CS291S Project Report

Michael Saxon Xinyi Wang

December 2021

1 Objectives and Challenges

The objective of this project is to understand and reproduce the results from the paper *Improving Efficient Neural Ranking Models with Cross-Architecture Knowledge Distillation* [5]. In this paper, the authors propose a new training technique for neural ranking models called Cross-Architecture Knowledge Distillation.

While neural ranking models show great performance improvement compared to classic ranking models, neural networks require a high amount of compute and there is usually a trade-off between efficiency and effectiveness at query time. The goal of this paper is to improve the effectiveness of efficient neural ranking models without compromising their query latency benefits.

2 State-of-the-art techniques

The paper leveraged an novel version of the knowledge distillation [4] to improve the effectiveness of four recent efficient neural ranking models with different architectures that are specially designed to transfer computation to the indexing phase to require less resources at query time.

Knowledge distillation [4] is a common approach to close the performance gap between a large teacher model (slow) and a small student model (fast). Teacher model and student model usually have the same architecture and only different in size, so the output scores of teacher and student model are usually in the same range. To distill knowledge from teach to student, a standard Mean Squared Error (MSE) loss is utilized to minimizing the difference between the absolute output scores of a teacher and a student.

The teacher model BERT_{CAT} is a direct adaptation from BERT [3], which is a large pre-trained language model that is widely used in neural ranking models. BERT_{CAT} concatenate the query and the passage to feed them to a pre-trained BERT model, then output a score from the resulting [CLS] representation. The ranking model can then use the score to produce the ranking between the retrieved passages for a specific query.

The first student model BERT_{DOT} also known in the literature as Tower-BERT [2], BERT-Siamese [11], or TwinBERT [8], is also based on BERT [3]. It uses dot product between the [CLS] token representation computed from a full BERT of query and passage as the output score.

The second BERT-based student model ColBERT [7] use the representations of all the tokens computed from a full BERT as the encoded query/passage, then calculate the output

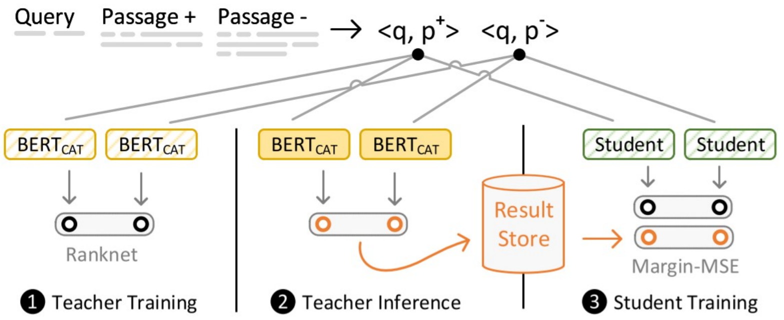


Figure 2: Our knowledge distillation process, re-visiting the same training triples in all steps: ❶ Training the BERT$_{\text{CAT}}$ model; ❷ Using the trained BERT$_{\text{CAT}}$ to create scores for all training triples; ❸ Individually training the student models with Margin-MSE using the teacher scores.

score as a dot product between each tokens in the query and passage. Then do max-pooling across the passage and sum across the query.

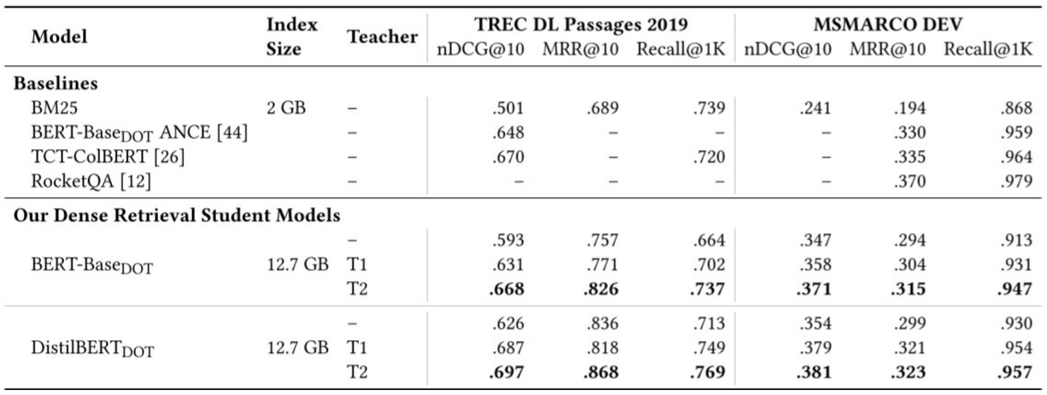
The third BERT-based student model PreTT [9] use the representations of all the tokens computed from the first b BERT-layers as the encoded query/passage, then concatenates the resulting passage and query sequences with a [SEP] separator token and computes the remaining BERT-layers. Then the score is produced by the final CLS token representation.

The fourth student model Transformer-Kernel (TK) [6] is not a BERT-based model. It represents each token using a weighted average over a classic word embedding (e.g. word2vec, GloVe) and a contextualized representation (e.g. a shallow transformer TF). For calculating the score, it first compute cosine similarity between each token in the query and the passage. Then apply a set of Gaussian kernels. Then aggregate the scores by summing over the passage and query and then weighted sum over different kernels.

3 Key algorithms

Since each student model has a different way of calculating scores, the output scores are usually in a different range for different model. However, the difference in score for a relevant and non-relevant passage pair is usually similar across different architectures. So instead of using the conventional pointwise MSE that minimize the difference between the absolute output scores, the authors proposed to use a Margin-MSE to minimize the margin between the scores of the relevant and the non-relevant sample passage per query.

The knowledge distillation process use the triplet (query, relevant passage, non-relevant passage) as the train data, and the whole process include three steps. In the first step, a teacher model, BERT$_{\text{CAT}}$ is trained using the normal RankNet [1] algorithm that use Sigmoid function to approximate the probabily of whether a passage is relevant to the query or not. In the second step, we use the trained BERT$_{\text{CAT}}$ to inference the teacher scores for all training triplets. In the third step, we use Margin-MSE to train each student model with teacher scores.



Model	Index Size	Teacher	TREC DL Passages 2019 nDCG@10	TREC DL Passages 2019 MRR@10	TREC DL Passages 2019 Recall@1K	MSMARCO DEV nDCG@10	MSMARCO DEV MRR@10	MSMARCO DEV Recall@1K
Baselines								
BM25	2 GB	–	.501	.689	.739	.241	.194	.868
BERT-Base$_{DOT}$ ANCE [44]		–	.648	–	–	–	.330	.959
TCT-ColBERT [26]		–	.670	–	.720	–	.335	.964
RocketQA [12]		–	–	–	–	–	.370	.979
Our Dense Retrieval Student Models								
		–	.593	.757	.664	.347	.294	.913
BERT-Base$_{DOT}$	12.7 GB	T1	.631	.771	.702	.358	.304	.931
		T2	**.668**	**.826**	**.737**	**.371**	**.315**	**.947**
		–	.626	.836	.713	.354	.299	.930
DistilBERT$_{DOT}$	12.7 GB	T1	.687	.818	.749	.379	.321	.954
		T2	**.697**	**.868**	**.769**	**.381**	**.323**	**.957**

Figure 1: Results reported in the paper.

