DeCodR: Using Dense Representations for Code and Repository Search

Julian Killingback
jkillingback@cs.umass.edu

Chris Samarinas
csamarinas@umass.edu

Nandyala Siddharth Kantu
nkantu@umass.edu

Siddharth Nammara Kalyana Raman
snammarakaly@umass.edu

1 Introduction

Finding answers to coding questions is a vital part of writing code and developing software. Current commercial search engines only allow for keyword based queries, making it hard to find answers for more complex questions where necessary keywords may not be known by the user. For example, often users know what they want their transformed data to look like, but they do not know what keywords to use when searching. Using a long form question, they can write out the original data and the data after a transformation and use that to find answers. Similarly, finding related open-source code repositories on Github can be hard with the existing exact match search interface. For example a query like *python libraries related to image enhancement*, would not return sufficient results using the existing term-matching search, because many repositories can mention other related terms such as *super-resolution*. Helping people find answers to coding related questions using natural language, or relevant code repositories using concept-based search, could reduce the effort and time spent by engineers on finding existing solutions and implementations for their problems.

For this project, we developed models to support two information seeking tasks: (1) *retrieval* and (2) *concept-based search* of repositories on Github. We experimented with a variety of retrieval models for these tasks, namely dense retrieval, re-ranking and doc2vec. Based on our early experimental results, dense retrieval using fine-tuned BERT-based models seem very effective for code and repository search. For the second task, doc2vec returns better results in lower cutoffs, in the top 5 and top 10 results.

Compared to our original proposal, we implemented all the main approaches related to retrieval and re-ranking, we compiled 2 datasets for code and repository search from Stack Overflow and Github, and we made a comparison with BM25 sparse retrieval. The tasks that we didn't have enough time and resources to accomplish are related to extractive QA for code search, and using autoencoders for repository search. Even though we implemented a re-ranking model, we only had time to experiment with it on the smaller-scale repository search dataset.

2 Related work

2.1 Code Search

Existing works have studied the problem of turning natural text descriptions, queries, and questions into a relevant piece of code. These systems operate in two distinct ways (1) using search to find related code in a large corpus of existing code snippets or (2) using a generative model to output the code directly. The majority of code snippet search systems have relied historically on term matching models (Lu et al., 2015; Nie et al., 2016; Hill et al., 2011) and have used additional augmentation to improve results. This includes expanding queries with synonyms using wordnet (Lu et al., 2015), using relevancy feedback to broaden query terms (Nie et al., 2016), and considering the position of terms in the query (Hill et al., 2011). More recent approaches have tried to address the lexical mismatch between natural language queries and code by using learned word embeddings (Ye et al., 2016) and jointly learning to embed code snippets and natural language text in a shared embedding space (Gu et al., 2018). Gotmare et al. (2021) used a two-stage architecture with a fast dense retriever and slow re-ranker. A similar architecture to this will be explored in this project. Code snippet search, although similar to our approach, dif-

fers from our proposal. It generally takes short form queries as input and returns only snippets of code. In our case, we accept long-form natural language questions (i.e. similar to those asked on Stack Overflow) in addition to shorter keyword queries and return an answer that can include code and accompanying explanations.

Unlike code snippet search, programming solution search seeks to find answers to coding questions which can be in the form of keyword queries or natural questions. A recent approach uses a mix of features including TF-IDF, word embeddings, and sentence embeddings to alleviate the problem of vocabulary mismatch in programming solution queries and answers (Silva et al., 2021). We attempted to use more sophisticated contextualized representations which can perform better given their superior performance in other domains. A related paper, Repo4QA (Anonymous, 2021), uses dense retrieval similar to our proposed method, but instead of answers it returns relevant GitHub repositories for Stack Overflow questions. This paper is related to both our tasks with the difference being we don't provide repositories to code questions and instead we provide answers with code and text, while we return repositories related to given concepts or categories.

2.2 Passage Retrieval for QA

Open-domain question answering (QA) is a task that answers factoid questions using large collection of documents. Retrieval in QA was earlier implemented using TF-IDF or BM25 (Robertson and Zaragoza, 2009), which matches keywords efficiently with an inverted index and can be seen as representing the question and context in high dimensional, sparse vectors (with weighting). Dense retrieval models did not outperform TF-IDF/BM25 until ORQA (Lee et al., 2019). In the paper, (Karpukhin et al., 2020), we find that the embeddings are optimized for maximizing inner products of the questions and relevant passage vectors, with an objective comparing all pairs of questions and passages in a batch.

2.3 Passage Re-ranking

A common approach for improving the precision of the top returned results of a retrieval model, is to use a re-ranking model. Models such as dual (dense) encoders, even though they perform better than traditional term matching approaches such as BM25, in many cases they fail to capture well the semantics of the queries and passages. The reason is that dual encoder architectures do not attend jointly on both the query and document at the same time, limiting their expressiveness. Transformer-based language models such as BERT, have been proven very effective for this task, surpassing all the previously proposed ranking models (Nogueira and Cho, 2019). Nogueira et al. (2019), in their subsequent work, tried to further improve ranking performance by introducing a multi-stage framework with BERT-based pointwise and pairwise ranking models. Pairwise ranking models can theoretically perform better than pointwise, because the nature of ranking is closer to pairwise comparisons. Xu et al. (2019) introduced a weak-supervision framework using unsupervised ranking functions and semantic feature similarities to improve the performance of BERT-based rankers. Ren et al. (2021) proposed a joint training method for dense retrieval and re-ranking. Leonhardt et al. (2021) showed that sentence-level representations can lead to better results in both first-stage retrieval and re-ranking compared to the commonly used document level representations.

