Que2Search: Fast and Accurate Query and Document Understanding for Search at Facebook

Yiqun Liu, Kaushik Rangadurai, Yunzhong He,
Siddarth Malreddy, Xunlong Gui, Xiaoyi Liu, Fedor Borisyuk
Facebook Inc.

ABSTRACT

In this paper, we present *Que2Search*, a deployed query and product understanding system for search. *Que2Search* leverages multi-task and multi-modal learning approaches to train query and product representations. We achieve over 5% absolute offline relevance improvement and over 4% online engagement gain over state-of-the-art Facebook product understanding system by combining the latest multilingual natural language understanding architectures like XLM and XLM-R with multi-modal fusion techniques. In this paper, we describe how we deploy XLM-based search query understanding model that runs <1.5ms @P99 on CPU at Facebook scale, which has been a significant challenge in the industry. We also describe what model optimizations worked (and what did not) based on numerous offline and online A/B experiments. We deploy *Que2Search* to Facebook Marketplace Search and share our deployment experience to production and tuning tricks to achieve higher efficiency in online A/B experiments. *Que2Search* has demonstrated gains in production applications and operates at Facebook scale.

CCS CONCEPTS

• **Information systems** → **Multimedia and multimodal retrieval**; **Online shopping**.

KEYWORDS

Product understanding, e-commerce, multi-modal learning, embedding, deep learning

ACM Reference Format:
Yiqun Liu, Kaushik Rangadurai, Yunzhong He,, Siddarth Malreddy, Xunlong Gui, Xiaoyi Liu, Fedor Borisyuk. 2021. Que2Search: Fast and Accurate Query and Document Understanding for Search at Facebook. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3447548.3467127

1 INTRODUCTION

Facebook Marketplace[1] is a global platform that enables businesses and buyers across the world to buy and sell items. Hundreds of millions of products are listed for sale, and Marketplace provides an ecosystem for buyers to explore, find and buy what they need. We targeted to improve the quality and recall of the Marketplace search engine so that we can surface the right set of products given the user textual query. See figure 1 with output of Marketplace Search engine for the query "hats".

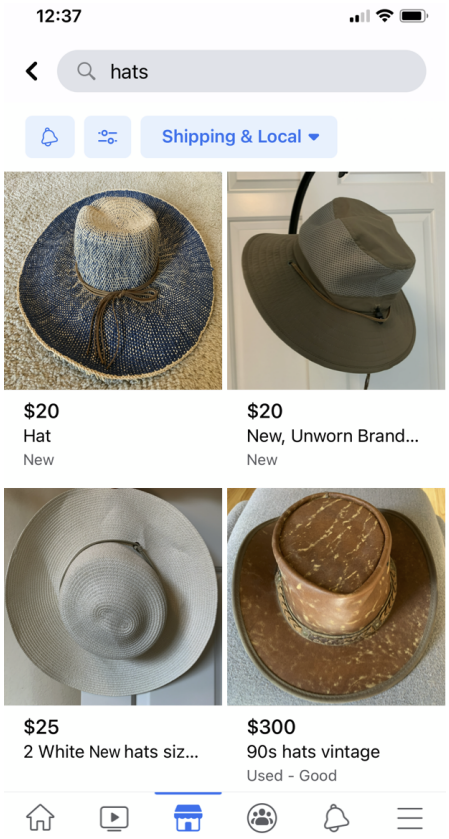

Figure 1: A screenshot of a Facebook Marketplace Search results page.

Traditionally search engines were based on various term matching methods [1]. Recently Huang et al. [14] introduced Embedding-based Retrieval to Facebook search: authors of [14] used character n-grams sparse features to represent textual features and relied on wide and deep neural networks to build embedding representations for the query and document [6]. This approach lacks representation of natural language and did not on-board BERT [10] based models primarily due to high computational requirements. In our previous works, we have built state-of-art image recognition systems (Bell et al. [2], Tang et al. [25]) for product image understanding. In addition to the product image, other textual and categorical product

отсылка к статье про ембедный поиск, в которой они указывают что плох тем, что нет берта

[1]http://www.facebook.com/marketplace

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467127

features, such as title, product description, location and product category, also provide valuable information.

This paper presents *Que2Search*, a query to product similarity model, which provides a modeling approach that takes into account comprehensive multi-modal features and utilizes XLM encoder (Lample and Conneau [17]) for textual features. *Que2Search* is trained on weakly-supervised datasets and achieves state-of-the-art performance for product representation compared to previous baselines at Facebook. *Que2Search* has been deployed to Facebook Marketplace search; it powers query-to-product Embedding-based Retrieval and is also used as a ranking feature. *Que2Search* operates at Facebook Marketplace scale, works across multi-languages and satisfies strict latency constraints.

There were several challenges as we build *Que2Search*, we list some of them here:

- *Noisy product descriptions:* The quality of seller-provided product descriptions varies a lot. In many cases, attributes of the marketplace products are missing or misspelled
- *Internationalization support:* we would like to build a model that would perform well across many languages where Facebook Marketplace is enabled
- *Efficient Handling of multi-modalities:* we need to take into account all multi-modal features, such as products images and textual information, effectively into one model
- *Strict latency constraints:* we need to satisfy strict latency constraints of the search engine, this is particularly challenging as we use transformer-based language models, which are known to be computationally expensive

Recent state-of-the-art technology in language understanding stems from BERT [10] based transformer language models. The challenge of scaling BERT-based models and improving inference latency is a known across the industry[21, 22]. The inference time is especially crucial for search problems where we need to have embedding representation for free form textual query in real-time. We use a 2-layer XLM encoder [17] to transform query text, and on the product side, we adopt a multi-lingual approach with XLM-R [7] encoder to encode textual fields of the product. We share our experience to scale XLM/XLM-R models for search scenarios in §3.5 to run under 1.5 milliseconds on CPU.

