The Journal of Open Source Software

DeBIR: A Python Package for Dense Bi-Encoder Information Retrieval

Vincent Nguyen [1,2], **Sarvnaz Karimi** [2], and **Zhenchang Xing** [1,2]

1 Australian National University **2** CSIRO's Data61

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

Information Retrieval (IR) is the task of retrieving documents given a query or information need. These documents are retrieved and ranked based on a relevance function or relevance model such as Best-Matching 25 (Robertson et al., 1995). Although deep learning has been successful in other computer science fields such as computer vision (Krizhevsky et al., 2012; Szegedy et al., 2014) and natural language processing (Devlin et al., 2019; Lee et al., 2019; Yang Liu & Lapata, 2019), success in information retrieval was limited until 2021 (Lin, 2019). In 2021, deep learning in information retrieval could surpass less computationally intensive keyword-based statistical models in terms of retrieval effectiveness (Yang et al., 2019), sparking the field of dense retrieval. Dense retrieval is the task of retrieving documents given a query or information need using a dense vector representation of the query and documents. The dense vector representation is obtained by passing the query and documents through a neural network. The neural network is usually a pre-trained language model such as BERT (Devlin et al., 2019) or RoBERTa (Yinhan Liu et al., 2019). The dense query vector representation is then used to retrieve documents using a similarity function such as cosine similarity.

Unlike statistical learning, tuning of deep learning retrieval methods is often costly and time-consuming. This makes it important to efficiently automate much of the training, tuning and evaluation processes.

DeBIR is a library for facilitating dense retrieval research, with a primary focus on bi-encoder dense retrieval where query and documents dense vectors are generated separately (Reimers & Gurevych, 2019). It allows for expedited experimentation in dense retrieval research by reducing boilerplate code through an interchangeable pipeline API and code extendability through inheritance of general classes. It further abstracts common training loops and hyperparameter tuning into easy-to-define configuration files. This library is aimed at helping practitioners, researchers and data scientists experimenting with providing them dense retrieval methods that are easy-to-use out of the box but also have additional hackability for more nuanced research.

A brief summary of the pipeline stages are:

1. Configuration based on TOML files, these are loaded in a class factory to create pipeline objects.
2. An executor object takes in a query builder object. The purpose of the query builder object is to define the mapping of the documents and which parts of the query to use for query execution.
3. The executor object asynchronously executes the queries.
4. Finally, an evaluator object uses the results for a listing of metrics defined by a configuration file against an oracle test set.

This pipeline is condensed into a single class that can be built from a configuration file.

Statement of Need

Dense retrieval has been popular in Information Retrieval for some time (Guo et al., 2017; Hui et al., 2017; Yin et al., 2015). In the early 2000s, there had been considerable stagnation in retrieval effectiveness as there was a lack of strong baselines (Armstrong et al., 2009) when comparing new methods. This observation was repeated with the rise of deep learning, where retrieval performance was again compared against weaker baselines and were not significantly stronger than older statistical models, such as a well-tuned BM25 model (Yang et al., 2019).

However, this was later recanted when transformer models could be used fine-tuned on Natural Language Inference tasks or ms-marco as a cross-encoder (where a query and document pair are encoded at ranking time) (Lin, 2019), significantly overtaking even the best BM25 models.

Cross-encoder models have demonstrated to be highly succesful. However, the use of bi-encoders is still under-used in the IR field. This is most likely due to the fact that cross-encoders often obtain higher retrieval effectiveness. However, the downside to cross-encoders is the need to encode the query and document pair at ranking time. This is computationally expensive and can be a bottleneck in production systems. Bi-encoders, on the other hand, can be used to encode the query and document separately, and then use a similarity function to retrieve documents. This is computationally cheaper and can be used in production systems. Furthermore, by combining with logarithmically with BM25 models, cross encoders have demonstrated strong retrieval effectiveness in biomedical research retrieval (Nguyen et al., 2022).

DeBIR is a library that mainly facilitates bi-encoder research (where query and document can be encoded independently) and provides base classes with flexible functionality through inheritance. Although we provide re-rankers for cross-encoders, the priority of the library is to facilitate bi-encoder research. Where the strength of bi-encoders lies in the use of offline indexing of dense vectors.

The DeBIR library exposes an API for commonly used functions for training, hyper-parameter tuning (Figure Figure 2 and evaluation of transformer-based models. The pipeline can be broken up into multiple stages: parsing, query building, query execution, serialization and evaluation (Figure Figure 1. Furthermore, we package a caching mechanism for the expensive encoding operations, which can be used to speed up the pipeline during repeated experimentation.

We believe this tool will be useful for facilitating research and rapid experimentation and will continue to improve the library.

Similar libraries that exist include sentence-transformers, openNIR, but have less of a focus on all stages of the dense retrieval pipeline, limited extendability (we provide base classes that can be inherited) or are tailored to general purpose machine learning.