3 Data

3.1 Stack Overflow Dataset

For code answer retrieval and re-ranking we used Stack Overflow data. The data was taken from SOTorrent (Baltes et al., 2018) which holds an archive of Stack Overflow posts. SOTorrent is conveniently located on Google's Big Query platform. We used the Big Query API to filter Stack Overflow questions that had the tags Python and a few other data science Python library tags such as PyTorch and Pandas. We tried not to pick Python questions about web development and thus excluded posts with tags such as Django and Jinja. Our rationale was that we wanted to limit the number of posts and figured staying within a certain domain would work better than sampling from a broad set of topics. We collected the following information for each question: question ID, question title, question tags, question body, question score, question views, and accepted answer ID. For each question we collected all the answers with the following information: answer body, answer score, answer ID, and the question ID the answer was attached to. With this information, we were able to create a queries file which had two fields query ID and query text. For our queries, we used both

the Stack Overflow's question body and question title. This provides us both a simulated keyword query from the title and a longer natural language question with the question body. We also created a collection file with answer IDs and answer text and a qrels file which maps the query IDs to relevant answer IDs.

After running our baselines we realized that there might be some bias in using the answers for a specific question as the gold answers. The reason is that a lot of answers will reuse parts of the question whether this is directly quoting from parts of the question, reusing variable or function names, or otherwise using similar language. This property of answers means that simply looking for exact matches in the text may give very good results when only searching for the provided answer for the question. To remedy this we needed questions and answers without overlapping terminology. We were able to do so using Stack Overflow questions that are marked as duplicates. When this happens, a link to an original question that is a duplicate is provided as well. Using Stack Overflow's data explorer tool we mined these duplicate questions and were able to collect 8,149 duplicate questions where the original that the duplicate pointed to was in our dataset. We could easily map relevant documents for these duplicate questions as we just had to provide the relevant documents for the original questions.

A challenging aspect of this data is that the questions and answers can both be very long. They also tend to have a lot of white space formatting such as tabs to show code indents. These two things make it hard to use traditional large neural language models effectively. Another difficult aspect is the task itself, even as a human it is non-trivial to decide whether a question is answered effectively in a passage, especially when there are different terms and symbols used. We were able to collect 3,136,186 queries from Stack Overflow data explorer tool. We have used a corpus of with 2,273,845 answers. Among the queries extracted we used 20,000 queries to run inferences on model errors. We used about 20,000 queries for testing and 4,000 for validation to check model convergence.

3.2 Github Dataset

For the repository search task, we constructed a new dataset using a public dump of Paperswithcode[1]. Paperswithcode is a curated database of Machine Learning papers with their corresponding open-source Github repositories. We used the topic annotations from each paper as groundtruth queries, and all the assosiated Github repositories as documents. In addition, we retrieved the relevant tags from the repositories on Github and added them as additional queries, and for the document content, we included the repository name, the description and the readme file text.

The final dataset, has a corpus of 136,155 documents and 4578 unique queries. We split the lists of relevant documents for each query in three subsets, 80% for training, 10% for validation and 10% for testing. The training set consists of 316,424 positive query-document pairs. The validation and testing sets have 39,448 and 41,955 pairs respectively. Figure 1 shows the frequency distribution of all the queries in this dataset in logarithmic scale. As we can see, there is a very big imbalance in the frequency of the queries. The most common query has frequency 9209 while the least common 10. This poses a challenge, because this means that some queries appear much more frequently than others in the training pairs. Training a model on such a skewed dataset could result in poor performance for low frequency queries.

Another observation after carefully examining the generated dataset, is that many queries are actually redundant or very related to each other. For example we can find overlapping queries such as *'general reinforcement learning'*, *'safe reinforcement learning'* and *'reinforcement learning algorithms'*. A model should be able to learn similar representations for these queries, thus a repository classification model would not perform well for instance. For this purpose, we chose to experiment with a BERT-based dense retrieval approach that can capture the latent semantic content of queries and repositories more effectively.

3.3 Data Preprocessing

In the Stack Overflow Dataset, the bodies of the posts, both questions and answers, were raw HTML. We parsed and removed all tags except the `<code>` tag as this wraps pieces of code and we assumed it might be a useful tag for the model. We split the queries into a training, validation, and testing set. We didn't need to create a split collection file as the documents being searched will stay

[1] https://paperswithcode.com/

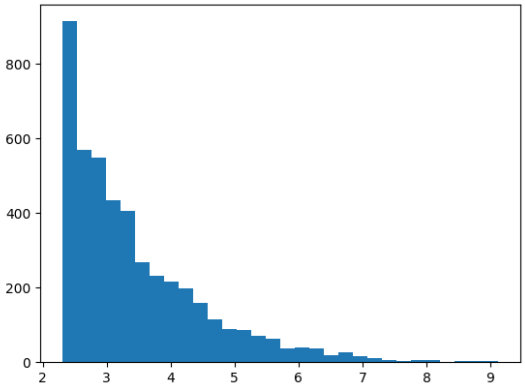


Figure 1: Frequency distribution of the queries/topics on the Github dataset in logarithmic scale using 30 bins.

the same in a real world scenario. However, we did make separate qrels files. While not strictly necessary, we did make a new collection file for the validation set that was smaller than the full validation set so that we could evaluate more quickly. To construct the validation collection, we searched for the queries using BM25 and added the top 100 documents to the collection along with the known relevant answers. This way, each correct answer would have at least some similar answers to confuse the retrieval model.

