Light Feed-Forward Networks for Shard Selection in Large-scale Product Search

Heran Lin[*]
Amazon.com, Inc.
Palo Alto, CA
linhr@amazon.com

Pengcheng Xiong[*]
Amazon.com, Inc.
Palo Alto, CA
xpengche@amazon.com

Danqing Zhang[*]
Amazon.com, Inc.
Palo Alto, CA
danqinz@amazon.com

Fan Yang
Amazon.com, Inc.
Palo Alto, CA
yafa@amazon.com

Ryoichi Kato
Amazon.com, Inc.
Palo Alto, CA
ryoichi@amazon.com

Mukul Kumar
Amazon.com, Inc.
Palo Alto, CA
mrajk@amazon.com

William Headden
Amazon.com, Inc.
Palo Alto, CA
headdenw@amazon.com

Bing Yin
Amazon.com, Inc.
Palo Alto, CA
alexbyin@amazon.com

ABSTRACT

Large-scale information retrieval systems store documents in different shards. Shard selection enables cost-effective retrieval by searching only relevant shards for the query. Most existing shard selection algorithms focus on web search, and rely on text similarity between the query and shard corpora. In contrast, in e-commerce product search, shards are defined according to product categories, and most queries imply product category intent. Such characteristics are yet to be leveraged for shard selection. In this work, we formulate shard selection in product search as a multi-label query intent classification problem. We show that light feed-forward neural networks, with language-independent features, suffice to achieve high performance for this recall-oriented task. The simple architecture allows for low-latency shard selection in the early retrieval process. We evaluate the model in terms of cost reduction and impact on the relevance of retrieved documents, both in offline simulation and online A/B testing. Without degrading customer experience, we achieve double-digit percentage of search engine cost reduction in multiple locales, and the model has been deployed to serve Amazon Search customers worldwide.

KEYWORDS

product search, federated search, shard selection, deep learning

ACM Reference Format:
Heran Lin, Pengcheng Xiong, Danqing Zhang, Fan Yang, Ryoichi Kato, Mukul Kumar, William Headden, and Bing Yin. 2020. Light Feed-Forward Networks for Shard Selection in Large-scale Product Search. In *Proceedings of ACM SIGIR Workshop on eCommerce (SIGIR eCom'20).* ACM, New York, NY, USA, 8 pages.

[*]The authors contributed equally to this work. The names are listed alphabetically.

1 INTRODUCTION

Online shopping has become an important part of people's daily life in recent years. The growth of e-commerce is accompanied with the growing cost to host product search engines, as more products are indexed and more queries are issued by customers. Sustainable growth of the business calls for control of the infrastructure cost while maintaining the same quality of service to the customer.

We consider the problem of product search cost reduction in the context of distributed information retrieval [7], or *federated search.* The documents (products) are partitioned in different *shards* according to the product category. Each shard hosts an inverted index and runs document retrieval algorithms independently. When the customer issues a query, the *shard selection* algorithm chooses the relevant shards to query. Within each shard, there are multiple *replicas*, which are servers that host the same inverted index. When each selected shard receives the request, the load balancer forwards the request to one replica which returns a ranked list of relevant documents. The results from different selected shards are merged into a single, coherent ranked list and are returned to the customer through *result merging*. The result merging algorithm is critical to balance the relevance and diversity of the result, especially for broad queries such as "harry potter", for which movies, books, or toys may be meaningful results for different customers.

Shard selection is critical to reduce retrieval cost in large-scale product search systems, but there are several **challenges**. First, most existing shard selection algorithms rely on corpora built from the documents in each shard [2, 3, 10, 19, 27–29], which are applicable to web search. In contrast, product catalog contains limited length of text information (e.g. title, description) while being rich in various attributes (e.g. category membership). Second, shard selection is part of the early retrieval process so it has a strict requirement on the inference latency. Third, for capacity planning purposes, it is important to quantify infrastructure cost in industrial-strength product search systems upon software changes.

In this paper, we explore deep learning models for shard selection in large-scale product search. Instead of utilizing text information from documents for shard selection, we propose a way to leverage the product's shard membership and past customer feedback of product interaction. Since most product search queries have clear product category intent and the association between queries and shards is often sparse, we formulate the classical shard selection problem in federated search as a multi-label query intent classification problem. We experiment with different model architectures and different textual features extracted from the query. To balance model quality and inference latency, we conclude that light feed-forward networks are the most appropriate for production. We deploy the deep learning model to run CPU-based online inference for infrequent queries, while shard selection for frequent queries is based on memorized statistics from past customer behavior. We evaluate cost reduction and impact on relevance of retrieved documents, both in offline simulation and online A/B testing. For online A/B testing, we present a method to estimate infrastructure cost reduction based on empirical curves between server throughput and utilization. We demonstrate that light feed-forward networks allow us to achieve double-digit percentage reduction in infrastructure cost across multiple Amazon locales, while maintaining the same relevance of retrieved documents.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 gives a more detailed description of the shard selection problem following which our model architecture is presented in Section 4. Section 5 and Section 6 discuss offline and online experiment results. We conclude in Section 7.