4 Experiments

We demonstrate running the model proposed by Hofstatter et al. [6] using checkpoints in `https://github.com/sebastian-hofstaetter/neural-ranking-kd`. We run our experiments using pyserini (`https://github.com/castorini/pyserini`) and test the model on a novel challenge retrieval set produced from SQuAD.

4.1 Dataset

Stanford Question Answering Dataset (SQuAD) [10] is a reading comprehension dataset, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable. In our experiments, we use the articles to build the index and then use the questions as queries to test the ranking performance of our models.

Each article consists of some paragraphs p_i which contain questions $q_{i,j}$. Each question is sufficiently answered by the information contained in its paragraph p_i. We thus produce our challenge SQuAD retrieval set as follows: Each paragraph is considered as a document in our index, leading to a total of 20,239 documents from all paragraphs in the training and test sets. We then use the test set questions as our query set for a total of 11,873 questions.

We produce this using the following python code:

```python
#squaddev[”data”][0]['paragraphs'][4]['qas']
#index with ascending number for each paragraph, treat each p as a doc
#['context'] : the paragraph itself
#['qas'] : a list of 'question', 'id' that match to this paragraph

import json
import random
from collections import defaultdict

random.seed(1248)

docs = 0
```

```
train = ”squad˙test/train-v2.0.json”
test = ”squad˙test/dev-v2.0.json”

# we build docs list to contain train and test qs, q list only to contain
    ↪  test qs
docs˙list = []
current˙did = 0

# minidoc˙doc -¿ get all minidocs that correspond to this current
    ↪ document
minidoc˙doc = defaultdict(list)
# a list of
doc˙minidoc = []

data = json.load(open(train))[”data”]
# each elem in data corresponds to a single wiki doc
# we will treat each paragraph as its own ”mini doc” for testing purposes

for j, document in enumerate(data):
    title = document[”title”]
    for paragraph in document['paragraphs']:
        docs˙list.append((current˙did, title, paragraph[”context”].
    ↪ replace(””n”,” ”)))
        minidoc˙doc[j].append(current˙did)
        doc˙minidoc.append(j)
        current˙did += 1
        docs += 1

# save how many pages we visited
jstart = len(data)
data = json.load(open(test))[”data”]
questions˙list = []

# now we also build questionlist simultaneously
for j, document in enumerate(data):
    title = document[”title”]
    for paragraph in document['paragraphs']:
        docs˙list.append((current˙did, title, paragraph[”context”].
    ↪ replace(””n”,” ”)))
        minidoc˙doc[j].append(current˙did)
        doc˙minidoc.append(j+jstart)
        paragraph˙questions = paragraph[”qas”]
        for question in paragraph˙questions:
            qid = question[”id”]
            qtext = question[”question”]
            questions˙list.append((qid, qtext, current˙did))
        current˙did += 1
        docs += 1
```

```
# build document outstring
docoutlist = []
for did, dtitle, dtext in docs˙list:
    docoutlist.append(json.dumps(–”id”:did,”contents”:dtitle+”“n”+dtext˝)
    ↪ +”“n”)
with open(”squad˙docs.jsonl”,”w”) as f:
    f.writelines(docoutlist)

# generate qrels file
# query-id 0 document-id relevance (0 or 1)
qrels˙lines = []
easy˙qrels˙lines = []
question˙file˙lines = []
for i, (qid, qtext, did) in enumerate(questions˙list):
    question˙file˙lines.append(f”–i˝“t–qtext˝“n”)
    false˙judge = random.sample(range(0,current˙did),10)
    qrels˙lines.append(f”–i˝ 0 –did˝ 1“n”)
    easy˙qrels˙lines.append(f”–i˝ 0 –did˝ 1“n”)
    for ˙did in random.sample(range(0,current˙did),10):
        if did == ˙did:
            continue
        qrels˙lines.append(f”–i˝ 0 –˙did˝ 0“n”)
        if did in minidoc˙doc[doc˙minidoc[˙did]]:
            continue
        easy˙qrels˙lines.append(f”–i˝ 0 –˙did˝ 0“n”)
    for ˙did in minidoc˙doc[doc˙minidoc[did]]:
        easy˙qrels˙lines.append(f”–i˝ 0 –˙did˝ 1“n”)
with open(”squad˙questions.tsv”,”w”) as f:
    f.writelines(question˙file˙lines)
with open(”squad˙questions.qrel”,”w”) as f:
    f.writelines(qrels˙lines)
with open(”squad˙questions˙easy.qrel”,”w”) as f:
    f.writelines(easy˙qrels˙lines)
print(f”Processed –docs˝ documents.”)
```

This generates the files squad_docs.jsonl, squad_questions.qrel, and squad_questions.tsv, containing the preprocessed index for dense processing by pyserini, the answers for comparison using the TREC eval scripts, and the input dev questions for answering by the eval script.

4.2 Execution Code

To produce the search results using scripts of the following form:

```
python -m pyserini.encode input --corpus squad˙docs.jsonl --fields title
    ↪ text
    output --embeddings squad˙test/index˙distilbertdot --to-faiss
    encoder --encoder sebastian-hofstaetter/distilbert-dot-margin˙mse-T2-
    ↪ msmarco --fields title text --batch 20

python -m pyserini.dsearch --topics squad˙questions.tsv --index
    ↪ squad˙test/index˙distilbertdot --encoder sebastian-hofstaetter/
    ↪ distilbert-dot-margin˙mse-T2-msmarco --batch-size 20 --threads 12
    ↪ --output squad˙test/out˙distilbertdot.trec
```

After executing the following commands, output `out_distilbertdot.trec` is produced containing the results chosen by the model.

4.3 Evaluation Metric

The metrics we use are nDCG@10, Recall@10, MRR@10 and MAP@100.

nDCG (Normalized Discounted Cumulative Gain) measures the usefulness, or gain, of a document based on its position in the result list. MRR (Mean Reciprocal Rank) measures the ranking quality by averaging the inverse of the rank of the correct answer over all the queries. MAP (Mean Average Precision) measures the precision of getting the correct answer.

