# DeText: A Deep Text Ranking Framework with BERT

Weiwei Guo, Xiaowei Liu, Sida Wang, Huiji Gao, Ananth Sankar, Zimeng Yang, Qi Guo, Liang Zhang, Bo Long, Bee-Chung Chen and Deepak Agarwal
LinkedIn, Mountain View, California
{wguo,xwli,sidwang,hgao,ansankar,zyang,qguo,lizhang,blong,bchen,dagarwal}@linkedin.com



## ABSTRACT

Ranking is the most important component in a search system. Most search systems deal with large amounts of natural language data, hence an effective ranking system requires a deep understanding of text semantics. Recently, deep learning based natural language processing (deep NLP) models have generated promising results on ranking systems. BERT is one of the most successful models that learn contextual embedding, which has been applied to capture complex query-document relations for search ranking. However, this is generally done by exhaustively interacting each query word with each document word, which is inefficient for online serving in search product systems. In this paper, we investigate how to build an efficient BERT-based ranking model for industry use cases. The solution is further extended to a general ranking framework, DeText, that is open sourced and can be applied to various ranking productions. Offline and online experiments of DeText on three real-world search systems present significant improvement over state-of-the-art approaches.

## KEYWORDS

Ranking, Deep Language Models, Natural Language Processing

## 1 INTRODUCTION

Search systems provide relevant documents to users who are looking for specific information through queries. A user receives a list of ranked documents ordered by search relevance, where ranking plays a crucial role to model such relevance that directly affects consequential user interactions and experience. Most search systems deal with a large amount of natural language data from queries, profiles, and documents. An effective search system requires a deep understanding of the context and semantics behind natural language data to power ranking relevance.

Traditional ranking approaches largely rely on word/phrase exact matching features, which has a limited ability to capture contextual and deep semantic information. In the recent decade, deep learning based natural language processing technologies present an unprecedented opportunity to understand the deep semantics of natural language data through embedding representation [13]. Moreover, to enhance contextual modeling, contextual embedding such as BERT [8] has been proposed and extensively evaluated on various NLP tasks with significant improvements over existing techniques.

However, promoting the power of BERT in ranking is a non-trivial task. The current effective approaches [6, 18, 22] integrate BERT as an embedding generation component in the ranking model, with the input a concatenated string of query and document texts. BERT is then fine tuned with ranking loss. The inherent transformer layer [27] in BERT allows direct context sharing between query words and document words, exploiting the power of contextual modeling in BERT to the greatest extent, as the query word embeddings can incorporate many matching signals in documents. This approach, in the category of interaction based models [7, 11, 28], comes with a significant challenge in online serving: a) the heavy BERT computation on the fly is not affordable in a real world search system; and b) the interaction based structure, as applied to concatenated query and document, precludes any embedding pre-computing that can reduce computation.

To enable an efficient BERT-based ranking model for industry use cases, we propose to use representation based structure [13, 26]. Instead of applying BERT to a concatenated string of query and document texts, it generates query and document embeddings independently. It then computes the matching signals based on the query and document embeddings. This approach makes it feasible for pre-computing document embedding; thus, the online system only needs to do BERT real-time computation for queries. By independently computing query and document embeddings, however, we may lose the enhancement on the direct context sharing between queries and documents at word-level [22]. This trade-off makes it a challenge to develop a BERT-based ranking model that is both effective and efficient.

In this work, we investigated the BERT-based ranking model solution with representation-based structure, and conducted comprehensive offline and online experiments on real-world search products. Furthermore, we extended the model solution into a general ranking framework, DeText (Deep Text Ranking Framework), that is able to support several state-of-the-art deep NLP components in addition to BERT. The framework comes with great flexibility to adapt to various industry use cases. For example, BERT can be applied for ranking components that have rich natural language paraphrasing; CNN can be applied when ease of deployment is a top concern for a specific system.

Beyond the ranking framework, we also summarized experience on developing an **effective and efficient** ranking solution with deep NLP technology, and how to balance effectiveness and efficiency for industry usage in general. We shared practical lessons of improving relevance performance while maintaining a low latency, as well as general guidance in deploying deep ranking models into search production.

The contribution of this paper is summarized below:

- We developed a representation based ranking solution powered by BERT and successfully launched it to LinkedIn's commercial search engines.
- We extended the ranking solution into a general ranking framework, DeText, that can be applied to different search

products with great flexibility. The code is open sourced for public usage.[1]
- We provided practical solutions and lessons on developing and deploying neural ranker models with deep NLP w.r.t. balance between efficiency and effectiveness.

## 2 RELATED WORK

In this section, we first introduce how Deep NLP models extract text embeddings, discuss their application in ranking, and then introduce the status of ranking model productionization.

### 2.1 Deep NLP based Ranking Models

There are two categories of deep NLP based ranking models: representation based and interaction based models. *Representation based* models learn independent embeddings for the query and the document. DSSM [13] averages the word embeddings as the query/document embeddings. Following this work, CLSM/LSTM-RNN [20, 26] encodes word order information using CNN[16]/LSTM[12], respectively. All these three works assume that there is only one field on the document side, and the document score is the cosine similarity score of the query/document embedding. NRM-F [30] adds more fields in the document side and achieves better performance. One major weakness of representation based networks is the failure to capture local lexical matching, since the text embedding, *e.g.*, a 128 dimensional vector, cannot summarize all the information in the original text.