2 RELATED WORK

2.1 Shard Selection for Federated Search

Shard selection is an important task in federated search, which is to select a small number of the most relevant shards to retrieve documents for a given query. There are three approaches to shard selection in federated search: term-based approach, sample-based approach and feature-based approach. Most early resource selection algorithms treat each individual source as a single big document and the problem is reformulated as a document ranking problem. Variants of TF-IDF [8] and language models [1, 22] have been used for such "large document ranking". Sample-based methods estimate the relevance of a shard by querying a small sample index of the collection, known as the centralized sample index (CSI). The query is run against the CSI and document ranking is returned. Given the document ranking of the CSI, document ranking of the centralized complete database can be approximated with different methods [19, 27–29]. The ranking of sampled documents of each shard is then used to quantify the relevance of each shard to the query. Recent methods focus on utilizing different features to estimate the relevance between a query and a shard. These methods use training data to either train a classification model [2, 3, 10], or a learning-to-rank model [12]. Existing shard selection classification models are generative models. The probability for shard s_i given query q is defined as

$$p(s_i|q) = \frac{p(q|s_i)p(s_i)}{p(q)} = \frac{p(q|s_i)p(s_i)}{\sum_j p(q|s_j)p(s_j)} \tag{1}$$

where $p(q|s_i)$ can be estimated using a language model or TF-IDF.

Features used by traditional resource selection methods are 1) query-dependent corpus features, which describe how well the query matches the corpus, such as CORI [8], GAVG [24], and ReDDE [3], 2) query-independent features such as shard popularity which acts as a shard prior.

2.2 Query Intent Classification

We reformulate shard selection as a multi-label query intent classification problem. Therefore, we review some prior work on query classification in information retrieval. Due to the small size of training data, most of the early research studies how to augment the query string input with various types of features, possibly derived from query logs [4], click-through data [32], search sessions [9], or documents in the target categories [20, 25]. Recent research use larger training query dataset and deep learning methods for query feature embedding. [14, 35, 36] propose to use convolutional neural networks (CNN) to extract query vector representations as the features for query classification. [17, 26] build LSTM (long short-term memory) for query classification on top of the query embedding.

2.3 Resource-Constrained Deep Learning

Transformer-based model architectures such as BERT (Bidirectional Encoder Representations from Transformers) [13] and XLNet [34] achieve large improvements on different NLP tasks, and normally they are fine-tuned for the downstream NLP tasks. However, these large-scale pretrained NLP models often have hundreds of millions of parameters, and are considerably slower than traditional statistical models. For production online serving with stringent latency constraints and very high request volumes, and in resource-constraint environments such as mobile phones, we have to balance model quality and inference latency.

The first line of work is to compress large neural networks for fast inference [6, 15, 23]. The other line explores small and shallow neural networks to achieve near state-of-the-art results on NLP tasks [5].

3 PROBLEM DEFINITION

As we have seen in Section 2, most existing work applies to web search and utilizes text information from documents. However this could be expensive and complex for product search systems at large scale, due to enormous product catalog, constant document update and different SLA (service level agreement) of various data feeds. In this work, we investigate shard selection by modeling product category intent in product search queries, since each shard corresponds to one product category. We define shard relevance given query q as

$$p(s|q) = \frac{\sum_{d \in s} n_{d,q}}{\sum_d n_{d,q}} \tag{2}$$

where $n_{d,q}$ is the number of clicks of document d for query q. We quantify shard relevance using document relevance and shard membership, similar to ideas found in the literature [10]. Here the document relevance is measured by clicks in the query log, which can be viewed as implicit feedback from customers. Since the shard has product category semantics, we note that our definition of shard relevance captures product category intent implied in the query.

In product search, one document may exist in multiple shards. For example, a "running shoes" product can exist in both "Clothing, Shoes & Jewelry" and "Sports & Outdoors" shards on the amazon.com website. Therefore we are solving a multi-label classification problem rather than a multi-class classification problem, which means $\sum_i p(s_i|q)$ does not always equal 1. We use $\mathbf{y}$ to denote the vector of relevance scores of all shards given query q where $y_i = p(s_i|q)$ corresponds to the relevance score of the i-th shard s_i. Our goal is to train a model to get prediction $\hat{\mathbf{y}}$ for an arbitrary input query. Given the predicted scores for a given query, shard selection can be done by applying a threshold α, shown as

$$\mathcal{S}_\alpha = \{s_i|\hat{y}_i > \alpha\}. \tag{3}$$

4 MODEL ARCHITECTURE

We propose to model the shard selection classification problem in a discriminative manner, making fewer assumptions about distributions within each label and relying more on data to learn the decision boundary for labels. The core problem of designing a discriminative multi-label classification model for the input query is to design the feature vector $\mathbf{x}$ and design a function $f(\mathbf{x})$ (possibly with parameters to be learned) to transform the features to some other representation. The prediction can then be written as

$$\hat{\mathbf{y}} = \text{sigmoid}(\mathbf{W} \cdot f(\mathbf{x}) + \mathbf{b}) \tag{4}$$

where matrix $\mathbf{W}$ and vector $\mathbf{b}$ are also parameters to be learned. The sigmoid function is applied to each value individually in the input vector. We can learn the parameters using binary cross-entropy (Equation 5) as the loss function.

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_i y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i). \tag{5}$$

In contrast, multi-class classification applies the softmax function at the end so that $\sum_i \hat{y}_i = 1$. The choice of loss function also makes the distinction between multi-label classification and multi-class classification, where for the latter case categorical cross-entropy is used, which does not contain the $(1 - y_i)\log(1 - \hat{y}_i)$ term.

