Tevatron: An Efficient and Flexible Toolkit for Neural Retrieval

Luyu Gao*
Carnegie Mellon University
Pittsburgh, United States
luyug@cs.cmu.edu

Xueguang Ma
University of Waterloo
Waterloo, Canada
x93ma@uwaterloo.ca

Jimmy Lin
University of Waterloo
Waterloo, Canada
jimmylin@uwaterloo.ca

Jamie Callan
Carnegie Mellon University
Pittsburgh, United States
callan@cs.cmu.edu

ABSTRACT

Recent rapid advances in deep pre-trained language models and the introduction of large datasets have powered research in embedding-based neural retrieval. While many excellent research papers have emerged, most of them come with their own implementations, which are typically optimized for some particular research goals instead of efficiency or code organization. In this paper, we introduce Tevatron, a neural retrieval toolkit that is optimized for efficiency, flexibility, and code simplicity. Tevatron enables model training and evaluation for a variety of ranking components such as dense retrievers, sparse retrievers, and rerankers. It also provides a standardized pipeline that includes text processing, model training, corpus/query encoding, and search. In addition, Tevatron incorporates well-studied methods for improving retriever effectiveness such as hard negative mining and knowledge distillation. We provide an overview of Tevatron in this paper, demonstrating its effectiveness and efficiency on multiple IR and QA datasets. We highlight Tevatron's flexible design, which enables easy generalization across datasets, model architectures, and accelerator platforms (GPUs and TPUs). Overall, we believe that Tevatron can serve as a solid software foundation for research on neural retrieval systems, including their design, modeling, and optimization.

CCS CONCEPTS

• **Information systems → Retrieval models and ranking**.

KEYWORDS

Neural IR, Dense Retrieval, Sparse Retrieval, Toolkit

ACM Reference Format:
Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2023. Tevatron: An Efficient and Flexible Toolkit for Neural Retrieval. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '23), July 23–27, 2023, Taipei, Taiwan.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3539618.3591805

1 INTRODUCTION

The popularity of neural retrieval in the research community has greatly grown in the past few years, from neural rerankers based on the cross-encoder architecture to the most recent neural retrievers based on the bi-encoder architecture [6, 13, 16, 22, 25]. However, many recent research papers [13, 25] have focused on developing their own software stacks with specialized support only for specific datasets and models. This approach can be limiting as it does not allow for flexible generalization across different models and datasets. To address these issues, Tevatron provides researchers with access to state-of-the-art models and makes it easy for them to start a new research project on a new dataset.

Taking dense retrieval as an example: in our past papers [6, 7, 18], we have run into several engineering challenges. For example, in terms of resources, large corpora and training sets require a large amount of CPU memory; accelerator (GPU/TPU) memory usage also grows with model size. While orthogonal to actual research, these engineering hurdles slow down and constrain researchers, especially those with limited hardware resources. With Tevatron, we aim at providing a unified solution to common engineering problems. To address these challenges, Tevatron provides a unified framework that incorporates several popular and widely used open-source packages. These packages include datasets, transformers, FAISS, and Pyserini, which respectively serve as the backbone for data management, neural network modeling, embedding-based retrieval engines, and evaluation metrics.

In addition, to accommodate different research needs, we select two deep learning frameworks for Tevatron, PyTorch [21] and JAX [3]. PyTorch's eager execution patterns and intuitive object-oriented design have gained it a massive user base in the research community. On the other hand, JAX, backed by just-in-time (JIT) XLA compilation, offers smooth transitions across hardware stacks with optimized performance.

Overall, Tevatron aims to streamline and simplify the research process, enabling researchers to focus on their actual research problems rather than engineering challenges. Additionally, Tevatron enables collaboration and ensures reproducibility, as others can easily reproduce the same experiments using the same packages.

The rest of the paper is organized as follows: Section 2 gives an overview of Tevatron. Section 3 demonstrates usage of Tevatron's command-line interface. Section 4 shares experimental results of running Tevatron with various models and datasets.