We assess MAP and Recall using the `trec_eval` system implemented by pyserini. To assess ndcg and mrr we use the following script using the `pytrec_eval` package.

```python
import pytrec˙eval
import json
from collections import defaultdict
import argparse

def file˙to˙dict(fname, delim, qkey˙field, dkey˙field, val˙field, valtype
    ↪ ):
    lines = open(fname,”r”).readlines()
    d = defaultdict(dict)
    for line in lines:
        line = line.strip().split(delim)
        d[line[qkey˙field]][line[dkey˙field]] = valtype(line[val˙field])
    return d

parser = argparse.ArgumentParser()
parser.add˙argument(”runf”, type=str)
args = parser.parse˙args()
qrel = file˙to˙dict(”squad˙questions.qrel”,” ”,0,2,3,int)
run = file˙to˙dict(args.runf, ” ”, 0,2,4,float)

evaluator = pytrec˙eval.RelevanceEvaluator(
    qrel, –'recip˙rank', 'ndcg'˝)

results = evaluator.evaluate(run)

ndcg˙sum = 0
map˙sum = 0
count = 0

for query in results.keys():
    ndcg˙sum += results[query][”ndcg”]
    map˙sum += results[query][”recip˙rank”]
    count += 1

print(f”ndcg“t–ndcg˙sum/count:.4f˝“nmrr“t–map˙sum/count:.4f˝”)
```

Finally, we compare the performance of the systems in terms of latency to complete a single search, using the following:

```python
from pyserini.dsearch import SimpleDenseSearcher, AutoQueryEncoder,
    ↪ DprQueryEncoder
import time
from transformers import AutoTokenizer, AutoModel

pre˙trained˙model˙name = ”sebastian-hofstaetter/distilbert-dot-margin˙mse
    ↪ -T2-msmarco”
index = ”./squad˙test/index˙distilbertdot”

def get˙runtime(pre˙trained˙model˙name, index, encoder=AutoQueryEncoder):
    #tokenizer = AutoTokenizer.from˙pretrained(pre˙trained˙model˙name)
    #bert˙model = AutoModel.from˙pretrained(pre˙trained˙model˙name)
    encoder = encoder(pre˙trained˙model˙name)
    searcher = SimpleDenseSearcher(
        index˙dir = index,
        query˙encoder = encoder
    )

    t1 = time.time()
    hits = searcher.search('What state is Beyonce from?')
    t2 = time.time()

    for i in range(0, 10):
        print(f'–i+1:2˝ –hits[i].docid:7˝ –hits[i].score:.5f˝')

    print(f”Search completed in –(t2-t1)*1000:.2f˝ms”)

get˙runtime(pre˙trained˙model˙name, index)
get˙runtime(”bert-base-uncased”, ”./squad˙test/index˙bertbase”)
get˙runtime(”facebook/dpr-ctx˙encoder-multiset-base”, ”./squad˙test/
    ↪ index˙dpr”, DprQueryEncoder)
```

4.4 Results

Fundamentally, attempting to replicate the results using pyserini was flawed. The 22GB prebuilt MS MARCO index (which we had hoped to use to get around the need to fully index the MS MARCO document set ourselves using each neural model) is built on top of faiss, a package for managing and reading from dense indices. Unfortunately, at present faiss-gpu attempts to load the entire index into RAM, leading to a “bad malloc call” error on Michael’s machine where we attempted replication. Thus, we were unable to do replication of the MS MARCO results using pyserini.

Thus we turned to instead analyze the performance on our SQuAD challenge set. However without fine-tuning the pretrained passage encoders on this modality, performance suffered.

In particular, it appears that pyserini had loading difficulties on BERT-base and DPR pretrained encoders. This is probably because those checkpoints on their own don’t include parameters that were fine-tuned for the final step of passage encoding and are instead just grabbing the unadapted CLS tokens.

Method	nDCG@10	Recall@10	MRR@10	MAP@100	CPU Spd	GPU Spd
BM25	0.8282	0.9060	0.7845	0.7845	95 ms	—
BERT-base	0.0342	0.0196	0.0091	0.0091	78 ms	57 ms
DPR	0.0045	0.0003	0.0002	0.0003	65 ms	56 ms
DistillBERT$_{\text{DOT}}$	0.6535	0.7648	0.5725	0.5725	70 ms	**19 ms**

Table 1: Our results on SQuAD

However, we were happy to see that some transfer learning performance takes place from MS MARCO to our SQuAD challenge set, as is demonstrated by the respectable performance of DistillBERT$_{\text{DOT}}$ on the challenge set in Table 1. Additionally, we were impressed to see that DistillBERT (on a GPU) is actually faster than the CPU latency for a pyserini simple search from the BM25 inverted index. However, on CPU we were surprised to find that all three neural models beat SimpleSearcher with BM25 on latency. We performed CPU-testing by running the same script in a separate conda environment with faiss-cpu rather than faiss-gpu installed.

5 Our efforts

We put a lot of efforts in studying the paper and reading related works (e.g. knowledge distillation [4], ColBERT [7], PreTT [9], Transformer-Kernel (TK) [6], RankNet [1]) to get a better understanding of the proposed method. And we learned pyserini to reproduce the results using the proposed method.

The lecture that introduced BERT showed that BERT-base ranking models are very strong in performance. However, it is very expensive to run BERT and we are interested in how to lower the query latency while maintaining the performance. So we study the paper *Improving Efficient Neural Ranking Models with Cross-Architecture Knowledge Distillation* [5] as a possible solution to this problem.

Due to the problems associated with running MS MARCO replication recipes in pyserini directly on our hardware (as mentioned above) we were instead forced to implement our own recipe using SQuAD. This led to further downstream difficulties as discussed above.

Despite these problems, we leave this project with a deeper understanding of the technical details required to produce a full search pipeline from scratch on a new dataset using pyserini. Michael will apply these learnings in instructing his ERSP group of undergraduates working with the NLP lab to create a new causal question answering dataset.

Additionally, we were able to confirm the superior performance of DistillBERT over the alternatives in terms of latency on GPU. We were surprised to see that the CPU latency of the neural architectures is actually the same. This is probably due to some implementation details in the pyserini package.

References

[1] C. J. Burges. From ranknet to lambdarank to lambdamart: An overview. Technical Report MSR-TR-2010-82, June 2010. URL `https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/`.

[2] W.-C. Chang, F. X. Yu, Y.-W. Chang, Y. Yang, and S. Kumar. Pre-training tasks for embedding-based large-scale retrieval. *arXiv preprint arXiv:2002.03932*, 2020.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://aclanthology.org/N19-1423`.

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

[5] S. Hofstätter, S. Althammer, M. Schröder, M. Sertkan, and A. Hanbury. Improving efficient neural ranking models with cross-architecture knowledge distillation. *arXiv preprint arXiv:2010.02666*, 2020.

[6] S. Hofstätter, M. Zlabinger, and A. Hanbury. Interpretable & time-budget-constrained contextualization for re-ranking. *arXiv preprint arXiv:2002.01854*, 2020.

[7] O. Khattab and M. Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48, 2020.

[8] W. Lu, J. Jiao, and R. Zhang. Twinbert: Distilling knowledge to twin-structured bert models for efficient retrieval. *arXiv preprint arXiv:2002.06275*, 2020.

[9] S. MacAvaney, F. M. Nardini, R. Perego, N. Tonellotto, N. Goharian, and O. Frieder. Efficient document re-ranking for transformers by precomputing term representations. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 49–58, 2020.

[10] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[11] L. Xiong, C. Xiong, Y. Li, K.-F. Tang, J. Liu, P. Bennett, J. Ahmed, and A. Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. *arXiv preprint arXiv:2007.00808*, 2020.