If each dimension of the input feature vector represents the presence of a token in the query, and f is the identify function, then the model is essentially performing logistic regression jointly for all the labels. As deep learning has demonstrated effectiveness in natural language processing tasks, we can have more sophistication in the design of function $f(\mathbf{x})$, which can be a recurrent neural network (RNN) [11, 16], or a Transformer [30]. However, while such models are better at modeling the dependency among tokens, it achieves high performance at great computation cost. When the input query often consists of a few tokens and does not resemble a complete natural language sentence, we argue that we do not need high model capacity to achieve high performance for our recall-oriented classification task. Also, recent work has shown that small and shallow feed-forward networks can achieve good performance in NLP with high efficiency in training and inference [5]. Inspired by these observations, we design a light feed-forward network for our prediction task.

Figure 1 illustrates the feed-forward model architecture. The feed-forward network consumes character-level and word-level n-gram tokens. For example, for the query "red running shoes", the list of character trigrams is "`^re`", "`red`", "`ed␣`", "`d␣r`", "`␣ru`", "`run`", ..., "`oes`", "`es$`", where "`^`" and "`$`" mark the start and end of the query and "`␣`" is the whitespace symbol.

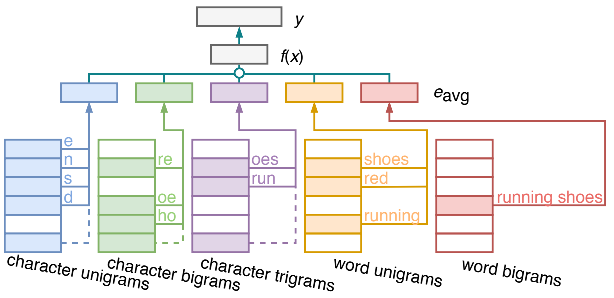

Figure 1: Feed-forward network for shard selection.

We define fixed-size buckets for character unigrams, bigrams, and trigrams where the position of each character n-gram is determined using a hash function. We learn the vocabulary of word unigrams and bigrams from training queries, and also define fixed-size buckets for out-of-vocabulary (OOV) word n-grams. The index into buckets or vocabularies are used to look up embedding vectors in the embedding table of each n-gram type. The "hashing trick" [31] is useful to restrict the size of the feature space, which empirically makes the model robust against misspelling. Including start, end, and whitespace symbols in character n-grams, as well as the addition of word bigrams enable the model to capture word context and dependency to some extent.

The embedding vectors of the same n-gram type are averaged. This is a simple way to summarize the entire n-gram sequence into a single dense vector. Typically recurrent neural network (RNN) would be used, but we observe that product search queries tend to be short and usually few dependency need to be modeled among tokens to capture the query semantics. As we shall see in Section 5.1, embedding average offers similar performance as RNN models for our use case.

Finally, the embedding averages are concatenated to form the whole-query embedding $\mathbf{e}_{\text{avg}}$. We transform the query embedding using a ReLU layer to get the final representation of the query

$$f(\mathbf{x}) = \text{relu}(\mathbf{W}_h \cdot \mathbf{e}_{\text{avg}} + \mathbf{b}_h) \tag{6}$$

where matrix $\mathbf{W}_h$ and vector $\mathbf{b}_h$ are also parameters to be learned.

5 OFFLINE EVALUATION

In this section we discuss two offline evaluation tasks. The first task measures the model performance at the shard level. The second task quantifies changes at the document level by simulating shard cutoff against our production search engine.

5.1 Model Performance

5.1.1 Dataset. We aggregate product click counts from anonymized query logs within a certain time window in the US (amazon.com) locale. Using the click count for all query-product pairs, we calculate the score of shards for all queries, using the membership information from products to shards according to the catalog. We remove queries whose frequency is below a certain threshold. This results in a dataset of 273 million unique queries. Since the dataset is huge, we use 98% of the data for training, and sample around 185,000 samples from the remaining 2% data for validation, and testing respectively. A query in the training set will never appear in the validation or test set.

We apply similar steps to obtain datasets in other locales to evaluate model performance across languages.

5.1.2 Metrics. We use binary cross-entropy as the loss function for training as well as model selection. At test time, we consider metrics that are more relevant to our shard selection task. We consider classification metrics rather than ranking metrics at the shard level, because the relative order does not change shard selection results.

Let $\mathbf{y}$ and $\hat{\mathbf{y}}$ be two vectors for the true and predicted shard relevance, where the i-th component is the probability for the i-th category. We first consider measuring the similarity between the two vectors. We notice that mean squared error (MSE) may not be the best option since the vector tends to be sparse so the metric is often of a very small value. Therefore we choose Jaccard similarity defined as follows.

$$\text{Jaccard}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\sum_i \min(y_i, \hat{y}_i)}{\sum_i \max(y_i, \hat{y}_i)}. \tag{7}$$

We define precision and recall at threshold α in Equation 8 and Equation 9, where $I[\cdot]$ is the indicator function.

$$\text{Precision}_\alpha(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\sum_i I[y_i > \alpha \wedge \hat{y}_i > \alpha]}{\sum_i I[\hat{y}_i > \alpha]}, \tag{8}$$

$$\text{Recall}_\alpha(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\sum_i I[y_i > \alpha \wedge \hat{y}_i > \alpha]}{\sum_i I[y_i > \alpha]}. \tag{9}$$

Note that the precision and recall metrics defined here are slightly different than their standard definition, where the ground-truth are binary labels and while the model may output a probability. The definition tailored to our task lets us understand how well the predicted shard cutoff mimics the decision made from past behavior data, at different threshold α. However the precision-recall curve may not be meaningful as the ground-truth also changes for different data points on the curve.