Que2Search has demonstrated gains in production applications by introducing several modeling techniques, including multi-stage curriculum learning, multi-modality handling, and multi-task learning that optimizes query-to-product retrieval task and product classification task jointly. We share our modeling techniques in §3, ablation studies in §5, and production deployment experience and tuning tricks to achieve higher performance in online A/B experiments in §6.

2 RELATED WORK

Embedding-based Retrieval. Traditionally Siamese networks [3] have been used for modeling pairwise relationships and is effective for retrieval in production applications because it provides a way to compute object embeddings independently of the query and store them in the search index. A set of recent works in the industry investigated semantic retrieval at scale in various applications using a combination of Siamese networks and wide & deep architecture [6] for each of the towers. Yang *et al.* [27] trained two towers neural network to power retrieval of a commercial mobile app store and introduce mixed negative sampling taking random negatives from the search index in addition to within the batch random negatives. Huang *et al.* [14] introduced the approach of two towers wide & deep network for Facebook search, where they used a set of sparse features based on character n-grams to model textual features and primarily dealt with single modality data. There have been several recent works using advanced BERT [10] based models in search applications such as [11, 23]. Guo *et al.* [11] proposed to use deep networks within the ranking framework and used several optimizations to reduce computations issues such as a smaller model for documents, and pre-calculated document side embeddings before serving. This paper shares our experience utilizing transformer-based language models for textual features and describes practical multi-modal training techniques to fuse textual, visual, and categorical modalities.

Multi-modal modeling. In recent years, authors [4, 15, 18] have been investigating solutions for multi-modal representation across textual and visual domains and achieved state-of-the-art results on research datasets. Many papers use early fusion across the two towers, which excludes the possibility of deploying the query tower model and document tower model independently. This paper focuses on two tower Siamese network with late fusion, where both towers can be computed independently in production.

Natural language processing. With the breakthrough of BERT [10] natural language understanding went on the next level of quality. Due to the high computational needs of transformers, various companies explored BERT optimizations, including [11, 21, 22]. We applied some of the similar approaches of [11] and discussed additional approaches we took to improve inference latency in production. Recently XLM-R (Conneau et al. [7]) adapted BERT approaches to multi-lingual applications. We use XLM-R to transform textual features and fine-tune using search logging data, we share our solutions in §3.

3 MODELING

In this section, we start by describing the model architecture and going over the input features. We then describe how we train and evaluate the model, including a curriculum training scheme where we feed harder negative examples to the model by using different loss functions without changes to the training data collection process. We also discuss the challenges we faced in multi-modal training and our solution to overcome them. We conclude this section by describing how we used ML interpretability techniques to understand the model and identify improvement areas.

3.1 Model architecture

The model architecture is depicted in Figure 2. For query tower, we use three input features - the character tri-gram [14] multi-categorical feature; the country of the searcher, a single categorical feature; and the raw query text, which is a textual feature. For character tri-grams, we apply a hash function (e.g. MurmurHash) to the sliding window of 3 characters of the query, thus getting a list of hash ids for the query. We then pass the hash ids list to an EmbeddingBag [2][24] which learns an embedding for each hash

[2] https://pytorch.org/docs/stable/generated/torch.nn.EmbeddingBag.html

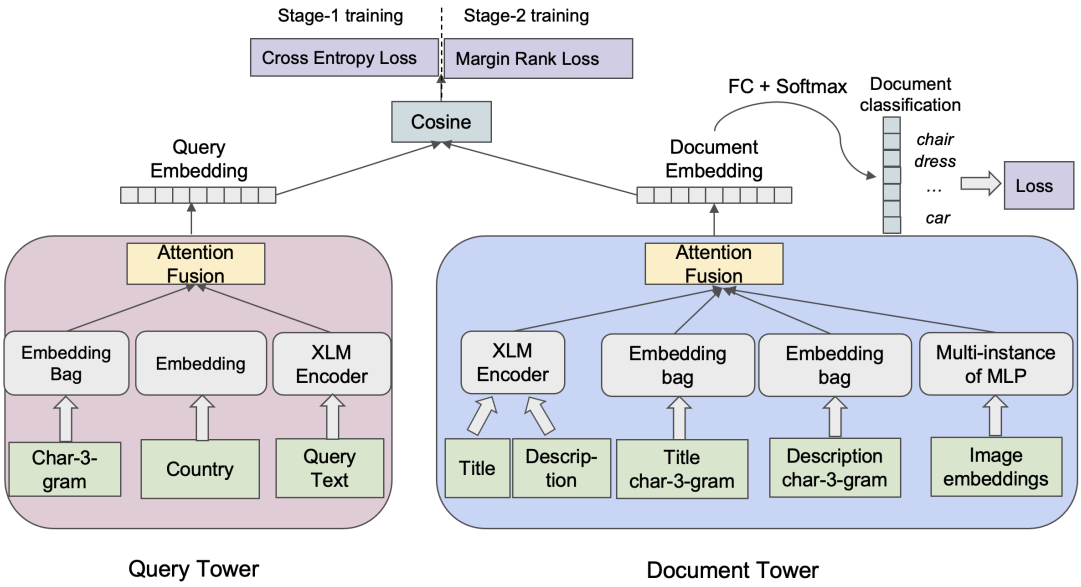

Figure 2: Architecture of *Que2Search*, Facebook's scalable query and product understanding model.

id and performs a sum pool to provide a fixed-size representation for each query. We also apply EmbeddingBag to learn country representation for searcher country single categorical feature. We apply a 2-layer XLM [17] encoder to the raw query text. We take the representation of the [CLS] token of the final layer as the encoder representation and project to the desired dimension to get the query's XLM representation. We then fuse these three representations based on attention weights to get the query's final representation.

For the document tower, we have input features like product title, description and images. We use one shared 6-layer XLM-R [7] encoder for all the textual fields (such as title and description); sharing of the encoder helped to achieve improvement in batch recall@1 (see table 3). We also apply EmbeddingBag [24] to encode the character tri-grams multi-categorical features of title and description respectively, as described in the previous paragraph. For each document, there are a variable number of images attached to it. We take the pre-trained image representations (Bell et al. [2]) for each of the attached images, apply a shared MLP layer and deep sets (Zaheer et al. [29]) fusion to get the image channel representation. Same as in the query tower, we then fuse the representations from different feature channels with learned attention weights to get the document's final representation.