Improving Efficient Neural Ranking Models with Cross-Architecture Knowledge Distillation

Authors: Sebastian Hofstätter, Sophia Althammer, Michael Schröder, Mete Sertkan and Allan Hanbury
TU Wien, Vienna, Austria

Presenters: Michael Saxon, Xinyi Wang

Knowledge distillation for neural ranking models

- **Neural ranking models**: trade-off between efficiency and effectiveness at query time
- **Knowledge distillation**: common approach to close the performance gap between a large teacher model (slow) and a small student model (fast)
- Teacher model and student model usually have the **same architecture** and only different in size ⟶ Output scores are in the **same range**
- **Pointwise Mean Squared Error loss:** minimizing MSE between the **absolute scores** of a teacher and a student
- **Goal**: improve the effectiveness of efficient neural ranking models without compromising their query latency benefits

Cross-architecture knowledge distillation

- **Teacher model (M_t: BERT$_{CAT}$):** concatenate query and passage and feed to a pre-trained BERT model, then output a score from the CLS representation
- **Student model (M_s)**: BERT-like architecture, but specially designed to transfer computation to the indexing phase to require less resources at query time (e.g. ColBERT, PreTT, etc)
- Output scores of different architectures are in **different ranges**
- **Margin Mean Squared Error loss** (Margin-MSE): optimizing the **margin between the scores** of the relevant (P+) and the non-relevant (P-) sample passage per query (Q)

$$\mathcal{L}(Q, P^+, P^-) = \text{MSE}(M_s(Q, P^+) - M_s(Q, P^-), \\ M_t(Q, P^+) - M_t(Q, P^-))$$

Motivation

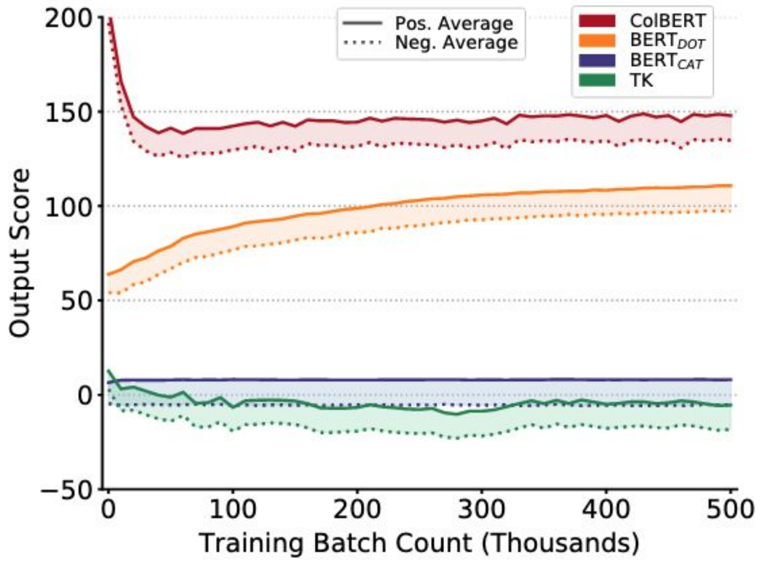


Figure 1: Raw query-passage pair scores during training of different ranking models. The margin between the positive and negative samples is shaded.

- Different student models exhibit unique dynamic ranges in raw scores
- Despite this, **margins are similar**, on average, between positive and negative examples
- Authors propose using the ***margin*-based loss** instead of *absolute output scores* to improve distillation

Cross-architecture knowledge distillation process

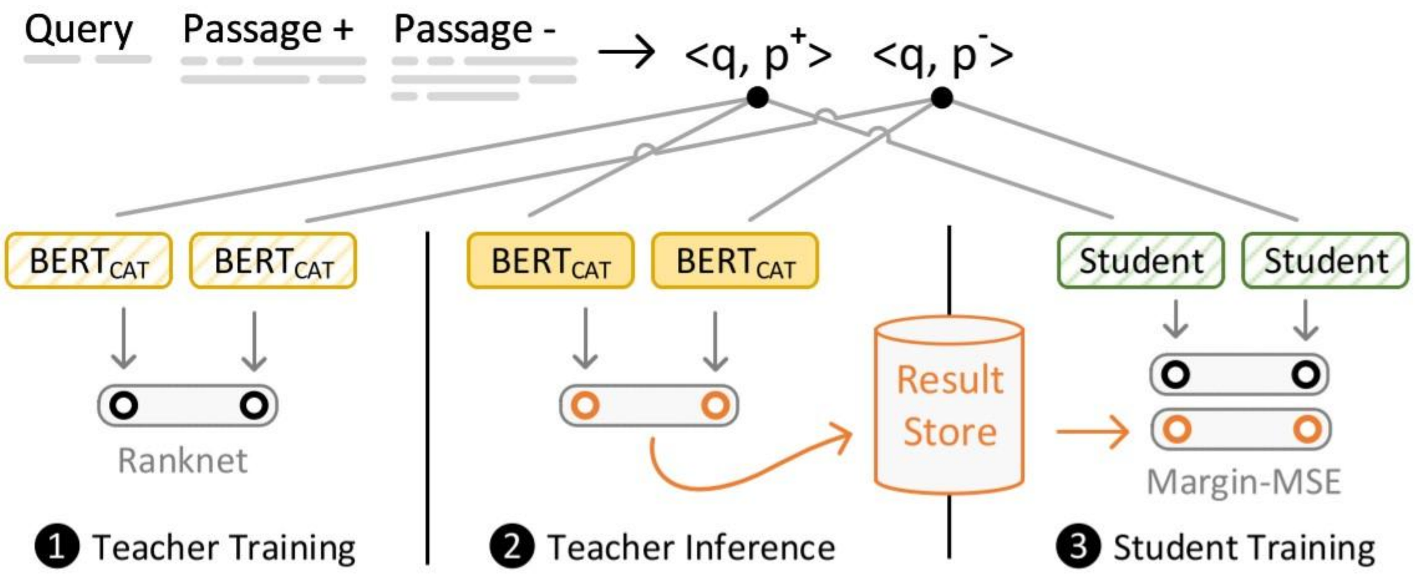


Figure 2: Our knowledge distillation process, re-visiting the same training triples in all steps: ❶ Training the BERT$_{CAT}$ model; ❷ Using the trained BERT$_{CAT}$ to create scores for all training triples; ❸ Individually training the student models with Margin-MSE using the teacher scores.