For the repository search dataset, we used the official Github API to retrieve the descriptions and tags. For the content of the readme files, we wrote a simple script that downloaded all the readme files. The readme files on Github have some special markup and often HTML code, so we used the markdown[2] and Beautifulsoup[3] and libraries to get the clean text without any formatting.

4 Approach

Task formulation Let us assume we have a large corpus of code-related answer passages (eg. from Stackoverflow) $\mathcal{C} = \{d_1, d_2, ..., d_m\}$. A retriever $R : (q, C) \rightarrow C_F$ is a function that takes as input a question q and a corpus C and returns a much smaller filter set of texts $C_F \subset C$, where $|CF| = k << |C|$. To accomplish this, we must learn a similarity function between questions and answers, so that similar (i.e. relevant) answers are marked similar to their corresponding questions. To make this process computationally feasible we define similarity as $\text{sim}(q, d) = E_q(q) \cdot E_d(d)$ where E_q and E_d are embedding functions. In essence, the inner product between a query embedding and answer passage embedding is the similarity between the question and answer. Using this formulation allows the passage embeddings to be precomputed prior to a question being asked, it also allows for fast retrieval using approximate nearest neighbor (ANN) search systems that allow for accurate and sub-linear ANN search. The returned result set can be re-ranked by a second-stage less-efficient re-ranking model.

The task formulation is the same for repository search, with the only difference being having keyword queries instead of natural language queries and multi-fielded documents with repository metadata (name, description and readme text).

4.1 Code Language Model Pretraining

Prior to training on our primary tasks, we thought it would be useful to pretrain a model for our code search task. We decided to fine tune a DeBERTa v3 (He et al., 2021a) base model which is a slight improvement on the DeBERTa v1 (He et al., 2021b) model that introduced disentangled attention. We picked DeBERTa for two reasons (1) it outperforms similar models such as BERT and RoBERTa (Devlin et al., 2019; Liu et al., 2019) on the GLUE benchmark (Wang et al., 2019) (2) starting in DeBERTa v2 it uses SentencePiece (Kudo and Richardson, 2018) embeddings which consider white spaces, which we thought would be important for understanding code where white space is often used to delineate different sections.

We started by generating our own vocabulary from Stack Overflow questions about Python using the Byte Pair Encoding method for training SentencePiece tokenizers. As there was significant overlap with the pretrained DeBERTa vocabulary, we simply extended the DeBERTa vocabulary with the new tokens. With a more relevant vocabulary, we began training our DeBERTa model on the same Stack Overflow questions we used to train the tokenizer, using a standard masked language modeling (MLM) pretraining task. For further details on the MLM training objective we refer readers to the original BERT (Devlin et al., 2019) paper. As we knew that both the Stack Overflow questions and answers can regularly exceed 512 tokens we increased the maximum length to 1024.

[2]https://python-markdown.github.io/
[3]https://www.crummy.com/software/BeautifulSoup/

4.2 Dense Retrieval

Based on the definition of relevance as the dot product of two embedded representations $E_q(q), E_d(d)$ for queries and documents, we fine-tune the parameters of a pretrained transformer language model by minimizing the following cross-entropy loss:

$$\mathcal{L}(q_i, d_i^+, \{d_{i,j}^-\}_{j=1}^{B-1}) = -\log \frac{e^{sim(q_i, d_i^+)}}{e^{sim(q_i, d_i^+)} + \sum_{j=1}^{B-1} e^{sim(q_i, d_{i,j}^-)}}$$

where d_i^+ is a positive document relevant to the query q_i and $d_{i,j}^-$ a negative example. For the embedding functions $E_q(q), E_d(d)$ we use the `[CLS]` token of a single transformer language model due to limited resources. During evaluation and inference, we use FAISS[4] to index the embedded representations of documents for efficient ANN search.

4.3 Re-ranking

For this project, we also implemented a custom BERT-based re-ranking model in an attempt to further improve the ranking of the results from a first-stage retriever. Our BERT-based re-ranking model takes as input a concatenation of a query and k document fields f_i with a special separation token, and tries to estimate a real-valued relevance score $\phi(q, d)$ by applying a linear transformation to the `[CLS]` token embedding. The model learns this score by minimizing the following pairwise cross inspired by RankNet (Burges et al., 2005):

$$\mathcal{L}(q, d_+, D_-) = -\sum_{d_- \in D-} log\left(1 + e^{-(\phi(q,d_+) - \phi(q,d_-))}\right) \quad (1)$$

$$\phi(q, d) = W \times \text{BERT}_{[\text{CLS}]}(q; f_1; ...f_k) + b \quad (2)$$

where ; represents concatenation using the special separation token. We prepare training mini-batches by sampling a number of positive examples d_+ and a set of negative examples D_-.