To overcome the issue, *interaction based models* compare each part of the query with each part of the document. In DRMM [11], a cosine similarity is computed for each word embedding in the query and each word embedding in the document. The final document score is computed based on the pairwise word similarity score histogram. K-NRM [28] and Conv-KNRM [7] extended DRMM by kernel pooling and pairwise ngram similarity, respectively. Recently, BERT [8] has shown superior performance [6, 18, 22] in ranking. It is considered an interaction based model, since the query string and document string are concatenated as one sentence, where transformer layer [27] compares every word pair in that sentence.

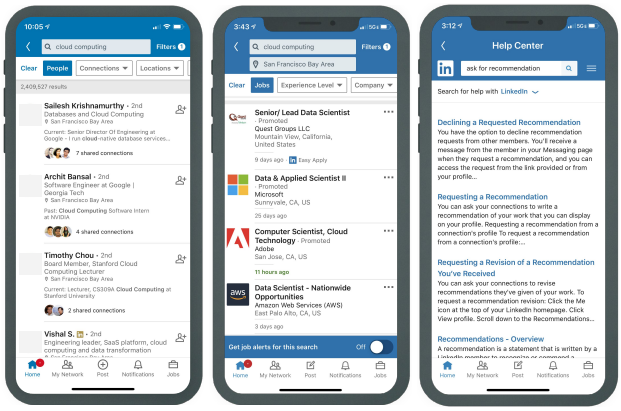
In experiments of previous works, interaction based methods usually produce better relevance results than representation based methods, at the cost of longer computation time introduced by the pairwise word comparison.

### 2.2 Productionizing Deep Neural Ranker

Commercial search engines have a strict requirement on the serving latency. Despite better relevance performance, the interaction based ranking approaches are not scalable due to the heavy interaction computation. Therefore, to our best knowledge, the representation based approaches are generally used for production.

With representation based approaches, existing work uses embedding pre-computing, either for documents [23, 29] or for member profiles (personalization) [10]. It requires a huge amount of hard disk space to store the embedding, as well as a sophisticated system design to refresh the embeddings when there are any document/profile changes.

[1] www.github.com/linkedin/detext




(a) People Search (b) Job Search (c) Help Center

**Figure 1: The first two figures show the search result of "cloud computing" in people search/job search, respectively. The last figure shows an example of query "ask for recommendation" in help center search.**

**Table 1: Summary of three vertical searches.**

| | People | Job | Help Center |
|---|---|---|---|
| **No. of Unique Docs** | 600M | 20M | 2,700 |

## 3 SEARCH SYSTEMS AT LINKEDIN

There are many search ranking systems at LinkedIn. Figure 1 shows three examples: people search that retrieves member profile documents; job search that ranks job post documents; and help center search that returns FAQ documents. The number of unique documents in each search system is listed in Table 1. In general, the common part of these ranking systems is to discover the relevant documents, based on many hand-crafted features. Similar to other vertical searches such as Yelp or IMDB, the documents at LinkedIn are semi-structured with multiple fields. For example, member profiles contain headline, job title, company, etc. In general, the retrieval and ranking process needs to be finished around one or several hundred milliseconds.

The data from these three search verticals are different in nature. The queries and documents in help center search are the most similar to natural language, *i.e.*, the text data is more likely to be a normal sentence with proper syntax, and majority queries are paraphrases of the problem that users want to address in help center search. People search is on the other end of the spectrum: the queries and documents are mostly entities without grammar; exact keywords matching such as company names is important. Job search data lies in between.

## 4 DETEXT FRAMEWORK FOR BERT-BASED RANKING MODEL

In this section, we propose a BERT-based ranking framework using representation-based structure. The framework can be extended to support other neural network components, such as CNN and LSTM, for deep natural language processing. Specifically, we refer to the BERT-based ranking model as DeText-BERT, and directly

illustrate the model using the open sourced DeText framework as shown in Figure 2.

We design the DeText framework to be (1) general and flexible enough to cover most use cases of ranking modules in search systems; (2) able to reach a good balance between efficiency and effectiveness for practical use cases.

## 4.1 Architecture

As illustrated in Figure 2, given multiple source (queries, user profiles) / target (documents) texts and traditional features, the DeText framework contains 6 components: Token Embedding Layer, Text Embedding Layer, Interaction Layer, Traditional Feature Processing, Multilayer-Perceptron Layer, and Learning-to-rank Layer. Specifically, DeText-BERT model uses BERT as the text embedding layer. In the rest of this section, we will illustrate the details of each component.

**Input Text Data**. The input text data is generalized as *source* and *target* texts. The source texts could be queries or user profiles. The target text could be documents. Both source and target could have multiple fields, which is different from most previous work [13, 20, 26], where only two fields (query and document) are available. There are several advantages of using multiple fields: 1). enable personalization with text fields from user profiles, and 2). achieve better and more robust results.

**Token Embedding Layer**. The sequence of text tokens is transformed into an embedding matrix $E$. For text with $m$ tokens, the matrix has a size of $d \times m$, where $d$ is the number of token embedding dimensions. Depending on the text encoding methods, different token granularities are used: in CNN/LSTM, the tokens are words; in BERT, the tokens are subwords [24].