2 TOOLKIT OVERVIEW

Tevatron[1] is packaged as a Python module available on the Python Package Index. The toolkit can be installed via `pip`, as follows:

```
$ pip install tevatron==0.1.0
```

In this section, we provide an overview of the core components of Tevatron. We demonstrate how these components support the full pipeline of data preparation, training, encoding, and search, respectively. Code and documentation are available at `tevatron.ai`.

*The first two authors contributed equally.

[1] http://tevatron.ai

```
{
   "query_id": "<query id>",
   "query":    "<query text>",
   "positive_passages": [
     {"docid": "<passage id>",
      "title": "<passage title>",
      "text":  "<passage body>"}, ...
   ],
   "negative_passages": [...]
}
```

Figure 1: Tevatron raw data template for IR datasets.

2.1 Data Management

Having data ready to use is a critical preliminary step before training or encoding starts. Data access overhead and constraints can directly affect training/encoding performance. In Tevatron, we adopt the following core design: (1) text data are pre-tokenized before training or encoding occurs, (2) tokenized data are memory-mapped instead of lazy-loaded or residing in-memory. The former avoids overhead when running sub-word/piece level tokenizers and also reduces data traffic compared to raw text. The latter allows random data access in the training/encoding loop without consuming a large amount of physical memory.

Tevatron defines raw input format templates, shown in Fig. 1. We organize a training instance into an anchor query, a list of positive target texts, and a list of negative target texts. The positive targets are usually human-labeled and the negative targets are usually non-relevant texts from the top results of a baseline retrieval system such as BM25 [6, 13].

Users can pass raw data file pointers and processing specifications to Tevatron's dataset classes (`HFTrainDataset` for training, `HFQueryDataset` and `HFCorpusDataset` for encoding), which will perform fast parallel data formatting and tokenization. Processed data is internally represented as a `datasets.Dataset` object and is stored in the Apache Arrow format, which can be memory-mapped and randomly accessed by offset.

For researchers who are focusing on building new models, we make a collection of popular open-access datasets self-contained within the Tevatron toolkit. For instance, with a single command, one can load the training set of MS MARCO. Under the hood, Tevatron will first download the raw data set we hosted through HuggingFace.[2] Then it will run the corresponding pre-defined pre-processing script to format and tokenize the downloaded data.

2.2 Neural Encoder Models

Tevatron's model class `EncoderModel` is a PyTorch `nn.Module` subclass that defines the encoder in a standard bi-encoder architecture, which converts queries and passages individually into vector representations. Based on the type of the retrieval model, the representation can be a low-dimensional dense vector or a high-dimensional sparse vector. Functionally, the class wraps an underlying Transformer model and provides methods for text encoding and loss computation. Thanks to duck typing in Python, the `EncoderModel` class supports models in the HuggingFace transformers library that

```
# initialize model
model = DenseModel.build(model_args)
# initialize dataset
train_dataset = HFTrainDataset(data_args).process()
train_dataset = TrainDataset(data_args, train_dataset)
# initialize trainer
trainer = GCTrainer if training_args.grad_cache \
                    else TevatronTrainer
trainer = trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    data_collator=QPCollator(data_args),
)
# start training
trainer.train()
```

Figure 2: Training process of Tevatron (PyTorch).

extend the standard base model. This means new Transformer models can be loaded into Tevatron as soon as they are available in the transformers model hub. This design helps our toolkit reduce code duplication. The `EncoderModel` class also handles loss computation during training. It implements a contrastive loss with in-batch negatives and can perform negative sharing across devices using the collective communication technique introduced in the NVIDIA NCCL library.[3]

To complete the neural retriever training setup, we introduce the `TevatronTrainer` class that implements miscellaneous training utilities. It controls basic setup such as the batch size and the number of training epochs. When running on multiple GPUs, the trainer will properly set up distributed training and wrap models for gradient reduction. In Tevatron, we also implement the `GCTrainer` class, which is a subclass of `TevatronTrainer` that uses gradient caching to support large batch training on memory-limited devices [9].