For all metrics, we report the average across all queries in the test set.

5.1.3 Benchmark Models. As simplicity is one of our design tenets, our model may not be the best-performing one. To better understand where our model lies in the performance spectrum, we compare the feed-forward architecture with other popular architectures found in the literature. We first consider using the LSTM model [16] to replace embedding average, where for each type of n-grams, the token embeddings are fed to an LSTM model as ordered input, and the state vector of the last timestamp is obtained as the summary of the corresponding n-gram type. The input vector and state vector are of the same size.

We also consider BERT [13], which has achieved the state-of-the-art in a variety of NLP tasks. It is built on top of the Transformer architecture [30], which consists of a multi-head attention module and a position-wise feed-forward module. Instead of using character or word n-grams, the BERT model takes WordPiece tokens [33] as input that are learned via a separate procedure where the most common co-occurrence between characters are learned and used to form subwords. For example, the query "harry potter" might translate into the following set of word pieces "ha r r y pot ter" where each spaced out set of letters is a subword or WordPiece. We evaluate two uncased English-only pre-trained models of different sizes. BERT-Base uses 12 layers of 12-head Transformers with 768-dimension hidden layers, while BERT-Large uses 24 layers of 16-head Transformers with 1024-dimension hidden layers. In order to integrate this model with the shard prediction layer, we use part of the final hidden layer of the model as the representation for the entire query, which is referred to as the CLS embedding in the literature.

5.1.4 Results. The metrics of different models in the US locale are shown in Table 1.

"FFN" is our proposed model based on the feed-forward architecture, while "LSTM" is the recurrent neural network benchmark. The character embeddings and of size 128, and word embeddings are of size 256. The ReLU layer are of size 128. We experimented with different combinations of tokenization methods, such as "C1,2,3 W1,2" which means the combination of character unigrams, bigrams, trigrams and word unigrams, bigrams. The embedding tables for character unigrams, bigrams, and trigrams are of size 61, 1021, and 65519, respectively. Only the most frequent 100,000 word n-grams from the training set are allowed in the vocabularies, and we ensure the pointwise mutual information of each n-gram is greater than 0.5 when $n > 1$. Additionally we reserve 100,003 slots in each word n-gram embedding tables for hashing out-of-vocabulary word n-grams. We apply batch normalization and drop-out to the last two layers in the LSTM and FFN models.

"BERT-B" and "BERT-L" refers to the pre-trained BERT-Base and BERT-Large models. As mentioned before, BERT models uses WordPiece tokenization, shown as "WP" in the table.

LSTM or FFN models are trained using one AWS (Amazon Web Services) p3.2xlarge instance with one NVIDIA® V100 Tensor Core GPU, while BERT models are trained using one p3dn.24xlarge instance with eight GPUs. The training cost in the table is measured by the training time multiplied by the number of GPUs being used. For LSTM or FFN models, we allow the model to see about 2 billion samples (i.e. repeating the training set about 7.3 times). Due to large training cost, we feed only 500 million samples to BERT models (i.e. repeating the training set about 1.8 times). Samples are weighted by the logarithm of the query frequency. We select the best model using the validation loss. The Adam optimizer [18] is used for all model training.

The FFN model has lower performance in terms of precision, compared with the LSTM model with the same tokenization method, but the latter can have up to 4 times the training cost as the former. By adding more types of n-grams, the performance gap narrows. The metrics from BERT models show the effectiveness of leveraging external knowledge encoded in pre-trained models of large capacity. We can see that if we use the BERT CLS embedding off-the-shelf, the model perform poorly, so fine-tuning the Transformer layers helps the embedding space adapt to the domain of product search queries. The BERT-Large model has slightly better performance than the BERT-Base model for our task, but it is much more costly to train. When we stop BERT model training, we observe the validation loss is still slowly decreasing, which suggests that we may improve performance marginally if we run model training even longer.

Table 1: Model Performance (US Locale)

Model	LSTM				FFN				BERT-B (no fine-tune)	BERT-B	BERT-L (no fine-tune)	BERT-L
Tokenization	C3	W1	C1,2,3 W1	C1,2,3 W1,2	C3	W1	C1,2,3 W1	C1,2,3 W1,2	WP	WP	WP	WP
# Parameters (million)	9	52	61	112	8	51	60	111	109	109	334	334
Training cost (GPU hour)	12	9	37	42	9	9	9	10	88	190	489	924
Jaccard Similarity	0.654	0.697	0.708	0.717	0.600	0.657	0.674	0.698	0.423	0.736	0.400	0.746
Precision$_\alpha(\alpha = 10^{-3})$	0.252	0.308	0.316	0.333	0.206	0.250	0.275	0.310	0.135	0.337	0.120	0.352
Precision$_\alpha(\alpha = 10^{-2})$	0.510	0.570	0.583	0.595	0.428	0.505	0.526	0.565	0.265	0.621	0.234	0.634
Precision$_\alpha(\alpha = 10^{-1})$	0.757	0.794	0.806	0.812	0.709	0.761	0.772	0.793	0.554	0.824	0.532	0.835
Recall$_\alpha(\alpha = 10^{-3})$	0.989	0.988	0.987	0.987	0.988	0.987	0.987	0.987	0.993	0.989	0.996	0.989
Recall$_\alpha(\alpha = 10^{-2})$	0.945	0.946	0.945	0.946	0.942	0.945	0.945	0.945	0.946	0.948	0.952	0.949
Recall$_\alpha(\alpha = 10^{-1})$	0.884	0.894	0.900	0.903	0.867	0.885	0.892	0.899	0.796	0.911	0.797	0.914