**Text Embedding Layer**. Under the representation based model structure, embedding is extracted independently for each text field. The embedding can be generated through various neural network components for deep natural language processing, such as BERT, CNN, LSTM, etc. The outcome of this layer is a $d$-dimensional embedding vector. More details are discussed in Section 4.3 and 5.2.

**Interaction Layer**. The interaction between source and target only happens after the text embedding is generated, which is the key difference of representation based methods from interaction based methods. Table 2 summarizes the different interaction methods, where $u_s/u_t$ is the source/target field embedding, respectively. Note that for every source and target pair, cosine similarity generates one feature, while the Hadamard product/concatenation generates many features (a vector).

**Table 2: Interaction features.**

| | | |
|---|---|---|
| Cosine similarity | $\frac{u_s^\top u_t}{\|u_s\| \cdot \|u_t\|}$ | one feature per source/target pair |
| Hadamard product | $u_q \cdot u_d$ | $d$ features per source/target pair |
| Concatenation | $u_q \oplus u_d$ | $d$ features per text field |

**Traditional Feature Processing**. The existing hand-crafted features, such as personalization features, social networks features, user behavior features, are usually informative for ranking. To integrate them with deep NLP features, we use standard normalization and elementwise rescaling [1] to better process the features:

$$x_i^{(1)} = \frac{x_i - \mu}{\sigma}$$
$$x_i^{(2)} = w x_i^{(1)} + b$$

where mean $\mu$ and standard deviation $\sigma$ are pre-computed from training data, and $w$ and $b$ are learned in the DeText-BERT model.

**MLP Layer**. Deep features, as the output of the interaction layer, are concatenated with the traditional features as the final features, followed by a Multilayer-Perceptron (MLP) [19] layer to compute the final document score. The hidden layer in MLP is able to extract the non-linear correlations of deep features and traditional features.

**LTR Layer**. The last layer is the learning-to-rank layer that takes multiple target scores as input. DeText provides the flexibility of pointwise, pairwise or listwise LTR [3], as well as Lambda rank [4]. Binary classification loss (pointwise learning-to-rank) can be used for systems where click probability is important to model, while pairwise/listwise LTR can be used for systems where only relative position matters.

## 4.2 Flexibility of DeText

The DeText framework enables model **flexibility** to adapt to demands of different productions, in terms of input data layer (multiple source/target fields), text embedding layer (CNN vs BERT), interaction layer (cosine/hadamard/concat), LTR (pointwise/pairwise/listwise), etc.

By enhancing the model flexibility, we can optimize the model **effectiveness** while maintaining **efficiency**. Firstly, representation based methods are used to bound the time complexity. Secondly, the flexibility of input data/interaction layer, together with traditional feature handling, enable us to experiment and develop scalable neural network models with strong relevance performance.

## 4.3 DeText-BERT for Ranking

To use BERT in ranking model, we follow the approach of fine-tuning on pretrained BERT model [8]: The BERT model is firstly pretrained on unsupervised data, and then fine-tuned in ranking framework with supervised clickthrough data. To extract the text embedding, we use the embedding of a special token "[CLS]". The source/target embedding will later go through the interaction layer to generate deep features.

Previous work [22] shows that directly training a representation based BERT ranking model does not yield good results. This is because the BERT fine-tuning requires a small learning rate (around 1e-5). Therefore, two optimizers are used in DeText, each with a different learning rate responsible for a different part of the model. For example, in our experiments (Section 6.1.4), we set 1e-5 for BERT components, and 1e-3 for other components. Using this dual learning rates strategy, a successful representation based ranking model with BERT can be trained.

In order to reduce the online serving latency and capture domain-specific semantics, we also pretrained a compact BERT model on LinkedIn's in-domain data, named as LiBERT. More detailed can be found in Section 6.1.5.

