measure the standalone impact of Slice, we also trained another Slice model using the 512d user-activity vector with concatenation of mean embedding of last-4 layers extracted from fine-tuned BERT (BERT_FT). We denote this method as **BERT_FT+Slice**.

Offline evaluation results on dataset **D2** are reported in Table 3. **Suggestion Coverage (SC)** denotes the number of unique predicted labels (among the top-100 predictions), across entire user base normalized by label space of 500K queries. This metric captures the diversity among the predicted entities for exploration. Low SC indicates that the model is biased towards predicting certain labels, which are highly-frequent (head) in the logs. We make several observations: 1) Among various feature extractors for user activity vector, BERT_FT performs the best compared to global fastText, CNN and BERT_PT, showing the utility of fine-tuning. 2) Among SC, Slice outperforms end-to-end training due to its candidate label-lists, that are generated by the nearest neighbors search on HNSW index of label embeddings. End to end training, including BERT+e2e are biased towards predicting head labels. One reason could be that the logits in the output layer are not well caliberated, which is a hard problem in XML setting. 3) Although precision@K seem lower due to its binary/indicator relevance, semantic similarity@k reveals that many of the predicted labels are very relevant and semantically related to the ground truth. This gap is wider in Slice variants. 4) Considering SC and semantic-similarity (which has a direct impact on ratio of exploratory suggestions vs repeating past queries on the UI), we chose BERT_FT+Slice as the overall winner.

Table 4: Suggestions for the activity document in Figure 2

Model	Top-5 query suggestions
BERT_FT + Slice	equity portfolio manager, quantamental, equity research associate, long/short equity, tmt analyst

Table 5: Editorial evaluation of top-5 query suggestions, with empty search query history

	Perfect	Good	Fair	Bad
fastText+Slice	0.375	0.191	0.23	0.21
BERT_FT+Slice	0.64	0.16	0.11	**0.09**

With no search history: We also construct another dataset **D3** of users, who did not issue any search query, but interacted through other channels. Since there isn't any ground-truth data, we evaluate query recommendation quality through editorial guidelines. Table 5 reports the evaluation results on dataset D3. In this study, editors are asked to grade if a query suggestion is semantically relevant to the user-trail of activities. With no calibration, or thresholding of Slice's one vs all classifier scores, we were able to achieve lowest fraction of bad-suggestions with Bert_FT+Slice (resulting in a deployable candidate).

Semantic Query Recommendation: For the activity document in Figure 2, top-5 suggested queries from BERT_FT+Slice are presented in Table 4. We highlight the semantic relevance of recommendations with certain query words being not present in either of query or action sequences. For ex. *quantamental, long/short equity, tmt analyst* from BERT_FT+Slice were not present in activity document, but are inferred based on pooled activity vector. Multi-task model **MNSRF**[2] had P@1=0.31, and higher 'unk' predictions. We attribute this to its neural architecture, which was designed to model homogeneous search action-sequences only. Upon inspection of few examples, we found that non-search activities were equally discriminative in predicting the target queries, hence lowering MNSRF's precision. We also experimented with generative models **ACG**[12] which had P@1=0.567 on dataset D2. These were trained using the repo from [3] with word vectors initialized from job-session pre-trained fastText embeddings. However, in 99.3% of the cases, the generated query was a past searched one, lowering its utility since there exists another card on 'Recent Searches'. This repeat querying behavior over a long horizon of job-sessions are possibly due to users clicking on past searches (available in search box) or through alerts, which the generative models cannot robustly

model. However, BERT_FT+Slice efficiently extracts the long term interests through much richer action sequence representation and scalable linear classifiers for precise exploration.

Online A/B test: We employ daily batch inference flows to compute personalized suggestions from BERT_FT+Slice, served through Venice as described in Section 4.5. We deployed BERT_FT+Slice discriminative model in both 'Suggested job searches', and 'Try searching for' cards to majority traffic as of May 14 2020. From the two weeks of A/B tests before the full-ramp, over a sequential phrase-vector (trained on member's job-title transition) baseline, we observed significant first-order impact with total clicks on these cards up by +5.7%, with CTR: +8.56% leading to +1.35% job-applies, and +0.82% predicted confirmed hires.