4.4 Experiments

4.4.1 Baselines

For both datasets, we used BM25 as a baseline, a commonly used lexical matching model which returns passages based on the term overlap with the query text. The weight for each term varies depending on the frequency of the term in the corpus. Additionally, the length of the document is considered as to penalize longer documents that would likely have more opportunity for term overlap. We used the standard implementation of BM25 included in (Yang et al., 2017) with parameters $k_1 = 0.9, b = 0.4$ using a Porter stemmer. In addition, for some experiments we used a more efficient Rust-based implementation from the tantity[5] library, which also has Python bindings.

For the repository search task, we also evaluated a doc2vec baseline, because we thought it could perform well given the nature of queries, which are treated as topics in this case rather than natural language queries. Doc2vec is known to perform well for some topic modeling and clustering tasks. We use the ngrams of the repository metadata to learn embeddings (paragraph vectors) for each repository and query/topic. We use the skip-gram variant of doc2vec, that tries to predict the words around a text window given one excluded word from that window. We initialize paragraph vectors for both documents and queries.

4.4.2 Code Search

For the code search task we performed experiments only for first-stage dense retrieval without re-ranking. We initially tried to use the pretrained DeBERTa model we discussed in Section 4.1, but quickly realized that the large embedding matrix and overall model size made it nearly impossible to train. Next we tried DeBERTa v3 extra small, but got poor validation results. We believe this was caused by a lack of hard negative in the batch which was particularly acute, because we did not provide selected negatives and also had a relatively small batch size. This meant that the negative passages for a given query are only the positive passages for the other queries in the batch, as the average answer is pretty unrelated this led to poor training quality and poor results. Given this difficulty, we decided to increase the batch size using gradient caching provided in the Tevatron (Gao et al., 2022) library. We also decided that a model pretrained to have meaningful inner product scores between embeddings would work better than DeBERTa after doing a preliminary test. We decided to use a pretrained Co-Condenser (Gao and Callan, 2021) model which is a BERT model pre-

[4] https://github.com/facebookresearch/faiss

[5] https://github.com/quickwit-oss/tantivy-py

trained to force information into the [CLS] token and then further pretrained to create meaningful inner products between embeddings.

Given that the Co-Condenser model was only trained for a maximum length of 512 we truncated our passages and queries to that length. Using gradient caching, we were able to train our model with a batch size of 64, where each example included a query, a correct answer, and eight incorrect answers. We sampled the incorrect answers by first embedding all the question titles from the Stack Overflow posts using an off-the-shelf embedding model, we then found the 32 closest titles for each title and randomly selected eight to use as negatives. The titles could then be easily converted to the answers on that post. This method was inspired by TAS (Hofstätter et al., 2021) and chosen as it is relatively quick and it is not too good so that all the selected negative answers are not false negatives.

With the hard negatives and larger batch size we saw much better training dynamics and saw our validation metrics improve over time. We trained with four RTX 2080Tis with a real batch size of 8 per GPU which we cached to achieve the size 64 batch mentioned above. We used AdamW for our optimizer with a learning rate of 5e-6. We trained for 3 days which was around one full epoch of training. Each step took 6.7 seconds due to the gradient caching process.

4.4.3 Repository Search

For the repository search task, we performed experiments on both first-stage dense retrieval and BERT-based re-ranking.

Dense Retrieval We used a similar but more simplified setup than the code search task on top of the SentenceTransformers(Reimers and Gurevych, 2019) library[6]. Here we used only in-batch negative samples and an older distilroberta-base model with less parameters. We fine-tuned the model with max sequence length 512, so that we can encode as much information as possible from the long repository readme files. We also used AdamW optimizer with learning rate 2e-5 and trained the dense encoder for 3 epochs, while performing validation every 1000 steps to store the best performing model. Apart from the distilroberta model, we also fine-tuned a model[7] that was previously pre-trained on multiple sentence representation, NLI and question answering datasets.

For the doc2vec baseline, we used an implementation from the gensim[8] library. We trained a distributed bag of words model (DBOW) using filtered ngrams from the repository metadata (name, description and readme), embedding size 100, 8 negative samples and window size 20 for 20 epochs. No GPU was required for training this model.

Re-Ranking For our custom re-ranking model, we also fine-tuned distilroberta-base, using 8 positives per batch and 4 negatives per positive example for a total batch size of 32 pairs. We trained the model with AdamW optimizer until convergence. For negative sampling, we used a mix of random negatives and negatives from the top 1000 results of a BM25 retriever.

5 Results

5.1 Code Search

The results on the test set, duplicate test set, and validation set can be seen in Table 1. We used two metrics to evaluate our performance mean reciprocal rank (MRR) and recall@100 (R@100). These metrics were chosen as they complement each other and also correspond with the average use case of a code question answer search system. MRR focuses on precision where the reciprocal rank for a given list of documents is the inverse of the rank of the first relevant document. This makes sense as a measure for question answering as people don't want to go searching for the answer and would like the relevant answer to be first on the list. The next metric recall@100 looks at how many of the relevant documents are present in the top 100 returned documents. This is a useful metric because although it is good to have a relevant result in the top few results, when this is not the case we hope that there is at least a relevant result in the top 100 results. Additionally, if we are using a second stage re-ranking model having a high recall is very important as the re-ranker can only do as well as the first stage allows it to.