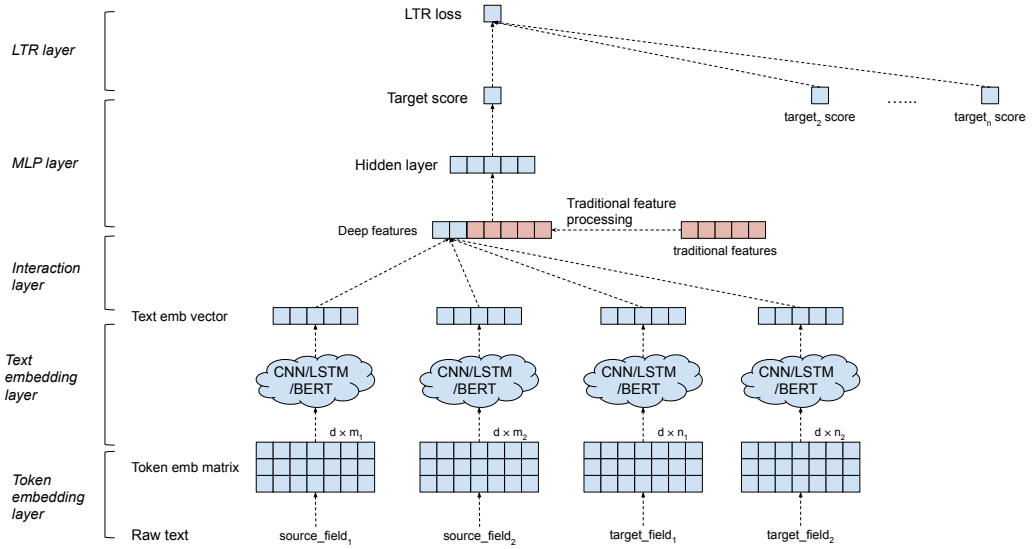



**Figure 2: The DeText framework. In this figure, there are two source fields, and two target fields.**

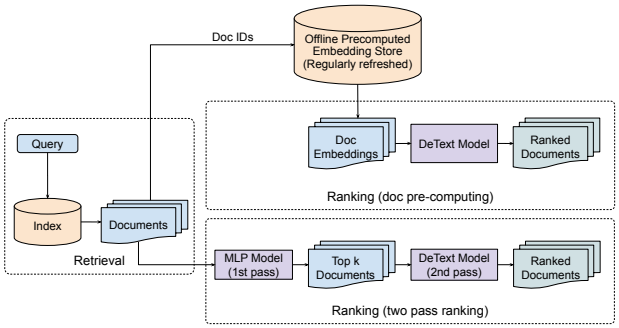



**Figure 3: Document embedding pre-computing and two-pass ranking.**

### 4.4 Support CNN for Ranking in DeText

DeText framework can support CNN for deep natural language processing in the text embedding layer. It is worth noting that we use word tokens instead of triletters as in prior work [13, 26], since the latter lifts the computation (by an order of the character length of words). We follow the previous work [14] to generate the text embedding from word embedding matrix $E$. Specifically, it uses a one-dimensional CNN along the sentence length dimension. After max-pooling, the resulting text embedding vector has $f$ elements, where $f$ is the number of filters.

## 5 ONLINE DEPLOYMENT STRATEGY

The major challenge of deploying deep neural models comes from serving latency. As shown in Figure 3, two different deployment strategies, document pre-computing and two pass ranking, are designed for BERT based models and CNN based models, respectively. They are discussed in detail in the following subsections. Note that the online deployment strategies only affect ranking components; the document retrieval components stay the same.

### 5.1 DeText-BERT with Document Embedding Pre-computing

The multiple transformer layers in BERT is computationally time-consuming. Since DeText uses the representation based method, we are able to adopt the document embedding pre-computing approach for search ranking, as shown in the boxed section (doc pre-computing) in Figure 3. For offline, document embeddings are pre-computed with BERT and saved in an embedding store, which is a fast key-value store where key is the document id, and value is the pre-computed document embedding vectors. The store is refreshed on a regular basis (e.g., daily). For online, after document candidates are retrieved from search index, the corresponding document ids are used to fetch the document embeddings from pre-computed embedding store. The benefit of this approach is to have the heavy BERT online computation only happen on the queries. It can significantly save online computation time, since the document texts are much larger than the query texts. In the setting of 10 documents for one query, the online latency can be reduced from hundreds of milliseconds to tens of milliseconds with this deployment strategy.

### 5.2 DeText-CNN with Real-time Inference

DeText-CNN can be adopted with a different online integration strategy: real-time inference for both sources and targets. Compared to document embedding pre-computing, real-time inference can simplify online system design without the need of pre-computing or refreshing document embeddings. Hence the real-time inference could be a lightweight solution readily applied for many search engines. In this paper, we show that real-time inference can be achieved by (1) choosing a compact DeText structure without hurting relevance performance too much, and (2) two pass ranking that reduces 99 percentile (P99) latency.

We find that a compact CNN structure with small dimensions can perform well in our experiments (Table 7). This is mainly because traditional handcrafted features from the production systems

**Table 3: Online metrics definitions.**

| Metric | Definition |
|---|---|
| CTR@5 | Proportion of searches that received a click at top 5 items |
| Session Success Rate | Proportion of search sessions that received a click. A new session is created if the user has no activity in help center search for 30 minutes. |
| Job Apply | Job search metric. Number of job applications from search. |
| Happy Path Rate | Help center search metric. Proportion of users who searched and clicked a document without using help center search again in that day, nor creating a ticket. |

already contain valuable information for ranking, so that the CNN model can be focusing on the signals that are missing in the traditional features.

Even with a simple network, the CNN computation time grows linearly with the number of retrieved documents. Therefore, we use a two pass ranking schema to bound the latency (Figure 3, two pass ranking box). The first ranker is a MLP [19] with one hidden layer without the deep features, which is fast. After ranking, only the top ranked hundreds of documents are sent to the DeText-CNN model.[2] This two pass ranking framework has several benefits: 1). easy to implement and deploy; 2). the MLP ranker can filter out a large amount of irrelevant documents, which provides a relatively small candidate set with high recall for CNN ranker; and 3). the latency is bounded since CNN is applied to a small set of candidates.

## 6 EXPERIMENTS

In this section, we discuss the offline and online experiments of DeText on search ranking tasks in English traffic.

### 6.1 Experiment Setting

*6.1.1 Datasets.* The models are tested on three document ranking datasets, *i.e.*, people search, job search, and help center search. The training data is collected from clickthrough data, which are sampled from 2 month traffic: 5 million queries for people search, 1.5 million queries for job search, 340 thousand queries for help center. Both development and test set have 50 thousand queries for each vertical from the later month. One query usually has 10 or more documents. Multiple document fields are used as the target fields of DeText: 1). In people search, the documents are the member profiles; three profile fields are used: headline, current position, past position. 2). In job search, the job post title, company name are used. 3). In help center search, document title and example question (illustrates the typical question for the document) are used.