5.3 Learning to Rank in Job Search

Experiment configuration: As described in Section 4.2, production baseline is a deep and wide architecture, where the deep part is a Transformer [29] (1-layer, 100-hidden, 10 attention-heads) encoder, which embeds search query, job title, and job company separately, followed by 4 fully-connected (fc) layers. We initialize with 100d glove word-embeddings (in-house), and apply batch normalization in all fc layers Wide part consists of multiple categorical (ex. job industry, geo-location), and continuous features (ex. overlap of query and job title) and contribute to 100+ diverse features representing sparse interactions of {query, user, job-document, context}. For nonlinearity, we use ReLU and tanh activation in deep and wide part respectively. At the end, features from wide and deep part are concatenated to optimize for job-engagement.

We extend existing deep and wide architecture, where we append user vector derived from BERT_FT in it's own tower in deep part of the architecture followed by 5-fc layers with hidden units of 512, 256, 256, 128, 128. Following other components in deep part, we also apply batch normalization and ReLU activation. Our aim is to show the utility of user activity vector (frozen), and task specific adaption without having to further fine-tune them. However, one can directly fine-tune the BERT_PT model within this architecture to learn task specific semantics leading to improved performance at the cost of increased training time and higher serving latency. We train the network with batch_size:64, train_steps: 360k, learning_rate:1e-1 using Adagrad optimizer with cross entropy loss [8]. During training, we primarily track NDCG@25 metric and save the model checkpoint based on validation set performance. The model is trained using TF-Ranking framework with 20 CPU workers and takes ~ 4 hours for training.

Results: In offline evaluation, with BERT_FT, we observed +1.21%, and +2.5% improvement in NDCG@25, and AUC-ROC over the production baseline model. These were significant given that there exists multiple activity related features, embeddings in the production model with years of feature engineering. We did not observe significant changes in metrics with number of fc-layers, and hidden-units variations in the activity vector tower. In online A/B tests, at 20% ramp over 4-weeks, we observed +1.95% CTR@K (organic job-postings), +1.46% job-applications, and +1.5% PCH lifts.

5.4 Audience Expansion in Job Notifications

Experiment configuration: To scale to hundreds of million users, we randomly partitioned the user activity vector corpus into 16 shards, and built HNSW indexes for parallel querying for nearest neighbors. HNSW parameters are set to M:48, efC:32, efS:1000. We used 300d fasttext based user activity vector, in order to tradeoff memory, and online latency over the BERT_FT vectors. Offline index takes around 6hrs to build and are refreshed every three days. During the online querying phase, we considered following query-vector construction mechanisms. 1) mean representation of the seed-set activity vectors, 2) pseudo-relevance feedback, where we independently retrieve top-K nearest neighbors for each of the seed-set users. We then use the mean-representation of the seed-set + expanded users. This is done to reduce the noise among the initial seed-set. The query-vector is then used to retrieve 500 nearest neighbors with 0.80 similarity score threshold, which are merged with the exact-match candidates, and are re-ranked using learning to rank models. Users satisfying certain relevance threshold for the jobPosting, are further considered for the push notification.

Table 6: Retrieval evaluation with predictive entities, over the production baseline

Candidate selection models	Offline Metrics	
	AUC	F1
(fasttext+Slice) + SANNS	−1.59%	−1.70%
Baseline + (fasttext+Slice) + SANNS	+4.01%	+3.88%
Baseline + (BERT_FT+Slice) + SANNS	+3.85%	+3.41%

Results: We evaluated *offline* recall of SANNS system, based on the applied job titles overlap between query member and retrieved similar members over the past 2 months data. Among the query-vector mechanisms, we found that mean-representation worked better with top-5 relevant members from the anchor set. This was measured using an internal ML model that is optimized for quality of matches between user, and jobs. We also found cosine-distance as a winning variant among similarity measures. In *online A/B test*, from the 50% ramp of mean-representation variant over 4-weeks, we observed increased notifications liquidity with relevant (unique) recommendations. Over the entity-based exact matches, we observed +79% unique recommended candidates, +127% job-applies with +123% PCH in notifications, resulting in overall marketplace wide lift of +1.45% PCH, and +0.43% unique applies.