As we mentioned, shard selection is a recall-oriented task. All models achieve high recall at the threshold $\alpha = 0.01$ and below, while the precision varies. Given the same recall, higher precision implies the model can produce more sparse labels and select fewer shards. However the reduced shard cost is counteracted by the cost and latency to host such more precise models, especially Transformer-based models. Moreover, low training cost enables frequent model refresh to capture changes in customer behavior and product catalog, and having up-to-date data can make up for the limitation in model capacity. Given all these tradeoffs, we conclude that feed-forward networks, using all five types of n-grams, are the most suitable for production.[1]

Now that we have shown the effectiveness of the feed-forward architecture, we evaluate its performance for other languages. Table 2 shows the metrics of the following locales: UK (www.amazon.co.uk), DE (www.amazon.de), FR (www.amazon.fr), IT (www.amazon.it), ES (www.amazon.es), JP (www.amazon.co.jp). The model size is kept unchanged, including the vocabulary or hashing bucket size for the five types of n-grams. We can see that the model performs well across languages, without customizing the features for each language. More language-specific optimization can be done, such as better tokenization tailored to Asian languages, but we leave that as future work.

5.2 Shard Cutoff Evaluation

So far we have analyzed model performance using ground-truth shard relevance scores of test queries. However we need a deeper analysis to evaluate the overall retrieval quality considering document ranking and result merging in the production system, using the FFN model of our choice.

We uniformly sample 1,000 unique queries from 1-day query log. Due to the power law distribution, the sample is more skewed towards less frequent queries, where only 10% of the queries occurred more than twice per day. This is a stress test of the generalization capability of our model as the model is less likely to memorize shard relevance for these queries from the training data. It also aligns with our production setting where the model is mostly useful to handle infrequent queries.

As we shall see in the top half of Figure 2, although we do not add any sparsity constraint during model training, the model naturally produces sparse prediction where only less than 20% shards are active at $\alpha = 0.01$. Different thresholds can be chosen to control the aggressiveness of shard selection.

In the bottom half of Figure 2, we analyze the impact of shard selection on the search results for the sample queries. Here we use the metric Overlap@N. It assumes the production system returns oracle ranked lists of relevant documents. At different position cutoff N, we measure how many top-N products can still be found after applying shard selection. Previous work has shown that Overlap@N is more suitable than other metrics to evaluate early-stage retrieval [21]. We report unweighted average of Overlap@N across all the sample queries. Overlap@N drops as shard cutoff becomes more aggressive with higher threshold α, and this observation is consistent across different position cutoff N. At $\alpha = 0.01$, about 3% of the products in the original top-16 results are no longer seen in the top-16 results after shard selection. These products are either removed

Table 2: Model Performance (Additional Locales)

Model	FFN					
Tokenization	C1,2,3 W1,2					
Locale	UK	DE	FR	IT	ES	JP
Jaccard Similarity	0.655	0.669	0.674	0.723	0.685	0.629
Precision$_\alpha(\alpha = 10^{-3})$	0.224	0.246	0.237	0.261	0.247	0.225
Precision$_\alpha(\alpha = 10^{-2})$	0.438	0.457	0.456	0.519	0.463	0.413
Precision$_\alpha(\alpha = 10^{-1})$	0.722	0.729	0.747	0.800	0.754	0.715
Recall$_\alpha(\alpha = 10^{-3})$	0.992	0.993	0.992	0.992	0.992	0.992
Recall$_\alpha(\alpha = 10^{-2})$	0.969	0.970	0.968	0.968	0.968	0.966
Recall$_\alpha(\alpha = 10^{-1})$	0.909	0.910	0.910	0.925	0.913	0.882

[1]Although the 5 n-gram FFN model has similar number of parameters as the BERT-Base model, the FFN parameters are mostly embedding vectors, which occupy memory but have no impact on inference latency. The 5 n-gram FFN model also has similar training cost and performance as the LSTM model with only word unigram feature, but in practice character-level features can make the model robust against spelling errors, so the FFN model is favored.

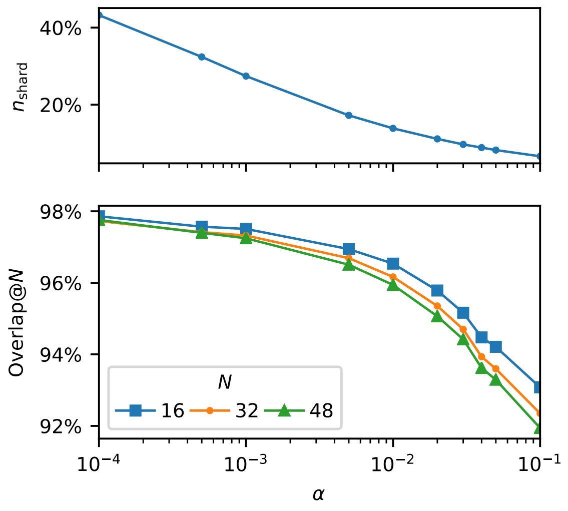

Figure 2: Shard cutoff evaluation. (Top) Average percentage of shards selected by the model at different thresholds α. (Bottom) Overlap@N of top-N results ($N = 16, 32, 48$) before and after shard selection at different thresholds α.

or ranked lower due to interaction between shard selection and result merging, which takes into account the relative popularity and diversity of shards when generating the final ranked list. We zoom into the 3% difference and found that the difference is mainly due to a few queries. These queries have unclear intent, or the website does not sell the product wanted. In such cases, the system tends to return less relevant products to begin with, so applying shard selection does not worsen search quality. In Section 6.2.2 we will further confirm this in online A/B testing.