Student architecture 1: **BERT$_{\text{DOT}}$**

- **Query q**: CLS token representation computed from a full BERT
- **Passage p**: CLS token representation computed from a full BERT, precomputed

$$\hat{q} = \text{BERT}([\text{CLS}; q_{1:m}])_1 * W_s$$

$$\hat{p} = \text{BERT}([\text{CLS}; p_{1:n}])_1 * W_s$$

- **Score**: use dot product between the encoded representations

$$\text{BERT}_{\text{DOT}}(q_{1:m}, p_{1:n}) = \hat{q} \cdot \hat{p}$$

Student architecture 2: **ColBERT**

- **Query q**: append several MASK tokens to the end, and use the representations of all the tokens computed from a full BERT
- **Passage p**: representations of all the tokens computed from a full BERT, precomputed (high storage cost)

$$\hat{q}_{1:m} = \mathrm{BERT}([\mathrm{CLS}; q_{1:m}; \mathrm{rep}(\mathrm{MASK})]) * W_s$$
$$\hat{p}_{1:n} = \mathrm{BERT}([\mathrm{CLS}; p_{1:n}]) * W_s$$

- **Score**: calculate the dot product between each tokens in q and p (m*n computation). Then do max-pooling across p and sum across q.

$$\mathrm{ColBERT}(q_{1:m}, p_{1:n}) = \sum_{1}^{m} \max_{1..n} \hat{q}_{1:m}^{T} \cdot \hat{p}_{1:n}$$

Student architecture 3: **PreTT**

- **Query q**: representations of all the tokens computed from the first b BERT-layers
- **Passage p**: representations of all the tokens computed from the first b BERT-layers, precomputed (high storage cost)

$$\hat{q}_{1:m} = \underset{1:b}{\text{BERT}}([\text{CLS}; q_{1:m}])$$

$$\hat{p}_{1:n} = \underset{1:b}{\text{BERT}}([\text{CLS}; p_{1:n}])$$

- **Score**: concatenates resulting p, q sequences with a SEP separator token and computes the remaining BERT-layers. Use the final CLS token representation to produce score. (high computation cost)

$$\text{PreTT}(q_{1:m}, p_{1:n}) = \underset{b:\hat{b}}{\text{BERT}}([\hat{q}_{1:m}; \text{SEP}; \hat{p}_{1:n}])_1 * W_s$$

Student architecture 4: **Transformer-Kernel (TK)**

- **Query q**: represent each token using a weighted average over a classic word embedding (e.g. word2vec, GloVe) and a contextualized representation (e.g. a shallow transformer TF)
- **Passage p**: same as q, precomputed (high storage cost)

$$\hat{q}_i = q_i * \alpha + \mathrm{TF}(q_{1:m})_i * (1 - \alpha)$$

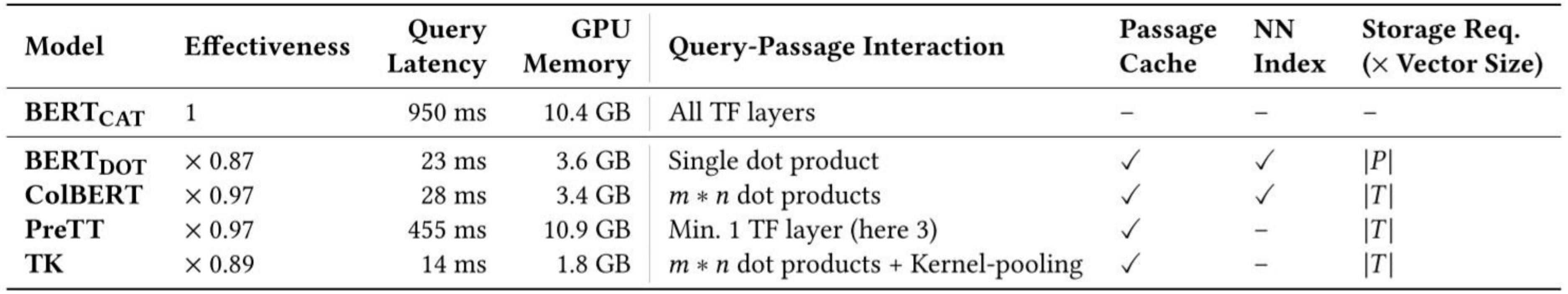
$$\hat{p}_i = p_i * \alpha + \mathrm{TF}(p_{1:n})_i * (1 - \alpha)$$

- **Score**: first compute cosine similarity between each token in q and p. Then apply a set of Gaussian kernels. Then aggregate the scores by summing over p and q and then weighted sum over different kernels.

$$K_{i,j}^{k} = \exp\left(-\frac{(\cos(\hat{q}_i, \hat{p}_j) - \mu_k)^2}{2\sigma^2}\right) \qquad \mathrm{TK}(q_{1:m}, p_{1:n}) = \left(\sum_{i=1}^{m} \log\left(\sum_{j=1}^{n} K_{i,j}^{k}\right)\right) * W_s$$

Compare different architectures

Table 1: Comparison of model characteristics using DistilBERT instances. *Effectiveness* compares the baseline nDCG@10 of MSMARCO-DEV. *NN Index* refers to indexing the passage representations in a nearest neighbor index. $|P|$ refers to the number of passages; $|T|$ to the total number of term occurrences in the collection; m the query length; and n the document length.



Model	Effectiveness	Query Latency	GPU Memory	Query-Passage Interaction	Passage Cache	NN Index	Storage Req. (× Vector Size)
BERT_{CAT}	1	950 ms	10.4 GB	All TF layers	–	–	–
BERT_{DOT}	× 0.87	23 ms	3.6 GB	Single dot product	✓	✓	$\|P\|$
ColBERT	× 0.97	28 ms	3.4 GB	$m * n$ dot products	✓	✓	$\|T\|$
PreTT	× 0.97	455 ms	10.9 GB	Min. 1 TF layer (here 3)	✓	–	$\|T\|$
TK	× 0.89	14 ms	1.8 GB	$m * n$ dot products + Kernel-pooling	✓	–	$\|T\|$

- **BERT**$_{\textbf{CAT}}$ is the most effective model yet not practical
- **TK** is the most efficient model while it’s less effective

Datasets, Training (for Distillation)

- MSMARCO-Passage
  - Training set
  - 8.8M Bing queries sampled
- MSMARCO-Dev
  - 49k queries, sparsely judged
- TREC-DL ‘97
  - 43 densely-judged queries
- All student LMs start from DistilBERT 6 layer checkpoint
- Trained using Adam with consistent LR

Experiments: KD Across Architectures?

- For each model the authors consider their margin-MSE distillation against RankNet and pointwise MSE
- **Pointwise:**

$$\mathcal{L}(Q,P^{+},P^{-}) = \text{MSE}(M_s(Q,P^{+}),M_t(Q,P^{+})) + \text{MSE}(M_s(Q,P^{-}),M_t(Q,P^{-}))$$

- **Ranknet:**

$$\mathcal{L}(Q,P^{+},P^{-}) = \text{RankNet}(M_s(Q,P^{+}) - M_s(Q,P^{-})) * ||M_t(Q,P^{+}) - M_t(Q,P^{-})||$$