5.5 Job Recommendation Candidate Selection

Experiment setup: Retrieval query consisting of many boolean clauses are re-ranked [7] to trade-off latency, system cost, and precision/recall. Baseline production model uses attributes from user profile, explicit onsite actions etc. to retrieve relevant job-postings for recommendation. It also re-ranks the clauses using a light-weight model that optimizes for P(apply|user,job) to yield learned weights per clause. We expand the boolean clauses with user attributes derived from guided-search recommendation and offline SANNS as mentioned in Section 4.4. e.g. predicted entity: 'tmt analyst' from Table 4 is rewritten as titleId: 335 AND skillId: 986 clause to capture jobs with 'analyst' titles, and skills with 'tmt' within the investment banking industry. We experiment with three variants: 1) replace baseline production model with entities from (fasttext+slice) + SANNS; extend baseline with clauses from 2) (fasttext+slice) + SANNS, and 3) (BERT_FT+slice) + SANNS.

Results: Offline evaluation with AUC, and F1 (precision/recall) are presented in Table 6. We observe that 1) predicted entities clauses (alone) are marginally worse than production baseline,

Table 7: Online A/B test results, and business impact

Search and Discovery usecase	Job-applies	PCH
Personalized Query Recommendation	+1.35%	+0.82%
Learning to Rank in Job Search	+1.46%	+1.5%
Audience Expansion in Job Notifications	+0.43%	+1.45%
Candidate Selection in Job Recs	+1.34%	+0.93%

which had more sources of attributes, and feature engineering, 2) Adding predicted entities over the baseline, significantly improves retrieval accuracy, 3) fastText variant performed slightly better than BERT_FT, due to the clause weights learnt were global and not user-specific. We A/B tested fasttext+Slice, and BERT_FT+Slice models with SANNS, extending the production baseline to observe marginally better metrics for (fasttext+Slice)+SANNS. At 20% ramp, these predictive entities lead to +1.34% job-applications, and +0.93% PCH lifts. We observed increased recall, and higher job-applies from passive job-seekers, who were exploratory in nature.

6 LEARNINGS FROM DEPLOYMENT

From an engineering perspective, we standardized the representation of user activities through language models for various downstream tasks. This enabled versioned development with changes in algorithm, and update mode (daily, hourly, or near real-time) of activity vectors, abstracted away from the downstream search and discovery tasks. Hence, we could iterate much faster with multiple paths to production, and impact. Diverse downstream usecases and business impact from a single artifact of user-activity vector are a departure from the traditional way of feature engineering through aggregate statistics, and pooling over the past interacted items embeddings[14]. We summarize the business impact in Table 7. We plan to experiment these activity vectors in domain-adaptation setting in non-job verticals e.g. LinkedIn Learning, where we want to study how job-seeking behavior can personalize what courses to watch, or skills to acquire.

Guiding users with relevant search suggestions in their job-seeking journey had a ripple effect. We observed an uptick of +0.5%, in users subscribing to these personalized suggestions as alerts. These explicit alerts trigger relevant job-notifications, increasing liquidity in the job-marketplace and also further personalization of job-recommendations. With personalized clauses in the retrieval, we observed *quality* of matches in job-recommendation, went up by +3.1%, measured through explicit user feedbacks. We hypothesize that, predicted entities based on long-term (5-chunks of activity document) interest modeling nudge specificity in the passive job-seeker segment, resulting in relevant job-matches. Frozen user-activity vector as global context, demonstrated ease of integration in learning to rank task, and its adaption to SERP engagement objectives. With neutral dismisses of notifications, liquidity improvements through expansion of seed-set of users leveraging job-seeker similarity, was one of the impactful experiments within job-notification system. We observed that one can fine-tune the aggregation window to finer interval of say recent-K activities (instead of K-chunks) to derive in-session activity vector for real-time personalization. Towards that end, we are working on near real-time data availability, and inference throughput.