Our results show that our dense retrieval model beats BM25 in all but one metrics on one dataset. The single area where BM25 does better than our model is MRR on natural language questions, but

[6] https://github.com/UKPLab/sentence-transformers

[7] sentence-transformers/all-mpnet-base-v2

[8] https://radimrehurek.com/gensim/

	Test Set						Duplicate Test Set						Validation Set	
	Titles		Questions		Combined		Titles		Questions		Combined		Combined	
	MRR	**R@100**	**MRR**	**R@100**	**MRR**	**R@100**	**MRR**	**R@100**	**MRR**	**R@100**	**MRR**	**R@100**	**MRR**	**R@100**
BM25	0.0612	0.1540	**0.3714**	0.4597	**0.2163**	0.3069	0.0086	0.0469	0.0050	0.0364	0.0068	0.0417	0.2805	0.4174
Co-Condenser Marco	0.0223	0.07066	0.1653	0.2850	0.0938	0.17783	0.0016	0.0171	0.0033	0.0291	0.0025	0.0231	0.1632	0.3529
Step 10,000	0.0571	0.1688	0.1926	0.2851	0.1249	0.2389	0.0074	0.0589	0.0061	0.0493	0.0068	0.0541	0.2886	0.5408
Step 20,000	0.0821	0.2502	0.3235	0.4616	0.2028	0.3559	0.0081	0.0652	0.0080	0.0522	0.0081	0.0587	0.3201	0.5750
Step 30,000	**0.0886**	**0.2716**	0.3281	**0.4803**	0.2084	**0.3760**	**0.0088**	**0.0683**	**0.0082**	**0.0559**	**0.0085**	**0.0621**	**0.3334**	**0.5914**

Table 1: Table of baseline and dense retrieval results on test set, duplicate test set, and validation set. Note the models named Step are our models after training for that number of steps.

	Recall			nDCG		MAP		MRR
	@5	@10	@100	@10	@100	@10	@100	
BM25	0.1459	0.2305	0.5227	0.1569	0.2381	0.0953	0.1167	0.1975
doc2vec	**0.2200**	**0.3191**	0.5926	**0.2141**	0.2953	**0.1387**	0.1608	0.2524
Dense	0.1495	0.2451	0.6210	0.1655	0.2707	-	0.1230	0.2089
Dense + pre-training	0.1960	0.3154	**0.7411**	0.2112	**0.3333**	0.1254	**0.1620**	**0.2573**
Dense + pre-training + re-rank	0.1233	0.2135	0.7411	0.1398	0.2840	0.0760	0.1112	0.1855

Table 2: Evaluation of various retrieval models for the repository search task on the test set.

as discussed in Section 3.1 this is likely because the answers and questions, on average, have a lot of shared tokens. This makes it easy for BM25 to return the right answer, but in a real world scenario where the question does not share as many tokens with the answer it will likely fail. Our model is even able to beat it on recall@100 despite this advantage as BM25 is overly reliant on the term overlap and some answers will not include term overlap leaving room for our model to beat it. The results on our duplicate question dataset, where a different question is asked that has the same answer as another, further shows BM25 is taking advantage of term overlap. Here BM25 cannot rely on a high token overlap and thus does worse than our model across the board, though both take a substantial hit to performance. This may indicate our model has also learned to put a significant emphasis on shared terms or phrases, which would make sense given the training data. Our model outperforms BM25 on the duplicate test set most significantly on the questions section, this is likely because questions can be very complex and require a more nuanced understanding than simple term matching. On the titles section of the duplicate test set there are more keywords that BM25 can leverage helping it to get relatively decent performance.

In terms of training dynamics, we can see that the test and validation performance seem to be leveling off a bit, although there does still seem to be more room for the model to train. Due to limited time we couldn't evaluate our model after training for a longer period. We do believe that additional training would likely tie the model in the areas where BM25 is beating it. Also note that we did not use the test set to pick our model or otherwise pick hyperparameters, we are only showing it to illustrate the change in performance over time and what potential performance might still be left.

5.2 Repository Search

The results for the repository search task can be seen in table 2. Here, apart from the recall and MRR metrics, we also report the normalized discounted cumulative gain (nDCG) and mean average precision (MAP), since for this task, we care about the order of multiple repositories in the results (e.g. for some queries we have thousands of relevant repositories). For low cutoffs, at the top 5 and top 10 results, surprisingly doc2vec seems to outperform dense retrieval, even when using a pre-trained model on multiple sentence representation tasks. This shows that for some types of retrieval tasks, where the nature of queries is more restricted (e.g. keyword or topic queries), topic modeling approaches might be quite effective. However, for higher cutoff at 100 results, the dense encoder model seems to outperform doc2vec in all the metrics, and by especially big margin in terms of recall. BM25 falls much lower than both supervised models.

When re-ranking the top 100 documents we observed bad performance. The reason behind this might be that re-ranking on a closed-domain dataset can be problematic, because there are often hundreds or even thousands of documents that are related to a query, even though they do not appear in our groundtruth training pairs. When doing negative sampling during training, this high level

of relatedness between multiple queries and documents can make the convergence of the model very difficult.