*6.1.2 Metrics.* For both offline/online metrics, only relative metric improvement over baseline models instead of absolute values are presented, due to the company confidential policy. The online metrics are defined in Table 3.

*6.1.3 Baseline Models.* The production models are trained with XGBoost [5]. The hyper-parameters (pairwise vs listwise, number of trees, etc) are optimized by both manual tuning and auto hyper-parameter tuning, which are proven effective in LinkedIn's commercial search engines.

[2]Note that the top hundreds of documents are in one worker, while the online ranking is distributed to many workers. Each worker is responsible for retrieving and ranking the documents on its own shard.

For each vertical search, there are existing hand-crafted traditional features. These features contain valuable information for the specific search engine verified by both offline and online experiments. The features can be categorized into three classes: 1). Text matching features. It includes not only exact matching features such as cosine similarity and jaccard similarity, but also semantic matching features, *i.e.*, named entity ids that are obtained by applying in-house entity linking tools [25]; 2). Personalization features. For example, in people search, the social network distance between the searcher and the retrieved profiles; in job search, the searcher's title overlapping with the job post title; and 3). Document popularity features. For example, in people search, static rank of a member profile; in job search/help center, the clickthrough rate of a job post/FAQ document, respectively.

*6.1.4 DeText Models.* Two models are evaluated in this section: DeText-LiBERT (BERT model pretrained on LinkedIn data) and Detext-CNN. The default setting of DeText training is introduced below, unless specified otherwise: 1). *Token embedding layer*: For DeText-CNN models, we always pretrain word embedding on the LinkedIn textual training data with Glove [21], which leads to comparable or better results than no word pretraining or existing word embedding trained on out-domain data. For DeText-LiBERT models, the word embeddings are from a BERT model pretrained on LinkedIn data. 2). *Text embedding layer*: For DeText-CNN, the CNN filter window size is fixed as 3 for all text fields (we do not observe significant gain from using multiple window sizes), and the number of filters is fixed as 64. For DeText-LiBERT, the model structure is described in Section 6.1.5. 3). *Interaction layer*: The best combination of "cosine similarity and hadamard" is used for each dataset. 4). *Feature processing layer*: Both normalization and element-wise re-scaling are performed on the traditional features. 5). *MLP layer*: one hidden layer of size 200. 6). *Learning-to-rank layer*: we stick to listwise, since we find listwise ranking performs better (people search and job search) or comparable (help center) to pairwise ranking.

Regarding training, both DeText-CNN and DeText-LiBERT models are trained for 2 epochs. Adam optimizer [15] is used with learning rate 1e-3; for the BERT component, the learning rate is 1e-5. Each minibatch contains 256 queries with associated documents.

*6.1.5 BERT Pretraining on LinkedIn Data.* The LinkedIn text data has many domain-specific terms, such as "Pinterest", "LinkedIn", resulting in a very different vocabulary from Wikipedia, as used by Google BERT (e.g., $\text{BERT}_{\text{BASE}}$) pretraining. Thus, we pretrained a LiBERT on domain specific data, and then fine tuned the parameters during DeText-LiBERT training.

In order to reduce model serving latency, we use a smaller architecture compared to Google's BERT base model [8]: (6 layers, 512 hidden, 8 heads). The resulting model has 34 million parameters, 1/3 of Google's $\text{BERT}_{\text{BASE}}$ model. We also use less data (around 1/5) than $\text{BERT}_{\text{BASE}}$ model, 685 million words vs 3.3 billion words. The statistics of LinkedIn data is listed in Table 4. Although LiBERT pretraining is conducted in an unsupervised learning manner, we collected pretraining data from the time period prior to the three verticals' training data collection period, to ensure there is no potential data leaking bias.

**Table 4: LinkedIn data for BERT pretraining.**

| Data Source | Description | # Words |
|---|---|---|
| Search queries | Query reformulation pairs | 204M |
| Member profiles | Member headlines and summaries | 98M |
| | Member position titles and descriptions | 105M |
| Job posts | Job titles and descriptions | 217M |
| Help center | Queries and doc titles | 61M |

**Table 5: Offline NDCG@10 score percentage lift in three searches over the production baseline XGBoost. † and ‡ denote statistically significant improvements at p < 0.05 using a two-tailed t-test over XGBoost and DeText-CNN, respectively.**

| Models | People Search | Job Search | Help Center |
|---|---|---|---|
| DeText-MLP | −0.07% | +0.05% | +0.15% |
| DeText-CNN | +3.02%$^{\dagger}$ | +4.65%$^{\dagger}$ | +11.56%$^{\dagger}$ |
| DeText-LiBERT | +3.38%$^{\dagger\ddagger}$ | +6.14%$^{\dagger\ddagger}$ | +13.94%$^{\dagger\ddagger}$ |

**Table 6: General BERT vs in-domain BERT on NDCG@10.**

| Model | People Search | Job Search | Help Center |
|---|---|---|---|
| DeText-CNN | +3.02% | +4.65% | +11.56% |
| DeText-BERT$_{\text{BASE}}$ | +3.08% | +3.60% | +13.80% |
| DeText-LiBERT | +3.38% | +6.14% | +13.94% |

## 6.2 Search Ranking Experiments

*6.2.1 Offline Experiments.* All the relative percentage lift is calculated w.r.t the production baseline model.