We note that our learnings are applicable in other recommender systems as well, e.g. news, music, and e-commerce. These domains also include multiple user interactions, e.g. click, view, search, add_to_cart, purchase, review_comment, return_product, tracked across search & discovery channels. Methods presented here provide a simple, yet effective intent representation at scale for deeper personalization. Our XML fine-tuning task that optimizes joint-ranking of entities (proxy for document), and queries, can be extended to multiple user-tasks involving predicting churn, life-time value, abuse as well.

7 CONCLUSION

In this work, we proposed self-attention based language models to learn user intents and interests in the job-seeking funnel. Under eXtreme Multi-Label (XML) classification setting, we fine-tuned the language model to predict entities that are relevant to user's trail of activities. We also showed that the fine-tuning task transfers well across multiple search and discovery tasks. Through experiments, we demonstrated user activity vector derived from the fine-tuned BERT model, works well for active job-seekers as shown in job-search ranking, notifications. For passive job-seekers, through guided query-recommendation, and job-recommendation, we showed increased job-applies, and satisfaction as there were higher subscription to queries as alerts, and thumbs-up feedback on job-matches. We plan to explore graph neural networks to model sparse sessions, and extend the language models in heterogeneous setting, where activities include actions beyond the jobs-marketplace. We plan to explore linear-attention mechanisms [4, 33], that have been shown to efficiently work with very long sequences. Lastly, we plan to exploit the rich structure within activity sequences through time, and relationship between micro-behaviors.

REFERENCES

[1] Abhishek, V., Fader, P., and Hosanagar, K. Media exposure through the funnel: A model of multi-stage attribution. *In SSRN Electronic Journal* (2012).

[2] Ahmad, W. U., Chang, K., and Wang, H. Multi-task learning for document ranking and query suggestion. In *Proceedings of the 6th ICLR (2018)*.

[3] Ahmad, W. U., Chang, K.-W., and Wang, H. Context attentive document ranking and query suggestion. In *Proceedings of the 42nd SIGIR* (2019), ACM, pp. 385–394.

[4] Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150* (2020).

[5] Bennett, P. N., White, R. W., Chu, W., Dumais, S. T., Bailey, P., Borisyuk, F., and Cui, X. Modeling the impact of short- and long-term behavior on search personalization. In *Proceedings of the 35th SIGIR* (2012), ACM, pp. 185–194.

[6] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. Enriching word vectors with subword information. *Transactions of the ACL 5* (2017), 135–146.

[7] Borisyuk, F., Kenthapadi, K., Stein, D., and Zhao, B. Casmos: A framework for learning candidate selection models over structured queries and documents. In *Proceedings of the 22nd SIGKDD* (2016), ACM, p. 441–450.

[8] Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., and Li, H. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th ICML* (2007), pp. 129–136.

[9] Carmel, D., Haramaty, E., Lazerson, A., and Lewin-Eytan, L. Multi-objective ranking optimization for product search using stochastic label aggregation. In *Proceedings of The Web Conference 2020*, ACM, pp. 373–383.

[10] Chen, W., Cai, F., Chen, H., and de Rijke, M. Attention-based hierarchical neural query suggestion. In *The Proceedings of 41st SIGIR* (2018), ACM, pp. 1093–1096.

[11] Cheng, Q., Ren, Z., Lin, Y., Ren, P., Chen, Z., Liu, X., and de Rijke, M. Long short-term session search with joint document reranking and next query prediction. In *The Web Conference* (2021), ACM.

[12] Dehghani, M., Rothe, S., Alfonseca, E., and Fleury, P. Learning to attend, copy, and generate for session-based query suggestion. In *The Proceedings of the CIKM* (2017), ACM, pp. 1747–1756.

[13] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the NAACL* (2019), pp. 4171–4186.

[14] Grbovic, M., and Cheng, H. Real-time personalization using embeddings for search ranking at airbnb. In *Proceedings of the 24th SIGKDD* (2018), ACM, p. 311–320.

[15] Gu, Y., Ding, Z., Wang, S., and Yin, D. Hierarchical user profiling for e-commerce recommender systems. In *Proceedings of the 13th WSDM* (2020), ACM, p. 223–231.