6 Error Analysis

6.1 Code Search Model

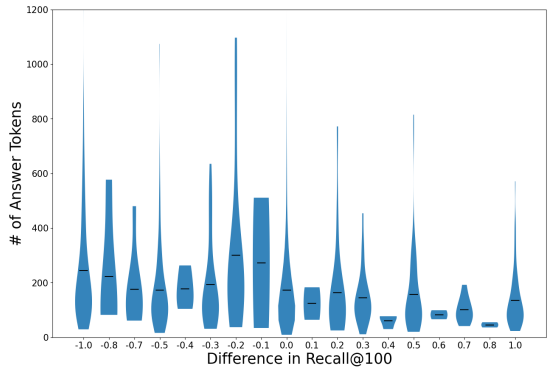
Our code search model does better than the BM25 baseline in several areas, but it still fails in some areas where BM25 succeeds, and can also fail on examples where BM25 also fails. We started our analysis by looking at the results for both BM25 and our own model on both natural language questions and keyword queries (i.e. the Stack Overflow question titles). We noticed a pattern where our model failed and BM25 succeeded. Namely, BM25 seemed to do a lot better than our model for longer answers as well as answers that had a lot of quoted text from the original question. To judge the veracity of these observations we came up with metrics to quantify key properties of the questions and answers and then aligned these with areas where both models did well or poorly. The metrics we picked were longest common subsequence of tokens in the answer and question, the number of tokens in the question, and the number of tokens in the answer. We then found the values for these metrics in different buckets based on how well each model performed. These buckets were found by taking the difference between our dense retrieval model's Recall@100 minus BM25's Recall@100 for each query. We could then find our metrics for each query as well and see how they varied with the differences in the two models.

The results can be seen in Figure 2. We can see that for all three metrics – number of tokens in the question, number of tokens in the answer, and the length of the largest consecutive sub-sequence – that there is a clear pattern where when BM25 does well and dense retrieval does not the values are higher compared to where dense retrieval does well and BM25 does not. This confirms are theory that BM25 does better when answers and questions are longer. We can see an example of this in Figure 1 where the dense retrieval model returns a fairly short answer while the BM25 answer is far longer (though it is truncated in the figure). It seems the number of tokens in the question is particular important with BM25 handily beating our dense retrieval models for longer questions. We see an almost equally as drastic trend in Figure 2c which shows the lengths of the longest shared sub-sequence between the question and answer. It seems that BM25 does especially well when there is a lot of overlap in terms with the question and answer. This is not particularly surprising as term overlap is essentially the only feature BM25 uses. The fact that BM25 does better with longer questions and answer is also not that surprising for two reasons (1) our BERT based retrieval model can only fit 512 tokens in to form a question or answer representation meaning it is likely missing crucial information for longer questions and answers leading BM25 to do better (2) longer questions are more likely to feature keywords BM25 can use to find relevant answers and the same is true of longer answer, additionally longer answer are probably also more likely to contain quotes from the question again helping BM25.

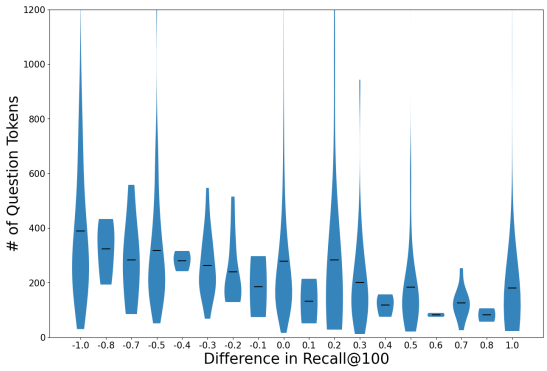
Our analysis shows that for future iterations being able to take in longer sequences is crucial. We had initially planned to do this using a more efficient transformer variant that we could pass the whole questions and answers to, but we decided against this as we couldn't find any models pretrained on a sentence similarity task which seemed to make a big difference in our tests. Thus, although we might have been able to use the whole text we might still have performed worse without substantially more training which we did not have time or resources for. Another way to achieve considering the whole text is to break it into smaller chunks that will fit with in our model. If we had more time we would have likely tried this for both training and inference. We also might consider having multiple representations for each piece of text passed to our model as compressing all the information into a fixed size representation may be loosing important details that BM25 preserves. It might also be worth looking at building a hybrid model to use both the strengths of BM25 and our dense retrieval model.

6.2 Repository Search

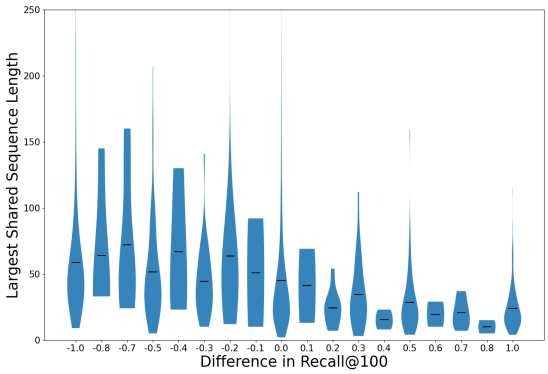
Table 3 shows the top 3 retrieved results for each model (BM25, doc2vec and dense retrieval). We can see that the top results of doc2vec and dense retrieval are quite relevant, and they are even surfaced from the corpus without relying on term overlaps. For example for the term 'nlp', we are able to retrieve repositories that mention 'natural language processing', and for 'reinforcement learning' we get a repository that mentions 'pol-



(a) Number of tokens in answer versus difference in Recall@100



(b) Number of tokens in question versus difference in Recall@100



(c) Length of largest consecutive sub-sequence between answer and question

Figure 2: Graphs showing different question and answer metrics along with the difference in Recall@100 scores for BM25 and our dense code retrieval model. The X-axis corresponds to (Dense Retrieval Recall@100 - BM25 Recall@100) meaning that -1 is when BM25 way outperforms Dense Retrieval while 1 is where Dense Retrieval far outperforms BM25. The difference is rounded to the nearest tenth to get these plots. The data was found on the natural language questions (i.e. Stack Overflow question bodies) test set and is not on the duplicate set.