Taking dense retrieval as an example, `DenseModel` is a subclass of `EncoderModel`, which encodes text into dense vector representations. Representative dense retrieval models such as DPR [13] and ANCE [25] can be implemented using this class. By combining the data processor, dense retrieval model, and trainer, Tevatron abstracts the training loop of a dense retrieval model into the code block shown in Fig. 2. The training process is defined similarly for neural sparse retrieval models (e.g., SPLADE [5], uniCOIL [17]).

2.3 Vector Search

At a high level, the bi-encoder model produces textual representations for both documents and queries. In order to perform end-to-end retrieval, Tevatron implements indexing and search modules for vector search (i.e., top-k retrieval) that rely on either dense or sparse vector representations. These modules are facilitated by the use of existing libraries.

For dense retrieval, we use the FAISS library [11] as our backend for top-k retrieval. It implements several efficient indexes in C++ and exposes them through Python interfaces. For users who want the best effectiveness, Tevatron provides a simple class called `BaseFaissIPRetriever` that wraps the `faiss.IndexFlatIP` flat

[2]https://huggingface.co/tevatron

[3]https://developer.nvidia.com/nccl

index for exact search. Those who wish to trade off efficiency and effectiveness can use the more powerful `FaissRetriever` class, which takes an additional `index_spec` string argument in its initialization method and uses `faiss.index_factory` method to build custom indexes. Users can take advantage of this interface to build approximate search indexes like HNSW [19] or PQ [10].

For sparse retrieval, Tevatron directly utilizes Pyserini [15] to build inverted indexes for conducting efficient lexical search.

2.4 Additional Modules

JAX. Tevatron has a sub-package `tevax` that implements core functionality for JAX. Following JAX's functional nature [3], `tevax` is designed with a different philosophy. We define loss functions that can be *composed* with other JAX transformations. In practice, they can be combined with Flax models in the HuggingFace transformer library for dense retriever training. With JAX as the backend, `tevax` makes it possible for a single piece of code to run on a single GPU, multiple GPUs, or TPUs.

Rerankers. Neural rerankers based on a cross-encoder architecture are commonly used in a rerank stage to further improve retrieval effectiveness. In Tevatron, we implement reranker training and inference based on the same data processing pipeline as retrievers. The reranker training implements the Localized Contrastive Estimation technique proposed by Gao et. al. [8].

Distillation. Distilling knowledge from rerankers to retrievers is a commonly used technique to improve the effectiveness of retrieval models. We implement knowledge distillation by using a reranker to assign soft labels to the positive and hard negative passages for each query. The retriever learns from the reranker teacher by optimizing KL divergence.

3 TOOLKIT USAGE

On top of the various core components, Tevatron provides a set of command-line interfaces (CLIs) to drive the retrieval pipeline. With a flexible design in data management and support for neural retrievers, one can conduct many types of research without writing code. In this section, we provide an example using the Tevatron CLI to run the previously discussed components to learn a model and perform open-domain retrieval on Natural Questions (NQ) [14]. We use dense retrieval as an example.

With Tevatron, we can replicate the training of a DPR model for the NQ dataset (details in Section 4) with a single command:

```
python -m tevatron.driver.train \
    --dataset_name Tevatron/wikipedia-nq \
    --model_name_or_path bert-base-uncased \
    --per_device_train_batch_size 128 \
    --train_n_passages 2 \
    --num_train_epochs 40 \
    --learning_rate 1e-5 \
    --fp16 \
    --grad_cache \
    --output_dir model-nq
```

As introduced in Section 2.1, Tevatron will automatically handle the downloading and pre-processing of our self-contained training data `Tevatron/wikipedia-nq`. Then, the pre-processed dataset and initialized `DenseModel` will be fed into `GCTrainer` class, as shown in Figure 2. Since the above command enables the `grad_cache` option, it will use `GCTrainer` during training. Here we also enable mixed precision training [20] via `-fp16` to improve efficiency.