**Overall:** Table 5 summarizes the offline NDCG percentage lift in the three search datasets. To understand the impact of deep NLP on text data, we included one baseline model DeText-MLP (DeText with only MLP and LTR layers on traditional features). Since DeText-MLP does not use any text embedding, it has comparable results as XGBoost, which is also observed in previous works [17]. For DeText-CNN, it consistently outperforms the strong production baseline model by a large margin. DeText-LiBERT is able to further improve the NDCG scores. The performance of DeText-CNN/DeText-LiBERT shows that deep learning models are able to capture a lot of semantic textual matching, hence a necessary complement to the existing hand-crafted features.

Meanwhile, it is worth noting that in Table 5 deep learning models achieve the largest improvement on help center, followed by job search and people search. This is mainly caused by the genre of the data, as discussed in Section 3: 1). In the help center, there are many paraphrases of the same scenarios, for example, query "how to hide my profile updates" to FAQ document "Sharing profile changes with your network". 2). In people search, exact matching is much more important as compared to the other two searches, for example, if the query contains the company word "twitter", generally we should not return a member profile who works at "facebook", even though the word embedding of the two companies could be similar. 3). Job search has less paraphrasing than help center, but more search exploration compared to people search.

**LiBERT v BERT$_{\text{BASE}}$:** The impact of pretrained BERT on LinkedIn data is evaluated and shown in Table 6. In people search and job search, DeText-LiBERT significantly outperforms google's BERT$_{\text{BASE}}$, i.e., DeText-BERT$_{\text{BASE}}$, which should be attributed to the pretraining on in-domain data. In the help center where vocabulary and language are closer to Wikipedia, LiBERT can still achieve comparable results. It is worth noting LiBERT has only 1/3 of the parameters of BERT$_{\text{BASE}}$.

**Limit of CNN:** To better understand the trade-off on DeText-CNN models w.r.t. efficiency and effectiveness, we experimented with different numbers of CNN filters, shown in Table 7. We observed with a large number of filters, the gain on people and job search is relatively small (less than +0.4%). This is probably because the powerful hand-crafted features on people/job search are already integrated in the DeText model. Based on the results, we decided to adopt the CNN model with 64 filters in production to reduce the online serving latency.

**Text Embedding Interaction:** Table 8 shows the impact of different text embedding interaction methods. We used cosine similarity as a baseline, and gradually added features computed by other interaction methods. The experiments show that using both cosine similarity and hadamard product features can produce the best or 2nd best results.

**Table 7: Number of CNN filters in DeText-CNN on NDCG@10.**

| #Filters | People Search | Job Search | Help Center |
|---|---|---|---|
| 64 | +3.02% | +4.65% | +11.56% |
| 128 | +3.07% | +4.81% | +11.94% |
| 256 | +3.10% | +4.82% | +12.37% |
| 512 | +3.16% | +4.92% | +12.74% |

**Table 8: Text embedding interaction in DeText-CNN on NDCG@10.**

| Interaction | People | Job | Help Center |
|---|---|---|---|
| cosine | +2.67% | +4.25% | +11.02% |
| cosine, hadamard | **+3.02%** | +4.65% | **+11.56%** |
| cosine, concat | +2.39% | +4.62% | +10.09% |
| cosine, hadamard, concat | +2.84% | **+4.73%** | +11.49% |

**Table 9: Traditional features for DeText-CNN models on NDCG@10. The first row does not use any traditional features.**

| Trad-ftr | Rescale | Norm | People | Job | Help Center |
|---|---|---|---|---|---|
| ✗ | ✗ | ✗ | −4.52% | −9.98% | +11.07% |
| ✓ | ✗ | ✗ | +2.31% | +3.17% | +11.13% |
| ✓ | ✗ | ✓ | +2.47% | +3.44% | +11.55% |
| ✓ | ✓ | ✗ | +2.71% | +4.49% | +11.24% |
| ✓ | ✓ | ✓ | +3.02% | +4.65% | +11.56% |

**Table 10: The impact of using multiple fields. In the single target field setting, the most important field is used: headline for people search and job post title for job search. In this experiment, all the traditional features are excluded. Note that DeText-CNN with a single target field is a special version of CLSM model [26] that operates on words.**

| Model | #fields | People | Job |
|---|---|---|---|
| DeText-CNN (CLSM on words) | single | −5.20% | −12.83% |
| DeText-LiBERT | single | −3.14% | −10.30% |
| DeText-CNN | multiple | −4.52% | −9.98% |
| DeText-LiBERT | multiple | −2.51% | −7.20% |

**Table 11: Online experiments of DeText-CNN and DeText-LiBERT.**

| Search | Model | Metrics | Percentage Lift |
|---|---|---|---|
| People search | DeText-CNN | CTR@5 | +1.13% |
| | | Session Success Rate | neutral |
| | DeText-LiBERT | CTR@5 | +1.56% |
| | | Session Success Rate | +0.23% |
| Job search | DeText-CNN | CTR@5 | +3.16% |
| | | Job Apply | +0.73% |
| Help center | DeText-CNN | Happy Path Rate | +15.0% |
| | | Session Success Rate | +6.1% |
| | DeText-LiBERT | Happy Path Rate | +26.1% |
| | | Session Success Rate | +11.1% |

**Traditional Features:** We evaluated the importance of processing traditional features, as shown in table 9. The first row, where no traditional features are used, proves that the traditional features are crucial in people/job search to capture social networks and personalization signals. In addition, both feature element-wise rescaling and normalization techniques are helpful; applying them together yields the best results.