Margin MSE consistently wins across architectures as best distilling strategy

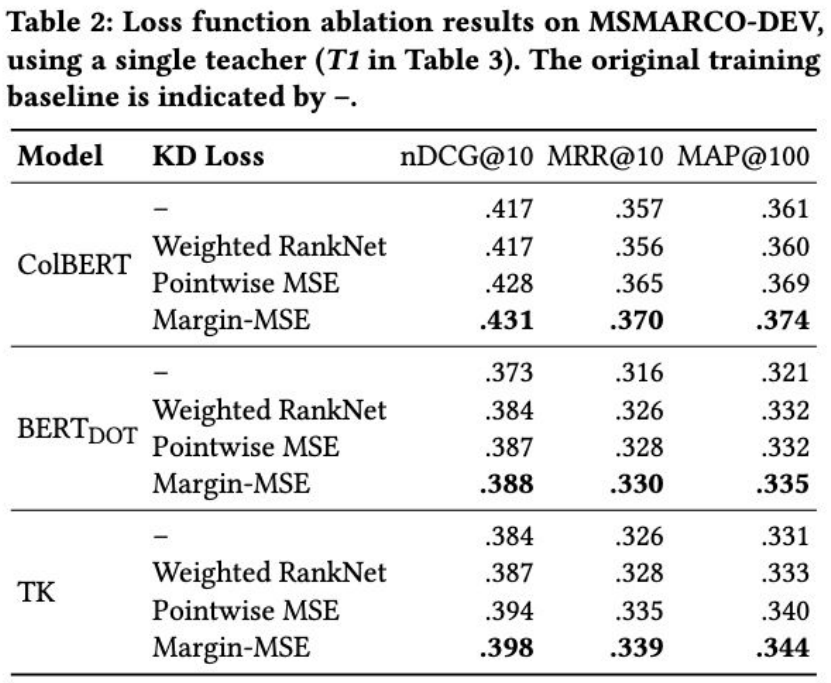
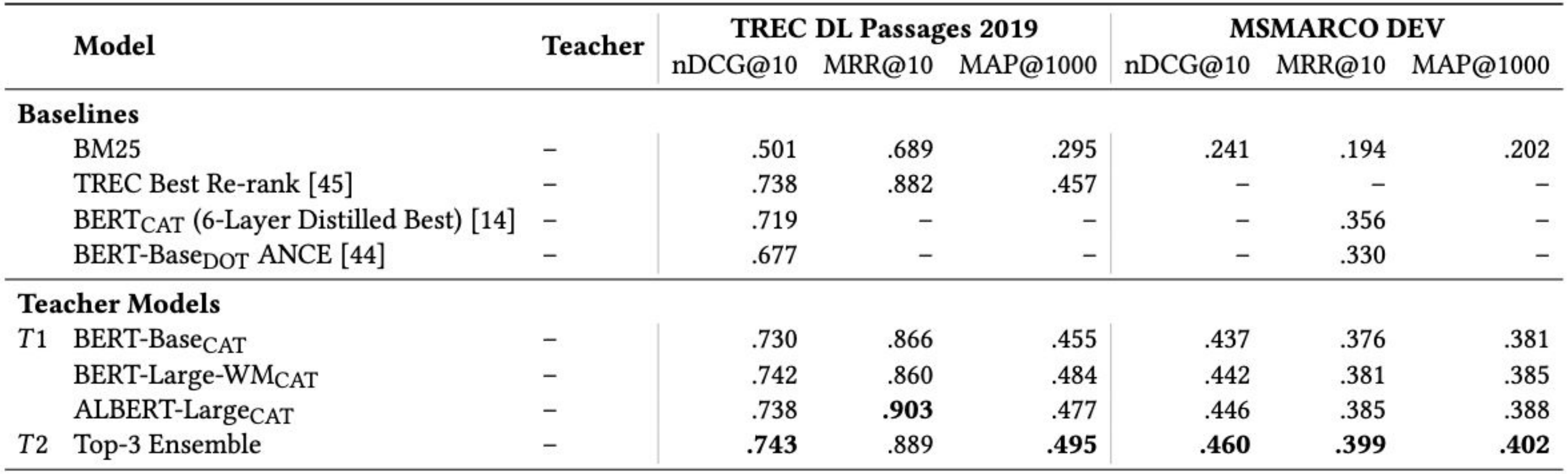


Table 2: Loss function ablation results on MSMARCO-DEV, using a single teacher (*T1* in Table 3). The original training baseline is indicated by –.

Model	KD Loss	nDCG@10	MRR@10	MAP@100
ColBERT	–	.417	.357	.361
	Weighted RankNet	.417	.356	.360
	Pointwise MSE	.428	.365	.369
	Margin-MSE	**.431**	**.370**	**.374**
BERT_{DOT}	–	.373	.316	.321
	Weighted RankNet	.384	.326	.332
	Pointwise MSE	.387	.328	.332
	Margin-MSE	**.388**	**.330**	**.335**
TK	–	.384	.326	.331
	Weighted RankNet	.387	.328	.333
	Pointwise MSE	.394	.335	.340
	Margin-MSE	**.398**	**.339**	**.344**

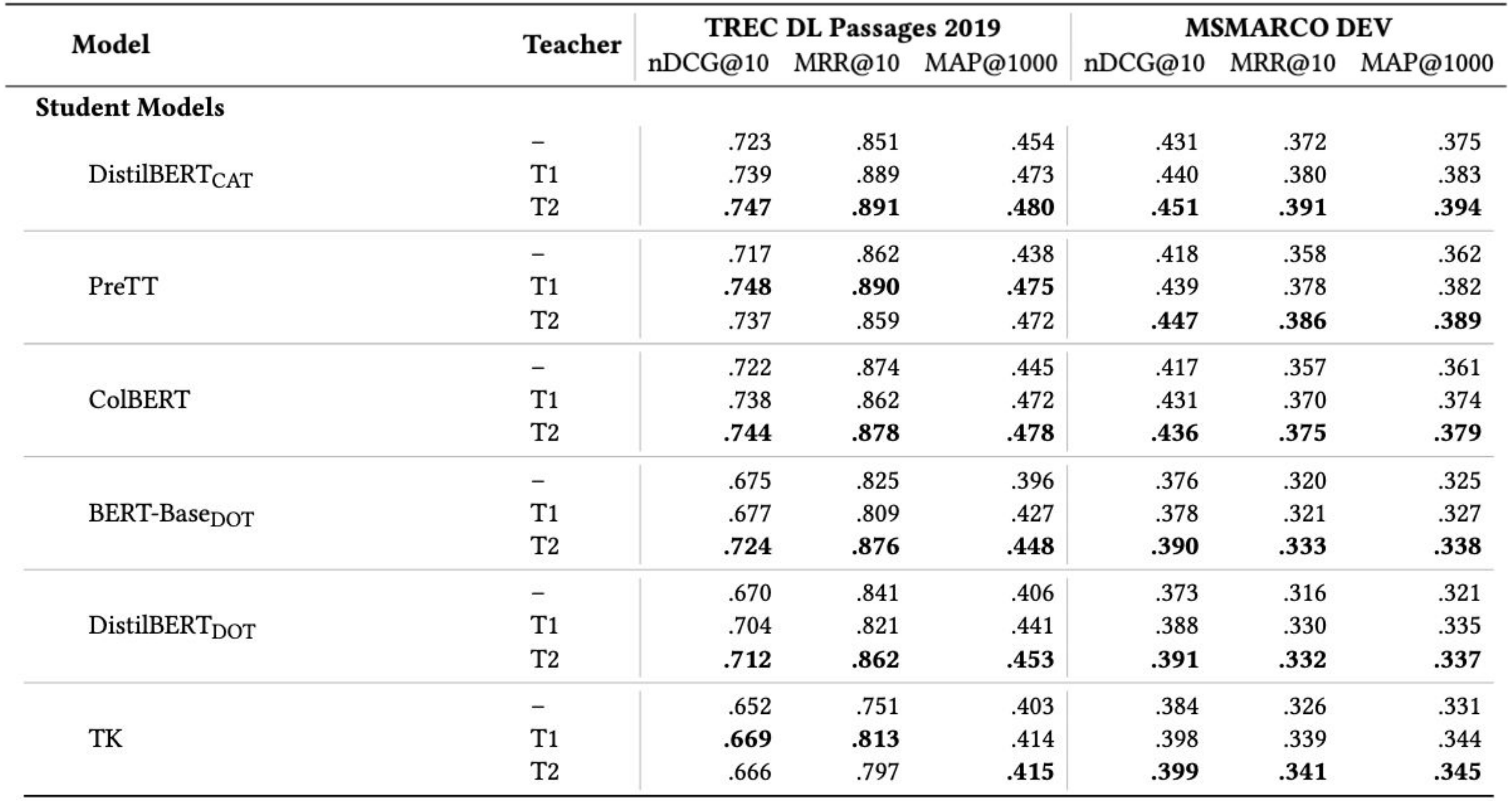
Results: Single teacher vs ensemble of teachers