Question	**Answers**
	BM25
I am trying to sort a list of lists based on the 2nd element in the sub list. Sample Data: <code> [[u'm3u33mm534o', 14, 23], [u'2w3dfbv333g', 20, 34], [u'7kv903nfjft9', 0, 35]] </code> Sort: <code> out.sort(key=lambda x: float(x[1])) </code> ...	<code>flagArray = ['4:a', '27:a', '33:a', '39:b', '5:c', '31:c', '44:c', '45:c', '18:d', '37:d', '2:e', '21:e', '25:e', ... '41:_','0:$']``` def getNumber(x): return int(x.split(":")[0]) sorted(flagArray, key = getNumber)
	Dense Retrieval
	You have to try <code>>>> list.sort(key=lambda x: x[2]) >>> list [[u'ingenieurin', u'NULL', u'0.0000004'], [u'ingenieurin', u'you', u'0.0000008'], [u'ingenieurin', u'anything', u'0.0000328'],... <code>list.sort</code> will perform operation on self. It will return None. If you have to create new sorted list then you have to use 'sorted' inbuilt function.

Figure 3: The first 2 non-identical results retrieved from BM25 and our DR model for a coding question.

icy optimization'. BM25 performs quite poorly in multiple cases, because in some cases, words with high frequency can be misleading. For example, for the 'NLP' query, we get accurate exact match results, but for 'reinforcement learning' BM25 returns results that just mention 'learrning' with high frequency. The 2 supervised models do not appear to have this issue. The main disadvantage of doc2vec is that it has a fixed vocabulary of ngrams. This means that it has much lower representation capacity that BERT-based language models. BERT models are able to capture much better the sequential meaning of words. doc2vec could theoretically use much larger vocabulary size and longer ngrams, but longer ngrams will have low frequency in a corpus and their learned representations would be quite poor.

7 Contribution of group members

- Julian: Collected Stack Overflow data, processed Stack Overflow data, created duplicate question dataset, pretrained DeBERTa model on Stack Overflow data, trained dense retrieval model on Stack Overflow dataset, evaluated Stack Overflow model, and wrote significant portion of final report.

- Chris Samarinas: Helped with the construction of the Github dataset, implemented and trained models for the repository search task, implemented the re-ranker model, and wrote a significant portion of the report.

- Nandyala Siddharth Kantu: Collected Stack Overflow data, ran some early experiments on re-ranking on the Stack Overflow dataset and wrote few parts of the report.

- Siddharth Nammara Kalyana Raman: Helped with the construction of the Github dataset, worked on multi-Label classification model that did not work well and was not included in the final report, wrote few parts of the report.

8 Conclusions & Future Work

In this project we explored the effectiveness of dense retrieval models for two tasks; code and repository search. Our experimental results showed that dense encoders are very effective for both tasks, outperforming BM25 by a big margin, and they probably have space for further improvement given more time and compute. For the repository search task, a doc2vec baseline gave better metrics in higher ranks, however we suspect that this could be due to insufficient training of the dense retrieval model. More work and experimentation and analysis are needed for making the re-ranking model work well for the repository search task, and a next step would also be some experiments with re-ranking on the larger-scale Stack Overflow dataset.

References

Anonymous (2021). Repo4qa: Answering complex coding questions via dense retrieval on github repositories. *ACL ARR*.

Baltes, S., Dumani, L., Treude, C., and Diehl, S. (2018). Sotorrent: reconstructing and analyzing the evolution of stack overflow posts. In Zaidman, A., Kamei, Y., and Hill, E., editors, *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*, pages 319–330. ACM.

Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. (2005). Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, page 89–96, New York, NY, USA. Association for Computing Machinery.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. In Burstein, J., Doran, C., and Solorio, T., editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Gao, L. and Callan, J. (2021). Condenser: a pre-training architecture for dense retrieval. In Moens, M., Huang, X., Specia, L., and Yih, S. W., editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 981–993. Association for Computational Linguistics.

Gao, L., Ma, X., Lin, J., and Callan, J. (2022). Tevatron: An efficient and flexible toolkit for dense retrieval. *CoRR*, abs/2203.05765.

Gotmare, A. D., Li, J., Joty, S. R., and Hoi, S. C. H. (2021). Cascaded fast and slow models for efficient semantic code search. *ArXiv*, abs/2110.07811.

Gu, X., Zhang, H., and Kim, S. (2018). Deep code search. *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 933–944.

He, P., Gao, J., and Chen, W. (2021a). Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing. *CoRR*, abs/2111.09543.