**Multiple Fields:** Table 10 shows the impact of multiple document fields (using all the fields described in Section 6.1.1). To provide a dedicated comparison, we excluded the traditional features in this experiment. The results demonstrate that using multiple document fields can significantly improve the relevance performance. This is a practical solution for many real-world applications, since the documents in vertical search engines could be semi-structured with many text fields containing additional valuable information.

*6.2.2 Online Experiments.* We performed online experiments in the production environment with model candidates showing promising offline performance. The experiments are conducted with each model under at least 20% traffic for more than two weeks, and the best models are later fully ramped to production. All reported metrics are statistically significant (p-value < 0.05) over the production baseline XGBoost.

For LiBERT models, document embeddings are refreshed daily. However, in job search, there are many new job postings on an hourly basis, which requires the embedding precomputing in a more frequent manner such as near-line update. Due to the computational resources and product priority, we leave the online experiment of DeText-LiBERT on job search to future work.

Table 11 summarizes the experiments of DeText-CNN/DeText-LiBERT on three search engines. From CTR@5 on people and job search, we observed a similar trend in online/offline metrics: the improvement on job search is larger than on people search. Furthermore, DeText-LiBERT is consistently better than DeText-CNN in people search and help center, indicating the importance of contextual embedding on capturing deep semantics between queries and documents in search.

*6.2.3 Latency Performance.* To better understand the latency performance, the offline P99 latency on people search is provided in Table 12. Similar patterns on job search and help center are observed, and they are not presented due to limited space. For each worker, there are thousands of documents to score. The CNN model in the two pass ranking will score hundreds of documents. All numbers are computed by a Intel(R) Xeon(R) 8-core CPU E5-2620 v4 @ 2.10GHz machine and 64-GB memory.

**Table 12: The latency at 99 percentile on people search.**

| Model | Deployment Strategy | Time |
|---|---|---|
| DeText-CNN People | all-decoding | +55ms |
| DeText-CNN People | two pass ranking | +21ms |
| DeText-LiBERT people | doc pre-computing | +43ms |
| DeText-$\text{BERT}_{\text{BASE}}$ people | doc pre-computing | +71ms |

We also compared with another variant, all-decoding, that is to score all the retrieved documents on the fly. By comparing the first two settings in Table 12, it proves two pass ranking is effective at reducing the P99 latency. Meanwhile, the online A/B test does not show significant relevance difference between all-decoding and two pass ranking strategies.

With the document precomputing strategy, we are able to fully ramp the DeText-LiBERT models to production within latency requirements. In addition, we are interested in the LiBERT performance w.r.t. $\text{BERT}_{\text{BASE}}$. Our experiments suggest that DeText-LiBERT is faster than DeText-$\text{BERT}_{\text{BASE}}$, due to the smaller model structure of the former.

## 6.3 Extension of DeText to Other Tasks

In this section, we show the great potential of applying DeText to applications beyond search ranking. We conducted extra experiments on two additional ranking tasks: job recommendation and query auto completion from job search.

For job recommendation, we model the job application probability. The input is a tuple of user id, job post id, and whether the user applied for the job or not. The source fields are from user profiles, including headline, job title, company, and skill. The target fields are from job posts, including job title, job company, job skill, and job country. We used logistic regression as a baseline that is close to production setting, and evaluated with AUC [9] metrics. For fair comparison, point-wise ranking (binary classification) is used with no hidden layer of MLP in DeText. Traditional features are kept the same as in the baseline model.

For query auto completion, the source fields are from member profiles, including headline, job title, and company; the target is the completed query. The baseline model is XGBoost with traditional hand-crafted features. We used the same set of traditional features in DeText with listwise LTR, and evaluated with MRR@10 [2], which is the reciprocal of the rank position of the correct answer.

Table 13 shows the offline results. DeText outperforms the baseline models in both tasks by a large margin, indicating that DeText is flexible enough to be applied in other ranking tasks.

**Table 13: Offline experiments of DeText-CNN on job recommendation and query auto completion datasets. Both improvements are statistically significant at $p < 0.05$.**

| Tasks | Metrics | Percentage Lift |
|---|---|---|
| Job Recommendation | AUC | +3.01% |
| Query Auto Completion | MRR@10 | +4.72% |

## 7 LESSONS LEARNED

We have conducted various experiments on several ranking tasks, where multiple practical methods are used regarding offline relevance, online deployment, latency optimization, etc. In this section, we summarize the interesting findings and practical solutions into lessons, which could be helpful for both academic and industry practitioners who apply deep NLP models for ranking tasks.

**Deep NLP model performance w.r.t. language genre.** Deep NLP models, especially BERT, are strong at handling paraphrasing. Help center is a good fit, since the queries are close to natural language with rich variation. For people search where queries are mostly named entities, the improvement of both CNN and BERT is smaller. Job search lies in between.