- First, we look at the overall results for every baseline or teacher candidate
- Notice top-3 ensemble almost always is best?
- Can we leverage these as multiple teacher signals in distillation?



	Model	Teacher	TREC DL Passages 2019			MSMARCO DEV		
			nDCG@10	MRR@10	MAP@1000	nDCG@10	MRR@10	MAP@1000
Baselines								
	BM25	–	.501	.689	.295	.241	.194	.202
	TREC Best Re-rank [45]	–	.738	.882	.457	–	–	–
	BERT_{CAT} (6-Layer Distilled Best) [14]	–	.719	–	–	–	.356	–
	$\text{BERT-Base}_{\text{DOT}}$ ANCE [44]	–	.677	–	–	–	.330	–
Teacher Models								
*T*1	$\text{BERT-Base}_{\text{CAT}}$	–	.730	.866	.455	.437	.376	.381
	$\text{BERT-Large-WM}_{\text{CAT}}$	–	.742	.860	.484	.442	.381	.385
	$\text{ALBERT-Large}_{\text{CAT}}$	–	.738	**.903**	.477	.446	.385	.388
*T*2	Top-3 Ensemble	–	**.743**	.889	**.495**	**.460**	**.399**	**.402**

Results: Single teacher vs ensemble of teachers



Model	Teacher	TREC DL Passages 2019 nDCG@10	TREC DL Passages 2019 MRR@10	TREC DL Passages 2019 MAP@1000	MSMARCO DEV nDCG@10	MSMARCO DEV MRR@10	MSMARCO DEV MAP@1000
Student Models							
DistilBERT$_{CAT}$	–	.723	.851	.454	.431	.372	.375
	T1	.739	.889	.473	.440	.380	.383
	T2	**.747**	**.891**	**.480**	**.451**	**.391**	**.394**
PreTT	–	.717	.862	.438	.418	.358	.362
	T1	**.748**	**.890**	**.475**	.439	.378	.382
	T2	.737	.859	.472	**.447**	**.386**	**.389**
ColBERT	–	.722	.874	.445	.417	.357	.361
	T1	.738	.862	.472	.431	.370	.374
	T2	**.744**	**.878**	**.478**	**.436**	**.375**	**.379**
BERT-Base$_{DOT}$	–	.675	.825	.396	.376	.320	.325
	T1	.677	.809	.427	.378	.321	.327
	T2	**.724**	**.876**	**.448**	**.390**	**.333**	**.338**
DistilBERT$_{DOT}$	–	.670	.841	.406	.373	.316	.321
	T1	.704	.821	.441	.388	.330	.335
	T2	**.712**	**.862**	**.453**	**.391**	**.332**	**.337**
TK	–	.652	.751	.403	.384	.326	.331
	T1	**.669**	**.813**	.414	.398	.339	.344
	T2	.666	.797	**.415**	**.399**	**.341**	**.345**

Results: Using these distilled models for dense retrieval

- Authors demonstrate baseline-beating retrieval results with their multi-teacher margin-distilled student LMs in dense nearest neighbor retrieval.

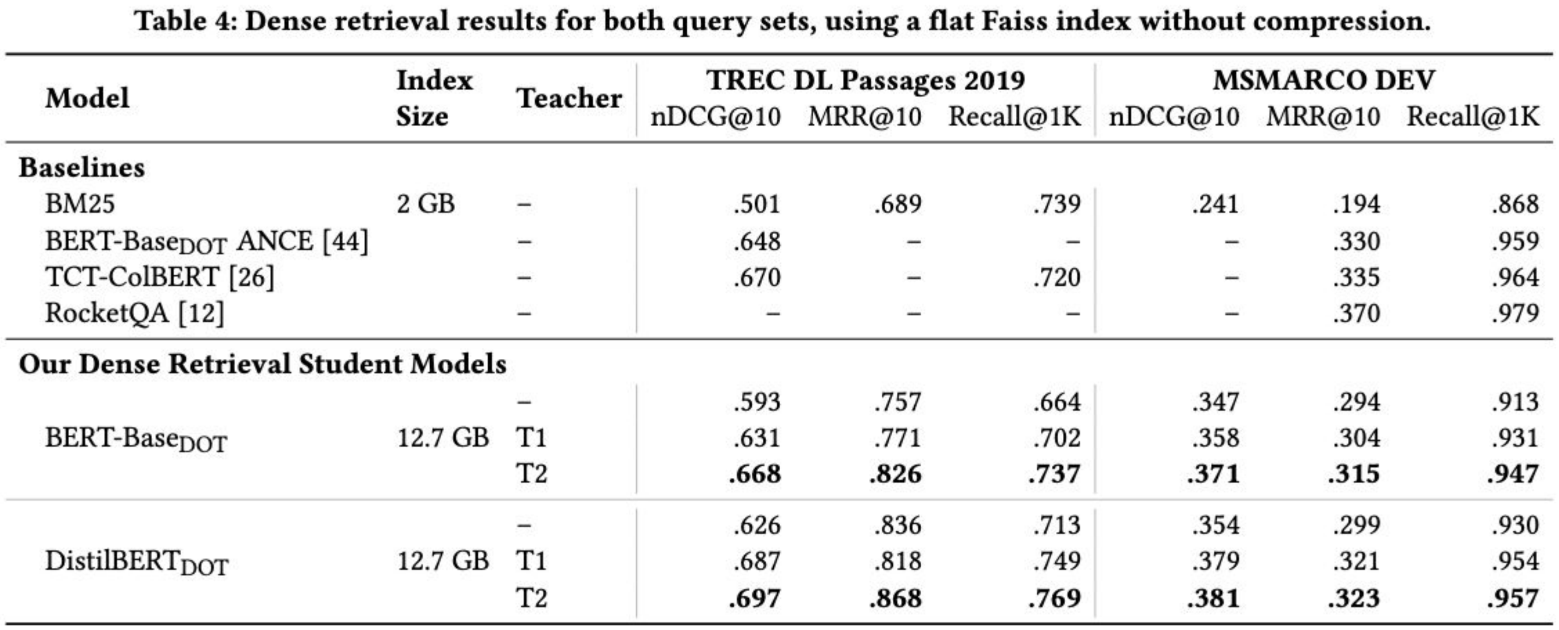


Table 4: Dense retrieval results for both query sets, using a flat Faiss index without compression.