We tried different late-fusion techniques to fuse the representations of different feature channels, like concatenation fusion (i.e., concatenate all encoder outputs followed by an MLP layer) and simple attention fusion (a weighted sum of channel representations based on learned channel weights as shown in Eq.1 where $\|$ means concatenation). Simple Attention fusion has shown better metrics performance as compared to concatenation fusion.

$$
\begin{aligned}
\varphi &= \{\varphi_i\}_{i=1}^{N} && \text{representations of N channels} \\
\Phi &= \varphi_1 \,\|\, \ldots \,\|\, \varphi_N && \text{concatenation} \\
a &= \text{Softmax}(\Phi W) && W \in \mathbb{R}^{ND \times N} \\
f &= \sum_{i=1}^{N} a_i \dot{\varphi}_i && \text{final tower representation}
\end{aligned}
\tag{1}
$$

We used PyTorch and several downstream libraries such as Facebook's PyText, Fairseq, and Multimodal Framework (MMF) to implement the model. We used a learning rate of $7e-4$ with a batch size of 768 and Adam optimizer. However, since we use pretrained XLM/XLM-R models, we implemented a different learning rate $2e-4$ to XLM module - this means that all the parameters inside the XLM module would have a different learning rate compared to the rest of the parameters. This gave us a 1% increase in ROC AUC to the validation set (referred as *varying XLM LR* in table 4). We used standard regularization techniques like dropout (dropout rate of 0.1) and gradient clipping of 1.0. We used ROC AUC metric on the validation set for early stopping with a patience of 3 epochs.

3.1.1 Two Tower with Classification. We also extend our two-tower model structure to have an additional classification task in document tower, as depicted in the upper right of figure 2. For each document, we collect a list of text queries estimated to be relevant for that document(defined in §3.2) from de-identified and aggregated Marketplace search log data. We keep the top 45k most common queries. We treat the problem as a multi-label multi-class classification task and use multi-label cross entropy, where each positive target is set to be 1/k if there are k positive labels for the document [19]. We compare the results of this multi-task architecture with the vanilla two-tower model in §5.

3.2 Training

Training data. In traditional supervised learning, we require the training data to contain both positive and negative examples. However, in our training setup, we take positive training (query, document) pairs only and automatically obtain the negative examples from the query to the other documents within a batch. Positive (query, document) pairs can be collected through search log. In particular, we use similar schema of data collection as in [2, 25]: we create a dataset of document-query pairs from Facebook Marketplace search log by filtering for the following sequence of events: (1) a user searches for a query, (2) clicks on a product,(3) messages a Marketplace seller, and (4) the seller responds back. If all 4 events happen within a short time, such as 24 hours, the user likely found what they were looking for, and thus the query is considered relevant to the document. We do not have access to message contents and only know that users interacted with the sellers. We call sequences of such user interactions as $in-conversation$ [25].

Batch Negatives: We have used several ways to generate negative examples. In order to explain how batch negatives work, we'll explain what happens during training in a single batch of batch size B. In every batch, we obtain the query embedding tensor $\{q_i\}_{i=1}^{B}$ with embedding dimension D, and similarly we also obtain the document embedding tensor $\{d_j\}_{j=1}^{B}$ of the same size. We then calculate a cosine similarity matrix $\{cos(q_i, d_j)\}_{i,j=1}^{B}$. The cosine similarity matrix represents query document similarity for every possible pair within a batch, with the row in the matrix belonging to a query and the column belonging to a document. We treat this as a multi-class classification problem, where the number of classes is B, and document d_i is the ground-truth class for query q_i (as shown in green grids in the figure 4) while the other documents $d_j, j \neq i$ are negative examples for q_i. We use a scaled multi-class cross-entropy loss (formulae 2) to optimize our network. The idea of having a scale times cosine is also mentioned in Deng et al. [9]. During training, we found that having a *scale* is important for loss to converge. In our use case, we choose scale between 15 and 20.

$$\text{loss}_i = -\log \frac{\exp(s \cdot \cos\{q_i, d_i\})}{\sum_{j=1}^{B} \exp(s \cdot \cos\{q_i, d_j\})} \quad \text{where s denotes scale} \quad (2)$$

We've also tried Symmetrical Scaled Cross Entropy Loss (formulae 3). This loss function optimizes query-to-document retrieval and also document-to-query retrieval. While this loss neither improves nor regresses the overall system metrics in our query to document two-tower model discussed in this paper, in a different use-case where we train document-to-document two-tower model, this symmetric loss has shown 2% improvement in ROC AUC on evaluation dataset.

$$-\frac{1}{2}(\log \frac{\exp(s \cdot \cos\{q_i, d_i\})}{\sum_j \exp(s \cdot \cos\{q_i, d_j\})} + \log \frac{\exp(s \cdot \cos\{q_i, d_i\})}{\sum_j \exp(s \cdot \cos\{q_j, d_i\})}) \quad (3)$$

3.3 Curriculum Training

In addition to the in-batch negative sampling described in §3.2, we designed a curriculum training scheme where the two-tower model is fed harder negative examples during 2nd stage training and results in an absolute over 1% ROC AUC gain on the validation set (see table 4). In the first stage of training, we train a model using in-batch negative examples and let the model converge by early stopping on ROC AUC metric on the validation set. We would like to feed "harder" negative examples to the model for the 2nd stage of training. While traditionally this would mean running a separate data pipeline to find hard negativs and feeding this dataset to the model, we propose to use negative examples within the batch: we get the cosine similarity matrix of shape $(B * B)$ as usual. For each row, we now extract the highest score besides the diagonal score as the hard negative sample; we denote the column index of this score as $n^{q_i} = argmax_{j \neq i}^{B} cos(q_i, d_j)$. This will help to generate training samples in format $(q_i, d_i, d_{n^{q_i}})$. We explored scaled binary cross entropy loss (formulae 4) and margin rank loss (formulae 5) with these triplet samples, and observed that margin rank loss with margin between 0.1 and 0.2 works best. In-batch hard negative sampling also reduces the additional CPU usage in finding hard negatives and the additional cost in maintaining another offline data pipeline.