**Pretraining BERT on in-domain data makes a big relevance difference.** The common practice of using BERT is to pretrain on general domain such as Wikipedia, and then fine-tune it for a specific task. Our experiments suggest that for vertical search systems, it is better to pretrain BERT on in-domain data. Table 6 shows that, with only 1/3 of the parameters of $\text{BERT}_{\text{BASE}}$, LiBERT significantly outperforms $\text{BERT}_{\text{BASE}}$ on people search and job search, while reaching a similar performance on help center.

**Handling traditional features.** Production models are strong and robust with many hand-crafted traditional features. We observed that 1). after carefully handling these features (Table 9), deep ranking models can achieve better performance than the production models. 2). When combining the traditional features with the BERT model, different learning rates should be used.

**Latency reduction solutions.** Latency is one of the biggest challenges to productionize deep learning models, especially the search ranking tasks that involve many documents for one search. In this paper, we present several effective solutions:

- For heavy models such as BERT, document pre-computing can save a large amount of computation. Note that the prerequisite is representation based structure.
- With two pass ranking, we can deploy a compact CNN based ranking model for real time inference in production for both queries and documents.
- Pretraining a BERT model on in-domain data can maintain the same level of relevance performance, while significantly reducing computation.

## 8 CONCLUSIONS

In this paper, we propose the DeText (deep text) ranking framework with BERT/CNN based ranking model for practical usage in industry. To accommodate the requirements of different ranking productions, DeText allows flexible configuration, such as input data, text embedding extraction, traditional feature handling, etc. These choices enable us to experiment and develop scalable neural network models with strong relevance performance. Our offline experiments show that DeText-LiBERT/DeText-CNN consistently outperforms the strong production baselines. The resulting models are deployed into three vertical searches in LinkedIn's commercial search engines.

## REFERENCES


[1] Selim Aksoy and Robert M Haralick. 2001. Feature normalization and likelihood-based similarity measures for image retrieval. *Pattern recognition letters* (2001).
[2] Ziv Bar-Yossef and Naama Kraus. 2011. Context-sensitive query auto-completion. In *WWW*.
[3] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11 (2010).
[4] Christopher J Burges, Robert Ragno, and Quoc V Le. 2007. Learning to rank with nonsmooth cost functions. In *NeurIPS*.
[5] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *KDD*.
[6] Zhuyun Dai and Jamie Callan. 2019. Deeper Text Understanding for IR with Contextual Neural Language Modeling. In *SIGIR*.
[7] Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. 2018. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *WSDM*.
[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*.
[9] Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern recognition letters* (2006), 861–874.
[10] Mihajlo Grbovic and Haibin Cheng. 2018. Real-time personalization using embeddings for search ranking at airbnb. In *KDD*.
[11] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W Bruce Croft. 2016. A deep relevance matching model for ad-hoc retrieval. In *CIKM*.
[12] Sepp Hochreiter and JÃijrgen Schmidhuber. 1997. Long short-term memory. In *Neural computation*.
[13] 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*.
[14] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *EMNLP*.
[15] Diederick P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.
[16] Yann LeCun and Yoshua Bengio. 1995. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks* (1995).
[17] Pan Li, Zhen Qin, Xuanhui Wang, and Donald Metzler. 2019. Combining Decision Trees and Neural Networks for Learning-to-Rank in Personal Search. In *KDD*.
[18] Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage Re-ranking with BERT. *arXiv preprint arXiv:1901.04085* (2019).
[19] Sankar K Pal and Sushmita Mitra. 1992. Multilayer perceptron, fuzzy sets, and classification. *IEEE Transactions on neural networks* (1992).
[20] Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab Ward. 2016. Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *TASLP* (2016).
[21] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*.
[22] Yifan Qiao, Chenyan Xiong, Zhenghao Liu, and Zhiyuan Liu. 2019. Understanding the Behaviors of BERT in Ranking. *arXiv preprint arXiv:1904.07531* (2019).
[23] Rohan Ramanath, Hakan Inan, Gungor Polatkan, Bo Hu, Qi Guo, Cagri Ozcaglar, Xianren Wu, Krishnaram Kenthapadi, and Sahin Cem Geyik. 2018. Towards Deep and Representation Learning for Talent Search at LinkedIn. In *CIKM*.
[24] Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. In *ACL*.
[25] Dan Shacham, Uri Merhav, Qi He, and Angela Jiang. 2017. Context-aware map from entities to canonical forms. US Patent App. 15/189,974.
[26] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. A latent semantic model with convolutional-pooling structure for information retrieval. In *CIKM*.
[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*.
[28] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. 2017. End-to-end neural ad-hoc ranking with kernel pooling. In *SIGIR*.
[29] Dawei Yin, Yuening Hu, Jiliang Tang, Tim Daly, Mianwei Zhou, Hua Ouyang, Jianhui Chen, Changsung Kang, Hongbo Deng, Chikashi Nobata, et al. 2016. Ranking relevance in yahoo search. In *KDD*.
[30] Hamed Zamani, Bhaskar Mitra, Xia Song, Nick Craswell, and Saurabh Tiwary. 2018. Neural ranking models with multiple document fields. In *WSDM*.