Model	**Index Size**	**Teacher**	**TREC DL Passages 2019** nDCG@10	MRR@10	Recall@1K	**MSMARCO DEV** nDCG@10	MRR@10	Recall@1K
Baselines								
BM25	2 GB	–	.501	.689	.739	.241	.194	.868
BERT-Base$_{DOT}$ ANCE [44]		–	.648	–	–	–	.330	.959
TCT-ColBERT [26]		–	.670	–	.720	–	.335	.964
RocketQA [12]		–	–	–	–	–	.370	.979
Our Dense Retrieval Student Models								
BERT-Base$_{DOT}$	12.7 GB	–	.593	.757	.664	.347	.294	.913
		T1	.631	.771	.702	.358	.304	.931
		T2	**.668**	**.826**	**.737**	**.371**	**.315**	**.947**
DistilBERT$_{DOT}$	12.7 GB	–	.626	.836	.713	.354	.299	.930
		T1	.687	.818	.749	.379	.321	.954
		T2	**.697**	**.868**	**.769**	**.381**	**.323**	**.957**

Results: Closing Efficiency-Effectiveness Gap

- The authors test latency assuming 1000 cached document reps are in memory, and test interaction speed for reranking for a single query against those docs

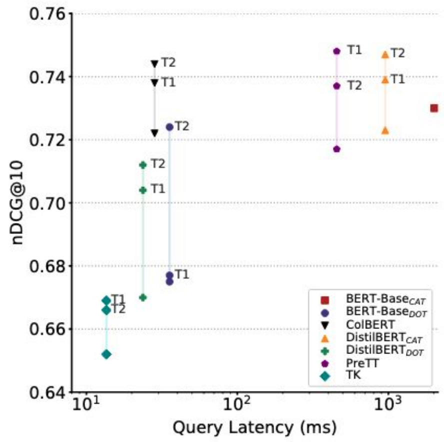


Figure 3: Query latency vs. nDCG@10 on TREC'19

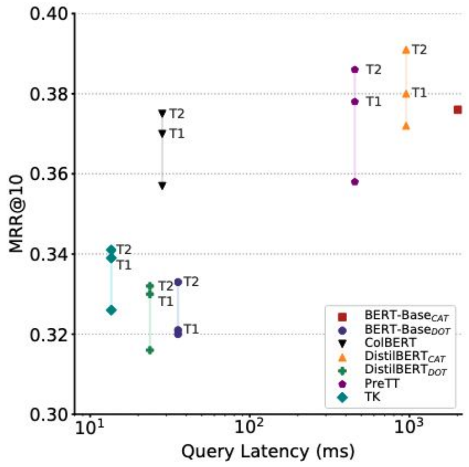


Figure 4: Query latency vs. MRR@10 on MSMARCO DEV

Teacher Analysis

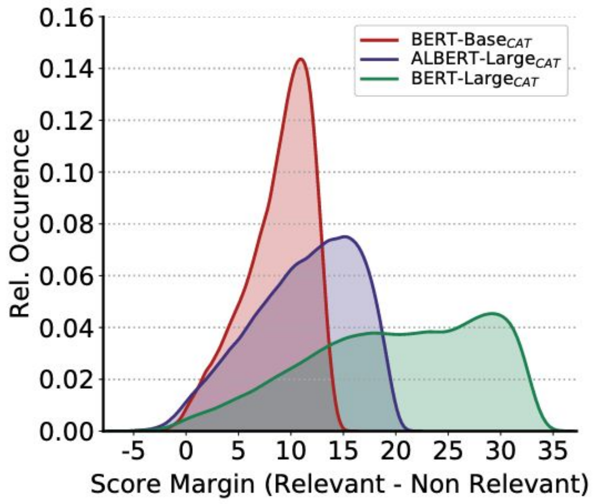


Figure 5: Distribution of the margins between relevant and non-relevant documents of the three teacher models on MS MARCO-Passage training data

- Different teachers exhibit different distributions of score margins (Relevant-non-relevant)
- Does this cause a difference in how rankings are marginally changed by single examples?

Teacher Analysis

- Sort of.
- This plot shows that examples will tend to have a stronger "pull" on the model at each step from ensemble (T2) than single (T1) teacher method
- However, this pull can both be positive (left side) and negative (right side)

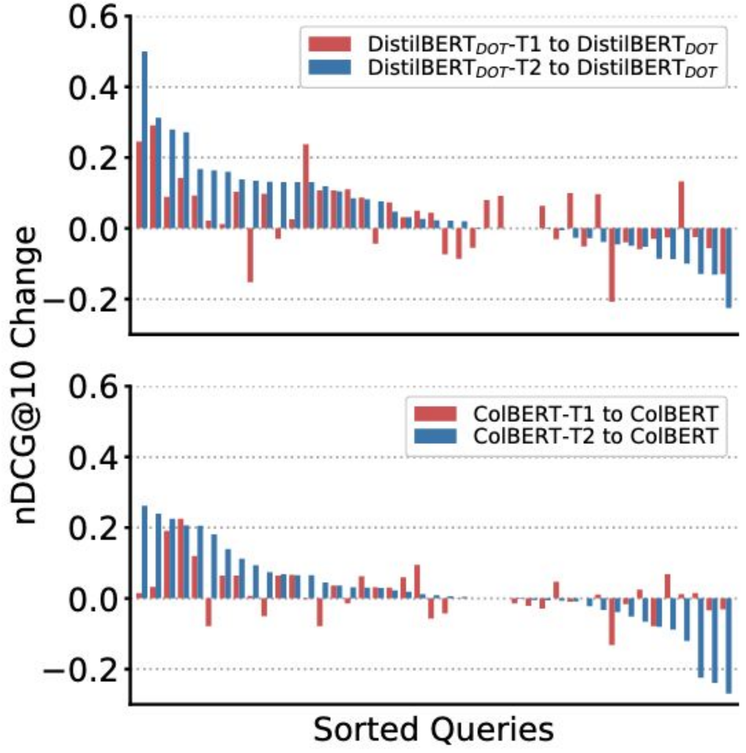


Figure 6: A detailed comparison between T1 and T2 training ndcg@10 changes per query of the TREC-DL'19 query set

Conclusion

- Margin-based distillation enables multi-teacher cross-model knowledge distillation in IR
- Multi-teacher tends to outperform single teacher across a variety of target architectures/strategies, particularly in MAP@100
- For some of the architectures and metrics, not only only is an **order of magnitude latency improvement** achieved, but also **accuracy is improved** in the student models over the baselines
    - E.g., ColBERT is ~100x faster than BERT-BaseCAT, but also has a higher NDCG@10
- We are in the process of replicating these results and trying to further expand the samplewise teacher analysis