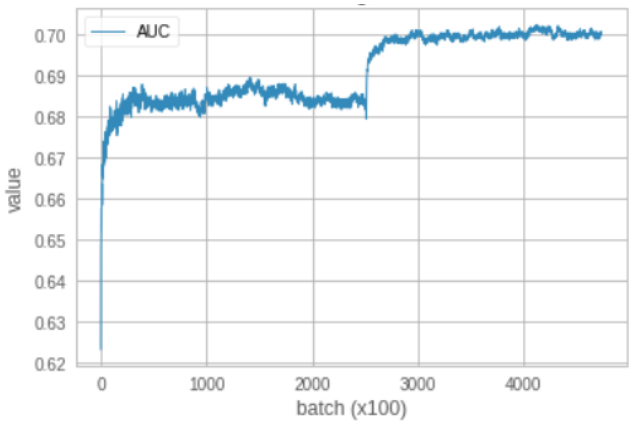

Figure 3: ROC-AUC curve for multi-stage training.

Initially, curriculum training with harder negatives did not work well. Through experiments, we found that it is crucial to make sure the first stage training converges before kicking off 2nd stage training. Additionally, we observed that $reduction = sum$ in $MarginRank$ loss[3] performs better than $reduction = mean$.

$$\text{loss}_i = -log(\sigma(s \cdot cos(q_i, d_i))) + log(1 - \sigma(s \cdot cos(q_i, d_{n^{q_i}}))) \quad (4)$$

$$\text{loss}_i = max(0, -[cos(q_i, d_i) - cos(q_i, d_{n^{q_i}})] + margin) \quad (5)$$

Figure 3 is ROC AUC evaluation metric curve of multi-stage training from one of our model training instances, we see a ROC AUC bump in 2nd stage training.

3.4 Evaluation

Before A/B tests through online traffic, we evaluate our model candidates offline and select the best one based on batch recall@K, ROC AUC, and KNN Recall and Precision.

Batch recall@K: This metric measures whether diagonal element $cos(q_i, d_i)$ is among the top K scores of the row $cos(q_i, d_j)$, $j \in [1, B]$. This metric is easy to compute during training and is the closest metric that the model optimizes, enabling us to iterate fast on modeling ideas.

[3] https://pytorch.org/docs/stable/generated/torch.nn.MarginRankingLoss.html

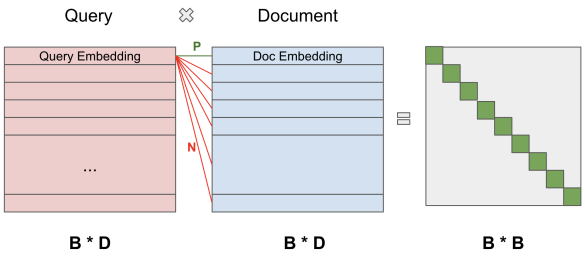

Figure 4: Generation of random negative examples.

ROC AUC: In Facebook Marketplace, we regularly collect human-rated data for the quality of search engine. Human-rated evaluation is done on publicly visibly products, and the query data is de-identified and aggregated before evaluation proceeds. Given the set of pre-selected search queries, numbers usually in thousands, we scrape search engine results and send them to human raters for evaluation: 1 for relevant document that satisfies the search intent of the query and 0 for the irrelevant. This gives us an evaluation dataset of (query, document) pair with a label. With cosine similarity of the inferenced query and document embeddings $cos(q, d)$ as score, we calculate ROC AUC. ROC AUC are used as a validation metric during training time for early stopping and an offline evaluation metric after model is trained. The human-rated data helped us evaluate potential impact on relevance and the search quality.

KNN Recall@K: On the other hand, we evaluate the potential impact on online engagement by KNN Recall and Precision. KNN Recall@K is an offline evaluation metric to calculate on the evaluation dataset after we trained the model. We trained our models using weeks of online engagement data ($in - conversation$ defined in §3.2), and evaluated on data of the future unseen days: given a trained two-tower model and an evaluation dataset, inference query embedding and document embedding. Given query embedding, we use Faiss (Johnson et al. [16]) K-nearest neighbor to retrieve N similar documents in the document embedding space and check whether the original query's ground-truth document is among the top K retrieved documents. We observe that KNN Recall@K metric is closely correlated to online performance for Embeddings-based Retrieval.

3.5 Speeding up model inference

At Facebook, our systems need to handle a large QPS at lightning speed to serve our large user base. Search queries are requested by the user in real-time and we expect to return the result within hundreds of milliseconds. The query side model needs to be fast enough to satisfy system requirements. Therefore, we experimented with various transformer architecture to handle the trade-off between latency and accuracy. We experimented with two XLM models in the query tower: one with 2 layers, 4 attention heads and 128 sentence embedding dimension and the other with 3 encoder layers, 4 attention heads and 256 sentence embedding dimension. We found the former to have a P99 latency of 1.5ms while the latter had a P99 latency of 3.5ms while the performance gain was minimal. We cap the query sequence length to be at the 99th percentile of all query lengths in Marketplace search. We used a dropout of 0.1 and a feed-forward embedding dimension of 3x of sentence embedding dimension. We used a SentencePiece vocabulary of size 150k for all of our experiments.

The document tower model has lesser requirement on execution speed, because document side embeddings can be pre-computed offline, or computed near real-time after document creation, and then ingested into the search system backend. We share more details on serving system optimizations in §4.

We also take advantage of Torch Just-In-Time (JIT) compilation to speed up our inference.