Table 8: MIND dataset statistics

	#docs	avg. tokens per sentence	total #tokens	unique #tokens
Pre-training	877K	47.0	870M	78K

[16] Han, F., Niu, D., Lai, K., Guo, W., He, Y., and Xu, Y. Inferring search queries from web documents via a graph-augmented sequence to attention network. In *Proceedings of WWW* (2019), ACM, pp. 2792–2798.

[17] Jain, H., Balasubramanian, V., Chunduri, B., and Varma, M. Slice: Scalable linear extreme classifiers trained on 100 million labels for related searches. In *Proceedings of the 12th WSDM* (2019), ACM, pp. 528–536.

[18] Kang, W.-C., and McAuley, J. Self-attentive sequential recommendation. In *Proceedings of the ICDM* (2018), IEEE, pp. 197–206.

[19] Kenter, T., Borisov, A., Van Gysel, C., Dehghani, M., de Rijke, M., and Mitra, B. Neural networks for information retrieval. In *Proceedings of the 11th WSDM* (2018), ACM, pp. 779–780.

[20] Li, J., Wang, Y., and McAuley, J. Time interval aware self-attention for sequential recommendation. In *Proceedings of the 13th WSDM* (2020), ACM, p. 322–330.

[21] Luong, T., Pham, H., and Manning, C. D. Effective approaches to attention-based neural machine translation. In *Proceedings of EMNLP*, ACL, pp. 1412–1421.

[22] Malkov, Y. A., and Yashunin, D. A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *CoRR abs/1603.09320* (2016).

[23] Meng, W., Yang, D., and Xiao, Y. Incorporating user micro-behaviors and item knowledge into multi-task learning for session-based recommendation. In *Proceedings of the 43rd SIGIR* (2020), ACM, p. 1091–1100.

[24] Pasumarthi, R. K., Bruch, S., Wang, X., Li, C., Bendersky, M., Najork, M., Pfeifer, J., Golbandi, N., Anil, R., and Wolf, S. Tf-ranking: Scalable tensorflow library for learning-to-rank. In *SIGKDD* (2019), ACM, p. 2970–2978.

[25] Sennrich, R., Haddow, B., and Birch, A. Neural machine translation of rare words with subword units. In *Proceedings of the 54th ACL* (2016), pp. 1715–1725.

[26] Shokouhi, M. Learning to personalize query auto-completion. SIGIR '13, ACM, pp. 103–112.

[27] Sordoni, A., Bengio, Y., Vahabi, H., Lioma, C., Grue Simonsen, J., and Nie, J.-Y. A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In *Proceedings of the 24th CIKM* (2015), ACM, pp. 553–562.

[28] Tanjim, M. M., Su, C., Benjamin, E., Hu, D., Hong, L., and McAuley, J. Attentive sequential models of latent intent for next item recommendation. In *Proceedings of The WWW* (2020), ACM, p. 2528–2534.

[29] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In *Advances in NIPS*. 2017, pp. 5998–6008.

[30] White, R. W., Bennett, P. N., and Dumais, S. T. Predicting short-term interests using activity-based search context. In *Proceedings of the 19th CIKM* (2010), ACM, pp. 1009–1018.

[31] Wu, C., Wu, F., Ge, S., Qi, T., Huang, Y., and Xie, X. Neural news recommendation with multi-head self-attention. In *Proceedings of the 9th EMNLP-IJCNLP* (2019), ACL, pp. 6389–6394.

[32] Wu, F., Qiao, Y., Chen, J.-H., Wu, C., Qi, T., Lian, J., Liu, D., Xie, X., Gao, J., Wu, W., and Zhou, M. MIND: A large-scale dataset for news recommendation. In *Proceedings of the 58th ACL* (2020), pp. 3597–3606.

[33] Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., and Ahmed, A. Big bird: Transformers for longer sequences. In *Advances in NIPS* (2020), pp. 17283–17297.

[34] Zhang, X., Zhou, Y., Ma, Y., Chen, B.-C., Zhang, L., and Agarwal, D. Glmix: Generalized linear mixed models for large-scale response prediction. In *Proceedings of the 22nd SIGKDD* (2016), ACM, p. 363–372.