6 ONLINE EVALUATION

In this section we evaluate shard selection in online A/B testing. We start with a discussion how we model server cost in the product search system. Then we present A/B testing results regarding the impact of shard selection on infrastructure cost and document relevance.

6.1 Infrastructure Cost Modeling

6.1.1 Overview. Large-scale search engines consist of fleets of multi-core CPU machines responsible for matching and ranking indexed documents within shards. Although other servers for request forwarding or result merging incur cost as well, the cost of document retrieval servers is dominant, so it becomes our focus during cost modeling and analysis. Let $P(t)$ be the required capacity (queries per unit time) for a shard at time t. Let λ be the query throughput, or the number of queries processed by a single server per unit time. Let λ^* be the maximum throughput, and ϵ be the fixed cost per second associated with each server. Assuming that all the servers are homogeneous and the load is perfectly balanced, the infrastructure cost over a time period τ is

$$I = \int_0^\tau \frac{P(t)\epsilon}{\lambda^*} \mathrm{d}t \qquad (10)$$

where $\frac{P(t)}{\lambda^*}$ tells us the number of servers needed at time t to meet the demand $P(t)$, and the demand is the real-time traffic plus some buffer. However, in a large scale high-availability system like our product search system, it is difficult to perform horizontal scaling in real-time due to availability risk of on-demand capacity in a shared pool. Therefore we consider the required system capacity being fixed at P^* across the time period so we have

$$I = \int_0^\tau \frac{P^*\epsilon}{\lambda^*} \mathrm{d}t = \frac{P^*\epsilon\tau}{\lambda^*}. \qquad (11)$$

To measure infrastructure cost I after enabling shard selection, we need to estimate the required capacity P^* and the maximum throughput λ^*. P^* can be estimated using the rate of requests that the shard receives. λ^* cannot be estimated directly, because we run A/B testing for a long time period in the production system, where the server is not stressed with the highest load. In the next section, we describe how we use a model between throughput λ and server utilization ρ to estimate λ^*.

6.1.2 Estimating the Maximum Throughput. Using the terminology from queueing theory, we can model a multi-server shard as an $M/M/c$ queue. Queries arrive at rate $c\lambda$ according to a Poisson process, so that each of the c servers handles requests at rate λ, assuming the system is stable and requests are distributed evenly to the servers. Let S be the server service time (excluding the queueing time), which follows an exponential distribution with parameter μ. We have $S = \frac{1}{\mu}$ so the server utilization ρ is defined as

$$\rho = \frac{c\lambda}{c\mu} = \lambda S. \qquad (12)$$

We can derive the empirical relation between S and λ using real system data in Figure 3, where each data point is a sample of a short time interval for a server. We find a quasi-linear correlation

$$S = a\lambda + b \qquad (a, b > 0) \qquad (13)$$

with different constants a and b for different shards. The correlation may be explained by hardware resource contention (e.g. for shared L2 cache or memory bus) for multi-core CPUs as the load increases. By substituting S in Equation 12 with Equation 13, we can write

$$\rho = a\lambda^2 + b\lambda \qquad (a, b > 0). \qquad (14)$$

Figure 4 shows the load-utilization (LU) curve between λ and ρ that we collected for different servers for different shards. We can see that as λ increases, ρ also increases. Correspondingly, the round-trip time for each request (composed of the service time S and the queueing time) and blocking rate also increases. According to certain service level agreement (SLA) of round-trip time and blocking rate, we can empirically define the maximum utilization ρ^*, so by solving Equation 14 we get the maximum throughput as

$$\lambda^* = \frac{-b + \sqrt{b^2 + 4a\rho^*}}{2a} > 0 \qquad (a, b > 0). \qquad (15)$$

6.2 Results

In this section we report the online A/B testing results in Amazon search system. We split the federated search traffic evenly into two groups. The control group (C) is the current production system which has some legacy model that is only able to select shards

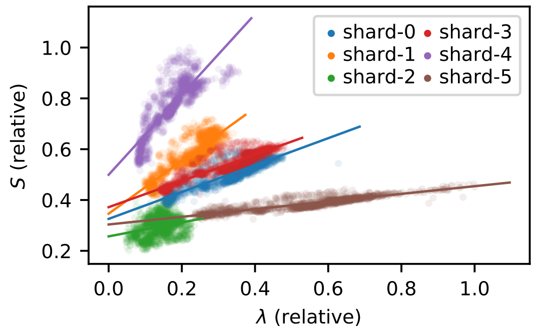

Figure 3: Quasi-linear correlation between throughput λ and service time S.

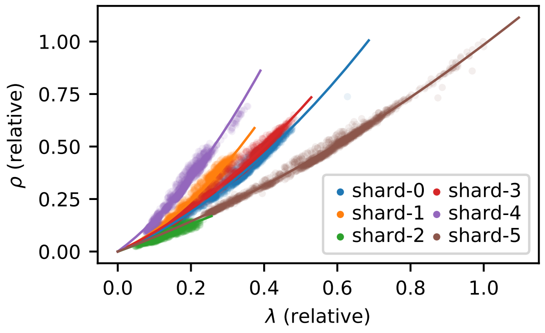

Figure 4: Quasi-quadratic correlation between throughput λ and utilization ρ.

for about 50-60% of the traffic. The treatment group (T) uses our shard selection model. The shard selection decision for the top 65% frequent queries is made by memorizing the training data, while the remaining 35% comes from the deep learning model. Note that since the query frequency follows a power law distribution, the number of unique queries served by the model is much larger than those served using the memorized data. Given the offline analysis in Section 5, we apply the threshold $\alpha = 0.01$ to shard scores for the selection. We run the experiment for one week.