3.6 Fusion of different modalities

We experimented with various approaches to mix different modalities, for example text modality and image modality. The default approach is simple attention fusion as described in Eq.1. We refer to this approach as *baseline modalities handling*.

Gradient blending [26] is an optimization technique to balance among different modalities and reduce overfitting on dominant modalities in a multi-modal setting. In gradient blending, we train M+1 models: M uni-modal models and one full multi-modal model, where M is the number of modalities. We compute losses for each model and take a weighted sum of the losses with estimated weights as the final loss. The intuition behind this is to let the model learn even when the modality is not present, making the model robust to sparse input features.

We extend Gradient Blending to Two Tower architecture by keeping one tower with full modalities at a time. For example in Figure 2, we say the query tower has $m = 2$ modalities: query and country; and the document tower has $n = 3$ modalities: title, description and image. We then get $m + n + 1$ losses as formulated below, where $\{\varphi_i^1\}_1^m$ and $\{\varphi_j^2\}_1^n$ are uni-modal deep networks for the left tower and right tower respectively, and $\oplus$ denotes a fusion operation:

$$\begin{aligned} \mathcal{L}_{multi} &= L(cos(\varphi_1^1 \oplus ... \oplus \varphi_m^1, \varphi_1^2 \oplus ... \oplus \varphi_n^2)) \\ \mathcal{L}_i^1 &= L(cos(\varphi_i^1, \varphi_1^2 \oplus ... \oplus \varphi_n^2)) \qquad i \in [1, m] \\ \mathcal{L}_j^2 &= L(cos(cos(\varphi_1^1 \oplus ... \oplus \varphi_m^1, \varphi_j^2)) \qquad j \in [1, n] \end{aligned} \tag{6}$$

Modality is a loosely define term. We can treat each input feature channel as an individual modality. In that case, in Figure 2, the query tower has $m = 3$ modalities and the document tower has $n = 5$ modalities. We can also group related features together to be considered as one modality. For example, if product description feature is missing, then neither description raw text nor description character tri-grams is present. We can group the two features together. Then in the document tower, we only have $n = 3$ modalities: title, description and image. We share results of our experiments on gradient blending in Table 1. We did not observe statistically significant improvement in offline metrics by using gradient blending with individual feature channels, and observed that grouped feature Gradient blending provides a boost in performance by +0.91% in ROC AUC, Gradient blending is only applied during training, and during inference we run the full multi-modal model as usual.

3.7 Model Interpretability

3.7.1 Does XLM encoder add value to the query tower? One question that was commonly asked is whether it is useful to have XLM

Technique	ROC AUC % (relative)
Baseline modalities handling	-
GB - individual feature	-1.2% (non-significant)
GB - grouped feature	+0.91%

Table 1: Evaluation of modalities fusion on human rated dataset.

Tower	Feature	Tower Importance
Query	XLM	60%
Query	Char trigram	40%
Document	GrokNet	55%
Document	Description XLM	13%
Document	Title XLM	9%
Document	Description Char trigram	1.5%
Document	Title Char trigram	1.5%
Document	Language	1%
Document	Country	1%

Table 2: Feature Importance Calculation

encoder for the query tower. One may argue that queries usually are short phrases, and for Marketplace the queries are head-heavy (most of the traffic is dominated by a small set of queries). Intuitively character tri-gram may already be good enough to represent the queries.

We use the attention weights to explore the answer in this section. In the query tower, we have an attention mechanism between the query character tri-gram channel and the query XLM encoder channel. The attention weights (which sum to 1) are formulated in Eq.1. We extract these attention weights and try to interpret which feature the model is paying more attention to. We find that the XLM encoder had an average attention weight of 0.64 compared to 0.36 for the EmbeddingBagEncoder. We find an interesting observation when we plot the attention weight as a function of the query length. We find the model attends to character tri-gram more when the query length is less than 5 characters, but XLM encoder weight dominates for longer queries.

идея - в ретриве больше давать вес ембедному поиску когда запрос длинный, иначе более простому BM25

3.7.2 Feature Importance. We think the two-tower model is more of a "search" problem than a "classification" problem.

As a result, we propose a different way to compute feature importance for the two-tower model where we compute feature importance within a tower while keeping the other constant. We use a feature importance technique known as Feature Ablation - where each input feature is replaced by a baseline (a tensor of zeros, or feature value from other random training point) and the change in output is computed.

We present the feature importance results in Table 2. As we can see, XLM dominates among all textual features in both towers. On the other hand, the character tri-gram features also provide non-trivial gains. This is consistent with our study in Section 3.7 where the character tri-gram features contribute most for short queries. On the document tower, we use pretrained image embedding from

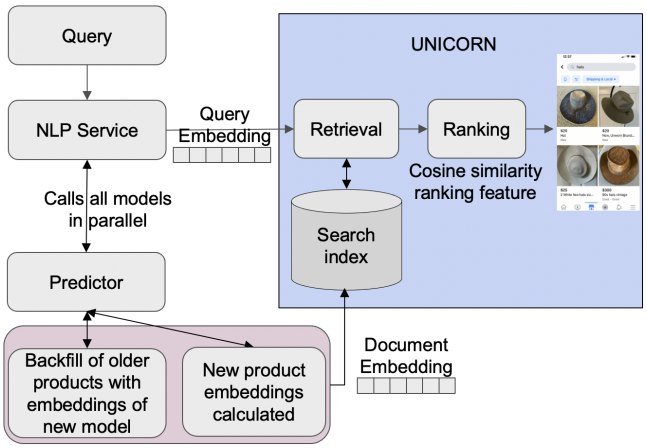

Figure 5: System architecture of *Que2Search*.

GrokNet [2], the state-of-the-art image recognition system at Facebook. From the Table 2 we see image modality provides significant value to the model.

Additionally we use deep sets [29] fusion over multiple images of the product, it provides additional benefits especially for products like *garage sale* where several products are getting sold within one listing.