	NQ	TriviaQA	SQuAD	CuratedTrec	WebQuestion
DPR [13]	78.4 / 85.4	79.4 / 85.0	63.2 / 77.2	79.8 / 89.1	73.2 / 81.4
Tevatron	79.8 / 86.9	80.2 / 85.5	62.3 / 77.0	84.0 / 90.7	75.4 / 82.9

Table 1: Top-20/top-100 retrieval accuracy of DPR model replication on five open-domain QA datasets.

Besides training data, Tevatron also packages corresponding corpus data for each dataset. Again, we simplify corpus encoding into a single command:

```
python -m tevatron.driver.encode \
    --model_name_or_path model-nq \
    --dataset_name Tevatron/wikipedia-nq-corpus \
    --encoded_save_path corpus-emb-00.pkl \
    --encode_num_shard 20 \
    --encode_shard_index 00 \
    --fp16
```

As encoding the entire corpus within a single process may be inefficient, Tevatron support encoding the corpus by sharding. For example, the above command encodes the first 1/20 split of the entire corpus. Users can easily run multiple processes for multiple shards in parallel to speed up the encoding process.

With the query and corpus embeddings, we can run retrieval with the following command:

```
python -m tevatron.faiss_retriever \
    --query_reps query.pkl \
    --passage_reps corpus-emb-*.pkl \
    --depth 100 \
    --batch_size -1 \
    --save_text \
    --save_ranking_to result.txt
```

where `-batch_size` controls the number of queries passed to the FAISS index for each search call and `-1` will pass all queries in one call for efficient parallel search.

4 EXPERIMENTS

In this section, we demonstrate the effectiveness and efficiency of Tevatron using DPR [13], which is among the first studies to show that text retrieval using learned dense representations outperforms text retrieval using "traditional" sparse representations (e.g., BM25) on open-domain question-answering tasks.

We evaluate the effectiveness of Tevatron by replicating retrieval results on QA tasks [2, 12, 14, 23, 24] reported in the original DPR paper [13]. We compare the models trained under the "Single" setting defined in the original paper, where each model is trained using the corresponding individual dataset. Following similar hyperparameters settings, we train the models with a learning rate of 1e-5 for 40 epochs with batch size 128. In Table 1, we see that Tevatron produces slightly higher top-k accuracy on four datasets compared to the original DPR paper, although the values are slightly lower on SQuAD. Overall, we conclude that this is a successful replication and that the Tevatron pipeline is effective.

	RAM	GPU memory	Time
DPR-repo	60G	20G × 4	2.0 hours
Tevatron-default	17G	17G × 4	1.5 hours
Tevatron-GradCache	4G	15G × 1	7.0 hours
Tevatron-TPU	10G	–	1.0 hours

Table 2: Training efficiency comparison between the original DPR repo and different settings with Tevatron.

We demonstrate the efficiency of Tevatron by comparing it with code from the original DPR repo[4] on three dimensions: RAM usage, GPU memory usage, and training time. The experiments are conducted on a machine with NVIDIA A100 GPUs. In both DPR-repo and Tevatron-default settings, we train the dense retrieval model on 4 GPUs in the distributed data-parallel mode of PyTorch. By comparing the first two rows in Table 2, we see that Tevatron is more efficient along all three dimensions than the original codebase: Tevatron consumes 3/4 less RAM, 12G less GPU memory, and is 1/4 faster. This means that given the same resources, Tevatron has the potential to support more training data, larger batch sizes, and faster training.