[35] Zhou, C., Bai, J., Song, J., Liu, X., Zhao, Z., Chen, X., and Gao, J. Atrank: An attention-based user behavior modeling framework for recommendation. In *AAAI* (2017).

[36] Zhou, Y., Mishra, S., Gligorijevic, J., Bhatia, T., and Bhamidipati, N. Understanding consumer journey using attention based recurrent neural networks. In *Proceedings of the 25th SIGKDD* (2019), ACM, pp. 3102–3111.

A REPRODUCIBILITY: MICROSOFT NEWS RECOMMENDATION CHALLENGE

Microsoft released a large-scale dataset for news recommendation research [32], obtained through the user-activity logs of Microsoft News. The dataset consists of click-behavior of 1 million users on more than 160k english news articles. Task is to re-rank the candidate news items for given user, at a given time to maximize clicks. There exists 4-weeks of click-history for a given user, and system needs to predict 5th week (test-data) click-behaviors. Given that 87.5% of the docs are new in the test-set, quality of news content understanding and user interest modeling are crucial to overall performance.

Table 9: Pointwise learning to rank baselines on MIND dataset

Feature config	AUC	MRR	nDCG@5	nDCG@10
{u*v}	0.671	0.322	0.355	0.417
{u,v,u*v}	0.681	0.330	0.365	0.426
{u,v,|u-v|}	0.689	0.333	0.369	0.431

We replicate the pre-training, fine-tuning, and learning to rank strategy, that was demonstrated on LinkedIn job-sessions dataset, here. We treat user as a document, with sequence of clicked news items as sequence of sentences. We concatenate tokens from {category, subCategory, Title, Abstract} for a given news item to yield 1 sentence. Corpus statistics are provided in Table 8. We experiment with subword vocab of 30k. For pre-training the BERT models, we follow similar masking strategy as job-sessions dataset, except that there are no other action_types. We set dupe_factor to 7 to yield a total of 24M tf-records of format: [CLS] newsId1_sentence [SEP] newsId2_sentence. We experiment with 12-layer, 256-Hidden units, and 4-attention heads BERT config. We optimize for both MLM+NSP loss over 20 epochs with batch_size:32, using 100 V100 GPU workers. We observed convergence in total loss, with MLM_loss = 1.069, and NSP_loss=0.507. Overall training took 12 hours.

To replicate finetuning in XML setting, we consider wikiIds, and category_subCategory as labels. In total, there were 40.5K labels from train+dev sets. We considered 5 weeks of click_histories per user, and form <input,output> pairs by predicting labels from future clicked-items (output). We consider a minimum of 4, and maximum of 10 clicked news items in context as input. We do a stride of 3 to sample many such pairs for a given user. Total records were 32M, out of which we randomly sampled 10K members' instances as dev-set. We follow similar strategy of pooling: concatenation of mean token-embeddings from last 4-layers as user-activity vector. In this case, we obtain 4*256=1024d vector, which are optimized for cross-entropy loss.

Similar to the job-search ranking task in Section 4.2, we experiment with tf-ranking [24] for news recommendation. We construct training example with 1-positive, and random 10-negatives (from impression candidates) to form a list_size:11. Hence, total records for training is same as the total clicked items (non-unique) in the provided training set. With fine-tuned BERT model we encode u:user_context, v:news_item to yield 2 1024d vectors. As simple baselines, we took hadamard product u*v as feature concatenation layer, along with other variations: {u,v,|u-v|}, {u,v,u*v}. This is followed by 4-fully-connected layers with hidden-units of [512,256,256,256]. Architecture is optimized using pointwise loss for 300K steps with batch_size:32. We observed that the variation {u,v,|u-v|} had **+1.7%, +2.7%** lifts in AUC, and nDCG@5 respectively, over the best single-model NRMS[31] reported. Table 9 provides a summary of evaluation with other feature combinations. These results with their extensions and finetuned BERT model, TF-ranking model checkpoints can be accessed [7] for further experimentation.

[7] https://drive.google.com/drive/folders/1ZkXghUk_zMkcfyo-Qkv4k0fPeCh32wOt?usp=sharing