4 SYSTEM ARCHITECTURE

We now describe the system architecture of deployment of *Que2Search*, Facebook's large-scale product understanding system for Marketplace. *Que2Search* is deployed in production and is designed to operate on products created in real-time. Model inferences are computed at cloud of machines called Predictor [12]. Predictor provides functionality to deploy the model, and API to call the model with a set of input features. Predictor is a cloud service and can scale accordingly. Given the call to Predictor, the query embedding is computed within 1.5 ms P99 and document embedding within 7ms. We store the product embeddings using Unicorn [8] along with other product indexes for search retrieval and ranking.

Document side model serving. Upon the creation and update of a product, an asynchronous call will be made to Predictor to generate its product embedding, and the embedding vector will be updated to search index in real-time to prepare the product for retrieval and ranking. In other words, product embeddings are already pre-computed and indexed when search queries are issued, which makes the computation of query product similarity across many product candidates tractable. In addition, Unicorn further clusters and quantizes [16] the embedding space to speed up the computations. Our infrastructure also supports backfills embeddings to older products. After a new model is deployed, it usually takes under 24 hours to backfill embedding to enough documents to the search index to be experimentation ready.

Query time model serving. At the inference time query flow follows the path of calling our backend Natural Language Understanding Service (NLP service), which calls to Predictor service in real-time. NLP service caches the embeddings in memory for a fixed period of time (e.g. 8 hours) to speed up the service response and save on the Predictor computations. Query embeddings is then

Technique	Batch Recall@1
Wide & dense char trigram baseline [25]	0.6788
Char tri-gram + raw text features	0.7292
Char tri-gram + image features	0.7237
Char trigram + raw text + image	0.7445
All features + Attention Fusion on both towers	0.7456
Above + Shared XLM for title & description	0.7459

Table 3: Ablation study of features and parameters. Each modality brings it's separate gain independently. Char trigram continue to be useful.

passed to Unicorn search engine backend for product retrieval. Unicorn search engine is capable to do approximate nearest neighbor (ANN) retrieval with boolean constraints, which means that we can, for example, require that we retrieve most relevant/matching products for the query, and at the same time require that products are within 50 miles radius of the user specified location, and along with other traditional term matching expression. Results are then passed back to several stages of ranking. We use the cosine similarity score produced by the two-tower model as a ranking feature in all stages of ranking.

5 ABLATION STUDIES

We performed offline ablation studies to confirm the efficiency of every step. Our baseline model [14] uses deep & wide architecture and applies character tri-gram features on the textual attributes of the products; the model has been previously shipped to production and achieved production gains. In table 3 we share results of the ablation studies comparing new model and features additions. We see that adding XLM/XLM-R based raw text feature transformers significantly improves batch Recall@1(§3.4).[4] Similar impact comes from image features, where we apply deep sets fusion [29] over up to five images of the product. Text transformers and image features each provide independent value; jointly introduction of text transformers and image features gives a relative improvement of +9.23%. Simple attention fusion (Eq.1) over tower modalities, as compared to concatenation fusion, gives 0.15% improvement; and having a shared XLM-R encoder across text fields of the product provides additional improvement of 0.18% (see Table 3).

Que2Search two-tower model also substantially improved ROC AUC on human-rated datasets. Results of evaluation are shown in Table 4. Both image and textual features provide additional value, where XLM encoder on raw text contributed most gains, with 2.1% absolute improvement in ROC AUC. Second biggest contributor comes from curriculum training with boost in ROC AUC by 0.8% absolute (§3.3). A different learning rate for XLM modules from the rest of the model (§3.1) helps to improve the quality further with 1% relative improvement. Addition of classification task head in the document tower (§3.1) provides 0.61% relative improvement in ROC AUC. Gradient blending with grouped features (§3.7) makes the model more robust for cases when one of the modalities is not present in the product description and gives an additional relative improvement of 0.91% in ROC AUC. In summary, *Que2Search* achieved over 5% absolute ROC AUC improvement over the existing Facebook product understanding system.

[4]We used batch Recall@1 in the initial stages of development and report details in Table 3. In subsequent studies, we reported ROC AUC improvements on human-rated dataset.

Technique	ROC AUC
Wide & dense char trigram baseline [25]	0.795
Char trigram + raw text features	0.816
Char trigram + image features	0.806
Char trigram + raw text + image	0.825
Above + Curriculum Training	0.833
Above + varying XLM LR	0.837
Above + Classification head	0.842
Above + GB-grouped feature	0.849

Table 4: Evaluation on human rated dataset. Each modality brings it's separate gain independently. Attention fusion and sharing of weight included.

Technique	In-conversation
(Baseline) No Embedding-based Ranking	-
(EBR) char tri-gram only model [25]	+3.3%
(EBR) *Que2Search* model	+2.80%
(Ranking) Cosine similarity ranking feature	+1.24%

Table 5: Online A/B testing results for incremental engagement gains from different techniques. Relative improvement from each technique assumes the productionalization of the techniques above. Model uses the techniques mentioned in Table 4.

6 DEPLOYMENT LESSONS

We deployed *Que2Search* on Facebook Marketplace powering hundreds of millions of product searches per day. Furthermore, we achieved this scalably by integrating it as an organic component of Facebook Marketplace's Embedding-based Retrieval and ranking stacks, via various optimizations in precision, recall, CPU usage and latency. Online A/B testing showed that *Que2Search* led to more than 4% increase in search product engagements, with around 2.8% coming semantic retrieval, and 1.24% from ranking. Table 5 shows a more detailed breakdown of how different techniques incrementally contribute to search product engagement gains.

6.1 Embedding-based retrieval

With the introduction of *Que2Search*, we are able to perform more semantic retrieval using Unicorn's native support for embedding-based retrieval (EBR) [14]. More specifically, as described in section 4, we deployed the query tower model to a NLP service to generate query embeddings in real-time, and deployed the product tower model on an async update path to Unicorn index upon the creation and update of a product. When a user issues a product search, its search query will be translated into an embedding vector to perform an ANN search in the product embedding space using Unicorn's NN-operator, and products within a threshold of cosine distance will be retrieved as a result.