The gradient caching feature of Tevatron can further improve GPU memory efficiency [9]. DPR training requires batch size 128 to obtain the level of retrieval accuracy reported above. With the original DPR repo, users cannot train a model with a sufficiently large batch size if GPU resources are limited, which will result in a drop in retrieval accuracy. Tevatron provides users the option to train dense retrievers using limited GPU resources, retaining the same batch size for each optimization step. To illustrate this, we conduct experiments with Tevatron-GradCache on a single GPU. Tevatron-GradCache trains dense retrievers by splitting each batch of size 128 into sub-batches of size 32. By conducting two rounds of forward steps [9], the model update step of Tevatron-GradCache is mathematically equivalent to Tevatron-default. This experiment only consumes 4G RAM and 15G GPU memory, to train a DPR model on the NQ dataset with the desired equivalent batch size (but at the cost of a longer training time). By reducing the sub-batch size, Tevatron-GradCache can save even more GPU memory.

We also evaluate the performance of training a dense retrieval model using the JAX backend of Tevatron on a V3-8 TPU VM. When DPR [13] was first published, it costs around a day to train the model on the NQ dataset using the original DPR repo with 8× NVIDIA V100-large GPUs (as noted by the DPR authors on their GitHub page). Now, it is exciting to see that the same can be accomplished within one hour with Tevatron.

To further show the flexibility of the Tevatron toolkit across model architectures, accelerator platforms, and languages, we train multiple dense retrieval baselines on the MS MARCO passage ranking task and the XOR-TyDi cross-lingual retrieval task [1, 4]. Results are shown in Table 3 and Table 4, respectively.

Table 5 demonstrates the effectiveness of teaching dense retrievers with a cross-encoder reranker. Notice that both reranker training and knowledge distillation are integrated into Tevatron and share the same data management pipeline. Specifically, we first train a reranker with the default MS MARCO training data. Then we use the reranker to teach a dense retriever student by optimizing KL divergence. Both retriever and reranker models are initialized with `bert-base-uncased`. We can see that the dense retriever student achieves higher effectiveness than the one trained with hard labels without distillation. Recent studies [5, 22, 26] demonstrate that combining hard negative mining and distillation is able to bring further effectiveness improvements, which can also be easily run in our Tevatron pipeline.

[4]https://github.com/facebookresearch/DPR; to be clear, the efficiency results are based on the master branch on 2022-02-12.

Model	MRR@10	TPU time	GPU time
1. `distilbert-base-uncased`	0.316	1.0 hours	1.5 hours
2. `bert-base-uncased`	0.322	2.0 hours	3.0 hours
3. `co-condenser-marco`	0.357	2.0 hours	3.0 hours
4. `bert-large-uncased`	0.327	6.0 hours	7.5 hours
5. `roberta-large`	0.339	6.0 hours	7.5 hours
6. `roberta-large` + HN	0.361	8.0 hours	10.0 hours
7. `co-condenser-marco` + HN	0.382	3.0 hours	4.0 hours

Table 3: Results of training dense retrievers using Tevatron with different Transformer backbones on the MS MARCO passage ranking task. The models are trained on 4× A100 GPU or V3-8 TPU with a learning rate of 5e-6 and batch size 64 for 3 epochs. HN indicates hard negative mining.

	Ar	Bn	Fi	Ja	Ko	Ru	Te	avg
mDPR	50.4	57.7	58.9	37.3	42.8	44.0	44.9	48.0
Tevatron	50.5	64.1	57.3	41.9	60.4	48.5	58.4	54.4

Table 4: Recall@5kt of mDPR replication on the dev set of the `XOR-RETRIEVE` task with Tevatron. The baseline model replicated with Tevatron improves on average 6 points over the original results on seven languages.

Model	MRR@10
Dense Retriever	0.322
Dense Retriever + Distil.	0.354

Table 5: Results of training a dense retriever with and without knowledge distillation from a reranker on the MS MARCO passage ranking task.

5 CONCLUSION

This paper introduces Tevatron, an efficient and flexible toolkit for training and running neural retrievers with Transformers. The toolkit has a modularized design for easy research exploration and a set of command-line interfaces for fast development and evaluation. Our experiments show that Tevatron can be used to train neural retrieval models effectively and efficiently. These flexible and generalizable functionalities provide the IR community convenience in future neural retrieval research.