6.2.1 Cost Reduction. We discuss the results obtained from one Amazon locale as an example. Given the even allocation of customer queries, we find that the treatment group only receives 29% of the requests at the shard level, compared with the control group. The rate of traffic drop is different in different product categories, such as Grocery (14%), Lawn & Garden (14%), Toys (30%), Office Products (28%), and Luggage (17%). This shows different categories fulfill a shopping intent to different degrees, and our shard selection model captures such fact. The traffic difference lets us know $P^*_\text{T}/P^*_\text{C}$, the drop in required system capacity.

However queries processed in shards are more expensive in T than C, as we shall see in the load-utilization curves between λ and ρ for C and T in Figure 5. On average the CPU service time in T has increased by 37% compared to C. This is actually a good sign since it shows that we pruned requests to shards which are less likely to contain relevant products for the query, so more product matching

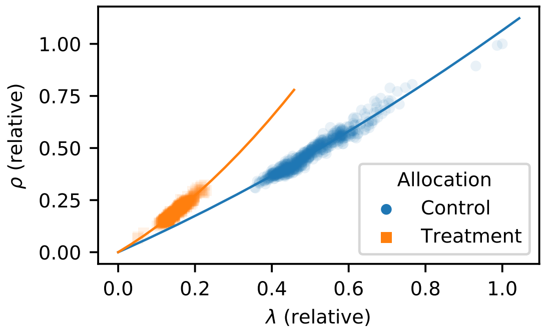

Figure 5: Sample load-utilization curves for one shard in the A/B Testing.

and ranking (which is CPU-intensive) needs to be done per request in T. It was confirmed by the facts that on average T returns 95% more products per shard than C for a single request.

The load-utilization curves for C and T fitted using the A/B testing data are

$$\rho_\text{C} = a_\text{C}\lambda_\text{C}^2 + b_\text{C}\lambda_\text{C}, \tag{16}$$

$$\rho_\text{T} = a_\text{T}\lambda_\text{T}^2 + b_\text{T}\lambda_\text{T}. \tag{17}$$

In order to meet the service level agreement, if we define ρ^*, we can solve for λ^*_C and λ^*_T using Equation 15, from which we can get $\lambda^*_\text{T}/\lambda^*_\text{C}$. Therefore, the overall infrastructure cost saving is

$$1 - \frac{I_\text{T}}{I_\text{C}} = 1 - \frac{\frac{P^*_\text{T}\epsilon\tau}{\lambda^*_\text{T}}}{\frac{P^*_\text{C}\epsilon\tau}{\lambda^*_\text{C}}} = 1 - \frac{P^*_\text{T}/P^*_\text{C}}{\lambda^*_\text{T}/\lambda^*_\text{C}} \tag{18}$$

where $P^*_\text{T}/P^*_\text{C}$ has been measured directly as discussed.

Using our internal value for ρ^* and following the calculation, we find double-digit percentage of cost reduction aggregated across shards in multiple locales.

6.2.2 Relevance. We need to evaluate whether the shard selection is too aggressive as this may cause issues with product discovery and loss of customer trust. This is done by measuring business metrics during A/B testing. No statistically significant changes are detected worldwide in metrics used to measure customer purchase behavior. Interestingly, we find shard selection even improves search result relevance in some cases, aligning with the observation in Section 5.2. Using an internal human judgment process, we find our shard selection decreases the rate of wrong product category results by 0.05% to 0.35% in a few locales ($p < 0.05$). This shows that shard selection helps coping with anomaly in the product catalog.

6.2.3 Model Hosting. The deep learning model for shard selection resides in a service hosting various query intent classification models, which is an upstream dependency of the product search engine. The model runs in Docker containers on CPU machines. Load tests show that the 50-percentile latency is a few milliseconds while the 99-percentile latency is also within our budget. The cost of hosting the deep learning shard selection model is only 1.4% of the search engine infrastructure cost savings that we achieve, so the additional cost is negligible.

7 CONCLUSION & FUTURE WORK

In this paper, we formulated the shard selection problem in product search as a multi-label query intent classification problem. We explained the way to measure shard relevance for queries given the product's shard membership and product clicks. We proposed simple but effective deep learning model architecture for low-cost training and inference. Both offline and online evaluation showed feed-forward networks are sufficient for search engine cost reduction while maintaining the same relevance of retrieved documents. As a next step, we can consider incorporating the threshold as part of the model to achieve dynamic, query-specific shard selection to further reduce search engine cost.

ACKNOWLEDGMENTS

We would like to thank Ashish Sadh, Jagadish Nallapaneni, and Vignesh Iyer for the great effort in deep learning online inference. Nick Blumm led the initial design of this project. Alexander Jaffe and Jiufeng Zhou contributed to dataset preparation, model training and A/B testing. We would like to thank anonymous reviewers for the valuable feedback.

REFERENCES

[1] Aly, R., Hiemstra, D., and Demeester, T. Taily: shard selection using the tail of score distributions. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval* (2013), pp. 673–682.