6.2 There is no silver bullet for retrieval

Note that we deployed *Que2Search* as an organic component of the whole Marketplace search retrieval, instead of using it as the only mean of search retrieval. Other key components of the whole retrieval process include a traditional token-based retrieval that is semantically parallel to the ANN search, and other possible constraints such as location, social [13], and product category. And this is done seamlessly using Unicorn's query language [8] (i.e. ANN and token-based retrieval are organized under an OR, and constraints are grouped into a high-level AND).

Empirically, we discovered that it is beneficial to treat EBR as a component to solve semantic matches, instead of seeing it as a silver bullet for all retrieval problems. Here are a few reasons (1) despite that *Que2Search* is location-aware, it is difficult to specify explicit location constraints in ANN search, which is crucial to a local commerce search engine like in Facebook Marketplace (2) ANN search is unable to handle user-selected filters which is a key feature in any commerce search engine (3) due to performance concerns, it is hard to make ANN search exhaustive, while token-based retrieval offers a cheap alternative to cover easier cases more exhaustively.

6.3 Precision, recall, CPU usage and latency

When deploying *Que2Search*, precision, recall, CPU usage and latency are the dimensions whose trade-offs need to be optimized. In terms of Unicorn's NN-operator, they can be translated into two hyper-parameters - *n_prob* and *radius*. Since Unicorn takes a clustering and product quantization approach [14, 16], the number of clusters to be scanned determines how exhaustive the ANN search can be, which is controlled by *n_prob*. Generally, more clusters scanned means higher precision but higher CPU usage. And when a product is scanned, its cosine distance to the query vector is calculated. Then *radius* parameter controls the threshold to determine whether to retrieve this product or not.

Although a grid search of the hyper-parameters is possible using online A/B testing, we want to minimize user impacts especially when poorly chosen parameters degrade user experiences. Therefore, we simulated the retrieval on a golden query set to reduce into a handful of reasonably good candidates before setting up online A/B testing. The golden set consists of search results scraped against Marketplace search using stratified sampling of search queries. We also asked raters to judge whether a result is relevant or not. With this golden set, we can simulate retrieval using different *radius* and determine the aggressiveness of retrieval by examining the number of results returned, and the recall of the relevant ones. However, online A/B testing is still essential to choose the right parameters, because apart from the need to measure the real user engagements, sometimes CPU usage and latency are difficult to be simulated accurately due to their close connections to user behaviors.

6.4 Optimizing for continuous pagination

One key observation on how user behaviors can affect CPU usage measures is around continuous pagination, which refers to users keep scrolling down the search result page to trigger more retrieval requests so that more products can be surfaced. Obviously, if a user is satisfied with the top results (equivalently, the first retrieval request offers great precision), we will save CPU usage because any subsequent requests will not need to be triggered. However, we also know empirically that users are sensitive to the latency of the first request - if no results are displayed in sometime, the entire search could be abandoned. This means we cannot make the first request too expensive in favor of precision. To summarize, continuous pagination could be used as a lever to trade off precision, recall and CPU usage, but subject to constraints like latency.

With that observation in mind, we enabled EBR with a conservative *n_prob* and *radius* setup for the first request, so that we can keep a high bar on latency, and made the EBR in subsequent requests more expensive to optimize for engagements further. Based on online A/B testing results, we discovered that this dynamic ANN configuration yielded the same engagement wins compared to keeping the same configuration throughout, but with a substantial reduction in both CPU usage cycles and latency. In fact, without this optimization, we would not able to enable *Que2Search* in production due to latency concerns.

6.5 Search Ranking

Based on table 4, we know that the query-product cosine similarity score from two-tower model is a good predictor of the human-rated dataset. Provided that the similarity score is readily available from retrieval stage, we also exposed the score to search ranking to leverage *Que2Search* further.

Facebook Marketplace's search ranking follows a two-stage scheme, where a lightweight GBDT ranker runs on individual index server instances to select the top results from each shard, and a more expensive DLRM-like [20] model is used to further select the best results. To utilize *Que2Search* in ranking, we simply added the score as a ranking feature to both stages. We examined feature importance and discovered that the cosine distance feature is of top one importance in both stages, and online A/B testing showed that such simple feature addition yielded statistically significant engagement improvements as illustrated in table 5.

6.6 Lessons from failures

Deploying *Que2Search* for better search quality is a trial-and-error process, and we encountered many phenomena that could seem counter-intuitive at first sight.

Precision matters:. Precision is obviously very important in various machine learning tasks, but it is less intuitive why we cannot use recall as the only criteria in a search retrieval setting. In other words, we discovered that relaxing the threshold of ANN search could lead to worse results although theoretically, it should not affect the total number of good results being retrieved. Further analysis showed that this is caused by the inconsistency between the retrieval and ranking models - our ranking model is not able to handle the noisier results retrieved via a more relaxed threshold. Improving the consistency between multi-stage models is a research topic by itself [5, 28] which we will not expand in details. And eventually, we simply ran online and offline parameter sweeps to decide the best threshold.

Relevance is not enough:. A naive approach to improve the consistency between retrieval and ranking model is to simply use the two

tower model's cosine similarity as the ranking score. A nice consequence of this approach is that it guarantees all results retrieved due to a more relaxed threshold can be ranked at the bottom. In fact we experimented with this idea by replacing our GBDT-based stage one model with the cosine similarity, but although we discovered a significant relevance improvement measured by NDCG, it regressed online engagement considerably. This is possibly because the two-tower model is trained to optimize query-product similarity instead of optimizing engagement, while the GBDT model is more engagement focused. Although it is possible to add engagement features to the two-tower model, it is not a trivial extension because many of our important engagement features are hand-tuned dense features based on both searcher and product signals, which violate the independence/late-fusion property of a two-tower model.