ACKNOWLEDGMENTS

This research was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada. The authors would like to thank Google's TPU Research Cloud (TRC) for access to Cloud TPUs and the anonymous reviewers for their feedback.

REFERENCES

[1] Akari Asai, Jungo Kasai, Jonathan Clark, Kenton Lee, Eunsol Choi, and Hannaneh Hajishirzi. 2021. XOR QA: Cross-lingual Open-Retrieval Question Answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online, 547–564.

[2] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic Parsing on Freebase from Question-Answer Pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, 1533–1544.

[3] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. *JAX: composable transformations of Python+NumPy programs*. Google.

[4] Jonathan H. Clark, Eunsol Choi, Michael Collins, Dan Garrette, Tom Kwiatkowski, Vitaly Nikolaev, and Jennimaria Palomaki. 2020. TyDi QA: A Benchmark for Information-Seeking Question Answering in Typologically Diverse Languages. *Transactions of the Association for Computational Linguistics* 8 (2020), 454–470.

[5] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021. SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '21)*. 2288–2292.

[6] Luyu Gao and Jamie Callan. 2021. Condenser: a Pre-training Architecture for Dense Retrieval. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic, 981–993.

[7] Luyu Gao and Jamie Callan. 2022. Unsupervised Corpus Aware Language Model Pre-training for Dense Passage Retrieval. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland, 2843–2853.

[8] Luyu Gao, Zhuyun Dai, and Jamie Callan. 2021. Rethink Training of BERT Rerankers in Multi-Stage Retrieval Pipeline. In *Advances in Information Retrieval: 43rd European Conference on IR Research (ECIR 2021), Part II*. 280–286.

[9] Luyu Gao, Yunyi Zhang, Jiawei Han, and Jamie Callan. 2021. Scaling Deep Contrastive Learning Batch Size under Memory Limited Setup. In *Proceedings of the 6th Workshop on Representation Learning for NLP (RepL4NLP-2021)*. Online, 316–321.

[10] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33 (2011), 117–128.

[11] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

[12] Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada, 1601–1611.

[13] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online, 6769–6781.

[14] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural Questions: A Benchmark for Question Answering Research. *Transactions of the Association for Computational Linguistics* 7 (2019), 452–466.

[15] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021. Pyserini: A Python Toolkit for Reproducible Information Retrieval Research with Sparse and Dense Representations. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, Canada) *(SIGIR '21)*. 2356–2362.

[16] Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. 2021. In-Batch Negatives for Knowledge Distillation with Tightly-Coupled Teachers for Dense Retrieval. In *Proceedings of the 6th Workshop on Representation Learning for NLP (RepL4NLP-2021)*. Association for Computational Linguistics, Online, 163–173.

[17] Xueguang Ma, Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. 2022. Document Expansion Baselines and Learned Sparse Lexical Representations for MS MARCO V1 and V2. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22)*. 3187–3197.

[18] Xueguang Ma, Kai Sun, Ronak Pradeep, Minghan Li, and Jimmy Lin. 2022. Another Look at DPR: Reproduction of Training and Replication of Retrieval. In *Advances in Information Retrieval: 44th European Conference on IR Research, ECIR 2022, Part I* (Stavanger, Norway). 613–626.

[19] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2020), 824–836.

[20] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed Precision Training. In *Proceedings of the 6th International Conference on Learning Representations*.

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. 8024–8035.

[22] Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. 2021. RocketQA: An Optimized Training Approach to Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online, 5835–5847.

[23] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas, 2383–2392.

[24] Ellen M. Voorhees and Dawn M. Tice. 2000. The TREC-8 Question Answering Track. In *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC'00)*. Athens, Greece.

[25] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval. In *Proceedings of the 9th International Conference on Learning Representations (ICLR 2021)*.

[26] Hansi Zeng, Hamed Zamani, and Vishwa Vinay. 2022. Curriculum Learning for Dense Retrieval Distillation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22)*.