[2] Arguello, J., Callan, J., and Diaz, F. Classification-based resource selection. In *Proceedings of the 18th ACM conference on Information and knowledge management* (2009), pp. 1277–1286.

[3] Arguello, J., Diaz, F., Callan, J., and Crespo, J.-F. Sources of evidence for vertical selection. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval* (2009), pp. 315–322.

[4] Beitzel, S. M., Jensen, E. C., Frieder, O., Lewis, D. D., Chowdhury, A., and Kolcz, A. Improving automatic query classification via semi-supervised learning. In *Fifth IEEE International Conference on Data Mining (ICDM'05)* (2005), IEEE, pp. 8–pp.

[5] Botha, J. A., Pitler, E., Ma, J., Bakalov, A., Salcianu, A., Weiss, D., McDonald, R., and Petrov, S. Natural language processing with small feed-forward networks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* (Copenhagen, Denmark, 2017), pp. 2879–2885. Supplementary material: http://aclweb.org/anthology/attachments/D/D17/D17-1309.Attachment.zip.

[6] Buciluǎ, C., Caruana, R., and Niculescu-Mizil, A. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* (2006), pp. 535–541.

[7] Callan, J. Distributed information retrieval. In *Advances in information retrieval*. Springer, 2002, pp. 127–150.

[8] Callan, J. P., Lu, Z., and Croft, W. B. Searching distributed collections with inference networks. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval* (1995), pp. 21–28.

[9] Cao, H., Hu, D. H., Shen, D., Jiang, D., Sun, J.-T., Chen, E., and Yang, Q. Context-aware query classification. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval* (2009), pp. 3–10.

[10] Cetintas, S., Si, L., and Yuan, H. Learning from past queries for resource selection. In *Proceedings of the 18th ACM conference on Information and knowledge management* (2009), pp. 1867–1870.

[11] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2014), pp. 1724–1734.

[12] Dai, Z., Kim, Y., and Callan, J. Learning to rank resources. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2017), pp. 837–840.

[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 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (2019), pp. 4171–4186.

[14] Hashemi, H. B., Asiaee, A., and Kraft, R. Query intent detection using convolutional neural networks. In *International Conference on Web Search and Data Mining, Workshop on Query Understanding* (2016).

[15] Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[16] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[17] Kim, J.-K., Tur, G., Celikyilmaz, A., Cao, B., and Wang, Y.-Y. Intent detection using semantically enriched word embeddings. In *2016 IEEE Spoken Language Technology Workshop (SLT)* (2016), IEEE, pp. 414–419.

[18] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[19] Kulkarni, A., Tigelaar, A. S., Hiemstra, D., and Callan, J. Shard ranking and cutoff estimation for topically partitioned collections. In *Proceedings of the 21st ACM international conference on Information and knowledge management* (2012), pp. 555–564.

[20] Li, Y., Zheng, Z., and Dai, H. Kdd cup-2005 report: Facing a great challenge. *ACM SIGKDD Explorations Newsletter 7*, 2 (2005), 91–99.

[21] Mohammad, H. R., Xu, K., Callan, J., and Culpepper, J. S. Dynamic shard cutoff prediction for selective search. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval* (2018), pp. 85–94.

[22] Nottelmann, H., and Fuhr, N. Evaluating different methods of estimating retrieval quality for resource selection. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval* (2003), pp. 290–297.

[23] Sanh, V., Debut, L., Chaumond, J., and Wolf, T. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).

[24] Seo, J., and Croft, W. B. Blog site search using resource selection. In *Proceedings of the 17th ACM conference on Information and knowledge management* (2008), pp. 1053–1062.

[25] Shen, D., Sun, J.-T., Yang, Q., and Chen, Z. Building bridges for web query classification. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval* (2006), pp. 131–138.

[26] Shi, Y., Yao, K., Tian, L., and Jiang, D. Deep lstm based feature mapping for query classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: Human language technologies* (2016), pp. 1501–1511.

[27] Shokouhi, M. Central-rank-based collection selection in uncooperative distributed information retrieval. In *European Conference on Information Retrieval* (2007), Springer, pp. 160–172.

[28] Si, L., and Callan, J. Relevant document distribution estimation method for resource selection. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval* (2003), pp. 298–305.

[29] Thomas, P., and Shokouhi, M. Sushi: scoring scaled samples for server selection. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval* (2009), pp. 419–426.

[30] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems* (2017), pp. 5998–6008.

[31] Weinberger, K., Dasgupta, A., Langford, J., Smola, A., and Attenberg, J. Feature hashing for large scale multitask learning. In *Proceedings of the 26th annual international conference on machine learning* (2009), pp. 1113–1120.

[32] Wen, J.-R., Nie, J.-Y., and Zhang, H.-J. Query clustering using content words and user feedback. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval* (2001), pp. 442–443.

[33] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., and Norouzi, M. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2018).

[34] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems* (2019), pp. 5754–5764.

[35] Zhang, C., Fan, W., Du, N., and Yu, P. S. Mining user intentions from medical queries: A neural network based heterogeneous jointly modeling approach. In *Proceedings of the 25th International Conference on World wide web* (2016), pp. 1373–1384.

[36] Zhang, H., Song, W., Liu, L., Du, C., and Zhao, X. Query classification using convolutional neural networks. In *2017 10th International Symposium on Computational Intelligence and Design (ISCID)* (2017), vol. 2, IEEE, pp. 441–444.