7 CONCLUSION

We presented approaches for building a comprehensive query and product understanding system called *Que2Search*. We presented innovative ideas on multi-task and multi-modal training to learn query and product representations. Through *Que2Search* we achieved over 5% absolute offline relevance improvement and over 4% online engagement gain over state-of-the-art Facebook product understanding system. We shared our experience on tuning and deploying BERT based query understanding model for search use-cases and achieve 1.5 ms inference time at 99th percentile. We shared our deployment story, and practical advice on deployments steps, and how to integrate *Que2Search* component in search semantic retrieval and ranking stacks.

8 ACKNOWLEDGEMENTS

The authors would like to thank Shaoliang Nie, Liang Tan, Rao Bayyana, Hamed Firooz, ZJ Yin, Ashish Gandhe, Yinzhe Yu and others who contributed, supported and collaborated with us.

REFERENCES

[1] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. 2011. *Modern Information Retrieval: The Concepts and Technology behind Search.*
[2] Sean Bell, Yiqun Liu, Sami Alsheikh, Yina Tang, Edward Pizzi, M. Henning, Karun Singh, Omkar Parkhi, and Fedor Borisyuk. 2020. GrokNet: Unified Computer Vision Model Trunk and Embeddings For Commerce. In *KDD*.
[3] Jane Bromley, Isabelle Guyon, Yann Lecun, Eduard Säckinger, and Roopak Shah. 1994. Signature Verification using a "Siamese" Time Delay Neural Network. In *NeurIPS*.
[4] Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. 2020. UNITER: UNiversal Image-TExt Representation Learning. In *ECCV*.
[5] Zhihong Chen, Rong Xiao, Chenliang Li, Gangfeng Ye, Haochuan Sun, and Hongbo Deng. 2020. ESAM: Discriminative Domain Adaptation with Non-Displayed Items to Improve Long-Tail Performance. *arXiv preprint arXiv:2005.10545* (2020).
[6] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide Deep Learning for Recommender Systems. In *RecSys*.
[7] Alexis Conneau, Kartikay Khandel wal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised Cross-lingual Representation Learning at Scale. In *ACL*.
[8] Michael Curtiss, Iain Becker, Tudor Bosman, Sergey Doroshenko, Lucian Grijincu, Tom Jackson, Sandhya Kunnatur, Soren Lassen, Philip Pronin, Sriram Sankar, Guanghao Shen, Gintaras Woss, Chao Yang, and Ning Zhang. 2013. Unicorn: A System for Searching the Social Graph. *VLDB* (2013).
[9] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. 2019. ArcFace: Additive Angular Margin Loss for Deep Face Recognition. arXiv:cs.CV/1801.07698
[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *ACL*.
[11] Weiwei Guo, Xiaowei Liu, Sida Wang, Huiji Gao, Ananth Sankar, Zimeng Yang, Qi Guo, Liang Zhang, Bo Long, Bee-Chung Chen, and Deepak Agarwal. 2020. DeText: A Deep Text Ranking Framework with BERT. In *CIKM*.
[12] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang. 2018. Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. In *HPCA*.
[13] Yunzhong He, Wenyuan Li, Liang-Wei Chen, Gabriel Forgues, Xunlong Gui, Sui Liang, and Bo Hou. 2020. A Social Search Model for Large Scale Social Networks. arXiv:cs.IR/2005.04356
[14] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-Based Retrieval in Facebook Search. In *KDD*.
[15] Zhicheng Huang, Zhaoyang Zeng, Bei Liu, Dongmei Fu, and Jianlong Fu. 2020. Pixel-BERT: Aligning Image Pixels with Text by Deep Multi-Modal Transformers. *CoRR* (2020).
[16] J. Johnson, M. Douze, and H. Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* (2019).
[17] Guillaume Lample and Alexis Conneau. 2019. Cross-lingual Language Model Pretraining. arXiv:cs.CL/1901.07291
[18] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. 2019. ViLBERT: Pretraining Task-Agnostic Visiolinguistic Representations for Vision-and-Language Tasks. In *NeurIPS*.
[19] Dhruv Mahajan, Ross B. Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. 2018. Exploring the Limits of Weakly Supervised Pretraining. In *ECCV*.
[20] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. 2019. Deep Learning Recommendation Model for Personalization and Recommendation Systems. arXiv:cs.IR/1906.00091
[21] Pandu Nayak. 2019. Understanding searches better than ever before. https://blog.google/products/search/search-language-understanding-bert/
[22] Emma Ning. 2019. Microsoft open sources breakthrough optimizations for transformer inference on GPU and CPU. https://cloudblogs.microsoft.com/opensource/2020/01/21/microsoft-onnx-open-source-optimizations-transformer-inference-gpu-cpu/
[23] Xichuan Niu, Bofang Li, Chenliang Li, Rong Xiao, Haochuan Sun, Hongbo Deng, and Zhenzhong Chen. 2020. A Dual Heterogeneous Graph Attention Network to Improve Long-Tail Performance for Shop Search in E-Commerce. In *KDD*.
[24] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).
[25] Yina Tang, Fedor Borisyuk, Siddarth Malreddy, Yixuan Li, Yiqun Liu, and Sergey Kirshner. 2019. MSURU: Large Scale E-commerce Image Classification with Weakly Supervised Search Data. In *KDD*.
[26] Weiyao Wang, Du Tran, and Matt Feiszli. 2019. What Makes Training Multi-Modal Networks Hard? *CVPR* (2019).
[27] Ji Yang, Xinyang Yi, Derek Zhiyuan Cheng, Lichan Hong, Yang Li, Simon Xiaoming Wang, Taibai Xu, and Ed H. Chi. 2020. Mixed Negative Sampling for Learning Two-Tower Neural Networks in Recommendations. In *WWW*.
[28] Bowen Yuan, Jui-Yang Hsia, Meng-Yuan Yang, Hong Zhu, Chih-Yao Chang, Zhenhua Dong, and Chih-Jen Lin. 2019. Improving ad click prediction by considering non-displayed events. In *KDD*.
[29] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. 2017. Deep Sets. *CoRR* abs/1703.06114 (2017). arXiv:1703.06114 http://arxiv.org/abs/1703.06114