Query	Results
nlp	**BM25**
	(1) spark-nlp: State of the Art Natural Language Processing (2) deep_learning_NLP: Keras, PyTorch, and NumPy Implementations of Deep Learning Architectures for NLP (3) ctr_nlp: Click-Through Rate Prediction with NLP
	doc2vec
	(1) indonlu: The first-ever vast natural language processing benchmark for Indonesian Language (2) squad-question-answering: Question answering on the SQuAD dataset, for NLP class at UNIBO (3) nlp-paradigm-shift: Paradigm shift in natural language processing
	Dense Retrieval
	(1) gpt2-dialogue-generation-pytorch: The PyTorch implementation of fine-tuning the GPT-2 for dialogue generation. (2) katecheo : Modular, multi-topic question answering on top of Kubernetes (3) wikipedia2vec : A tool for learning vector representations of words and entities from Wikipedia
reinforcement learning	**BM25**
	(1) Awesome-Learning-with-Label-Noise: A curated list of resources for Learning with Noisy Labels (2) Hands-on-One-Shot-Learning: This repository is for coding exercises listed in Book Hands on One Shot Learning. (3) Adaptive_Multi-curricula_Learning_for_Dialog: The codebase for "Learning from Easy to Complex: Adaptive Multi-curricula Learning for Neural Dialogue Generation
	doc2vec
	(1) ICCV2019-LearningToPaint: ICCV2019 - Learning to Paint With Model-based Deep Reinforcement Learning (2) Upside-Down-Reinforcement-Learning: Implementation of Schmidhuber's Upside Down Reinforcement Learning paper in PyTorch (3) ros2learn: ROS 2 enabled Machine Learning algorithms
	Dense Retrieval
	(1) CHQ-Summ: PyTorch code for ACL 2021 paper Reinforcement Learning for Abstractive Question Summarization with Question-aware Semantic Rewards (2) pd_zdpg_plus: Code for MLSP 2021 paper on "Model-Free Learning of Optimal Deterministic Resource Allocations in Wireless Systems via Action-Space Exploration" (3) walk_the_blocks: Implementation of Scheduled Policy Optimization for task-oriented language grouding

Table 3: The top three retrieved examples from all retrieval models for two queries in the Github dataset

He, P., Liu, X., Gao, J., and Chen, W. (2021b). Deberta: decoding-enhanced bert with disentangled attention. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Hill, E., Pollock, L. L., and Vijay-Shanker, K. (2011). Improving source code search with natural language phrasal representations of method signatures. *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 524–527.

Hofstätter, S., Lin, S., Yang, J., Lin, J., and Hanbury, A. (2021). Efficiently teaching an effective dense retriever with balanced topic aware sampling. In Diaz, F., Shah, C., Suel, T., Castells, P., Jones, R., and Sakai, T., editors, *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, pages 113–122. ACM.

Karpukhin, V., Oguz, B., Min, S., Lewis, P. S. H., Wu, L., Edunov, S., Chen, D., and Yih, W. (2020). Dense passage retrieval for open-domain question answering. In Webber, B., Cohn, T., He, Y., and Liu, Y., editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 6769–6781. Association for Computational Linguistics.

Kudo, T. and Richardson, J. (2018). Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In Blanco, E. and Lu, W., editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018*, pages 66–71. Association for Computational Linguistics.

Lee, K., Chang, M., and Toutanova, K. (2019). Latent retrieval for weakly supervised open domain question answering. In Korhonen, A., Traum, D. R., and Màrquez, L., editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 6086–6096. Association for Computational Linguistics.

Leonhardt, J., Beringer, F., and Anand, A. (2021). Exploiting sentence-level representations for passage ranking. In *LWDA*.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Lu, M., Sun, X., Wang, S., Lo, D., and Duan, Y. (2015). Query expansion via wordnet for effective code search. In Guéhéneuc, Y., Adams, B., and Serebrenik, A., editors, *22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015, Montreal, QC, Canada, March 2-6, 2015*, pages 545–549. IEEE Computer Society.

Nie, L., Jiang, H., Ren, Z., Sun, Z., and Li, X. (2016). Query expansion based on crowd knowledge for code search. *IEEE Transactions on Services Computing*, 9:771–783.

Nogueira, R. and Cho, K. (2019). Passage re-ranking with bert. *ArXiv*, abs/1901.04085.

Nogueira, R., Yang, W., Cho, K., and Lin, J. J. (2019). Multi-stage document ranking with bert. *ArXiv*, abs/1910.14424.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084.

Ren, R., Qu, Y., Liu, J., Zhao, W. X., She, Q., Wu, H., Wang, H., and Wen, J.-R. (2021). Rocketqav2: A joint training method for dense passage retrieval and passage re-ranking. *ArXiv*, abs/2110.07367.

Robertson, S. E. and Zaragoza, H. (2009). The probabilistic relevance framework: BM25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389.

Silva, R. F., Rahman, M. M., de Carvalho Dantas, C. E., Roy, C. K., Khomh, F., and de Almeida Maia, M. (2021). Improved retrieval of programming solutions with code examples using a multi-featured score. *J. Syst. Softw.*, 181:111063.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2019). GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Xu, P., Ma, X., Nallapati, R., and Xiang, B. (2019). Passage ranking with weak supervsion. *ArXiv*, abs/1905.05910.

Yang, P., Fang, H., and Lin, J. (2017). Anserini: Enabling the use of lucene for information retrieval research. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 1253–1256, New York, NY, USA. Association for Computing Machinery.

Ye, X., Shen, H., Ma, X., Bunescu, R. C., and Liu, C. (2016). From word embeddings to document similarities for improved information retrieval in software engineering. *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 404–415.