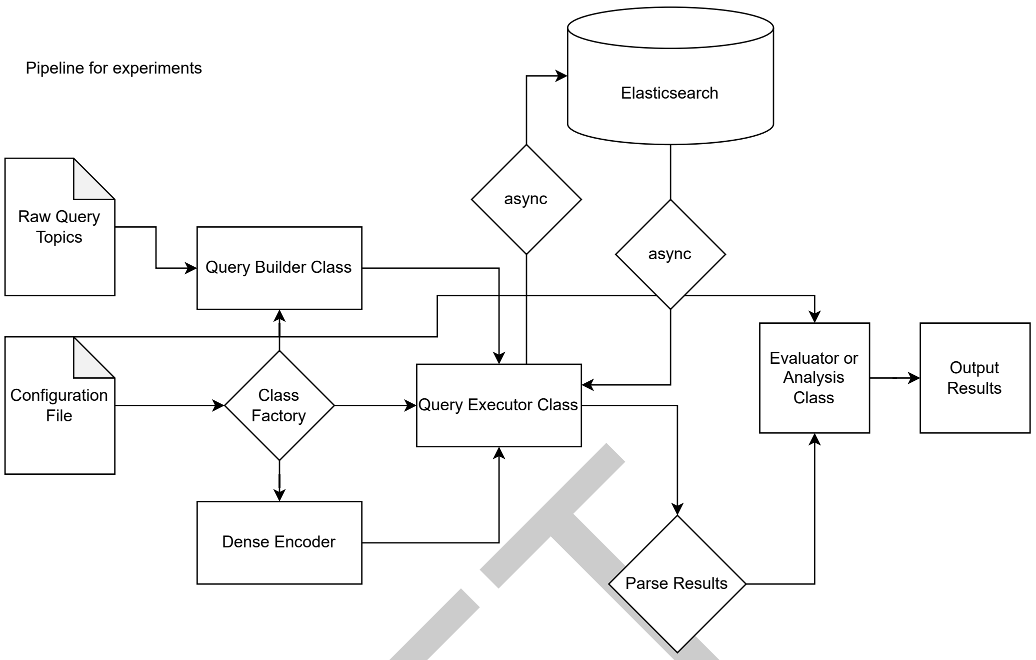

Figure 1: Standard flow of the DeBIR query/evaluation loop.

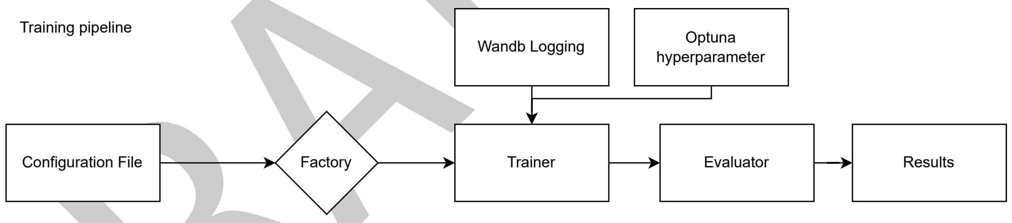

Figure 2: Standard flow of the DeBIR training loop.

Acknowledgments

The DeBIR library uses sentence-transformers, hugggingface's transformers and datasets, allRank, optuna, elasticsearch and trectools python packages.

This search is supported by CSIRO Data61, an Australian Government agency through the Precision Medicine FSP program and the Australian Research Training Program. We extend thanks to Brian Jin (Data61) for providing a code review.

Examples

Pipeline

The pipeline is a single class that can be built from a configuration file. The configuration file is a TOML file that defines the pipeline stages and their parameters. The pipeline is built using a class factory that takes in the configuration file and creates the pipeline stages. The pipeline stages are then executed in order.

```
from debeir.interfaces.pipeline import NIRPipeline
from debir.interfaces.callbacks import (SerializationCallback,
                                        EvaluationCallback)
```

```python
from debir.evaluation import Evaluator

p = NIRPipeline.build_from_config(config_fp="./tests/config.toml",
                                  engine="elasticsearch",
                                  nir_config_fp="./tests/nir_config.toml")

# Optional callbacks to serialize to disk
serial_cb = SerializationCallback(p.config, p.nir_settings)

# Or evaluation
evaluator = Evaluator.build_from_config(p.config, metrics_config=p.metrics_config)
evaluate_cb = EvaluationCallback(evaluator,
                                 config=p.config)

p.add_callback(serial_cb)
p.add_callback(evaluate_cb)

# Asynchronously execute queries
results = await p.run_pipeline()

# Post processing of results can go here
```

88 **Training a model**

```python
import wandb

from debeir.training.hparm_tuning.trainer import SentenceTransformerTrainer
from debeir.training.hparm_tuning.config import HparamConfig
from sentence_transformers import evaluation

# Load a hyper-parameter configuration file
hparam_config = HparamConfig.from_json(
        "./configs/training/submission.json"
)

# Integration with wandb
wandb.wandb.init(project="My Project")

# Create a trainer object
trainer = SentenceTransformerTrainer(
    dataset=get_dataset(), # Specify some dataloading function here
    evaluator_fn=evaluation.BinaryClassificationEvaluator,
    hparams_config=hparam_config,
    use_wandb=True
)

# Foward parameters to underlying SentenceTransformer model
trainer.fit(
    save_best_model=True,
    checkpoint_save_steps=179
)
```

Hyperparameter tuning

```python
from sentence_transformers import evaluation
from debeir.training.hparm_tuning.optuna_rank import (run_optuna_with_wandb,
                                                       print_optuna_stats)
from debeir.training.hparm_tuning.trainer import SentenceTransformerHparamTrainer
from debeir.training.hparm_tuning.config import HparamConfig

# Load a hyper-parameter configuration file with optuna parameters
hparam_config = HparamConfig.from_json(
   "./configs/hparam/trec2021_tuning.json"
)

trainer = SentenceTransformerHparamTrainer(
   dataset_loading_fn=data_loading_fn,
   evaluator_fn=evaluation.BinaryClassificationEvaluator,
   hparams_config=hparam_config,
)

# Run optuna with wandb integration
study = run_optuna_with_wandb(trainer, wandb_kwargs={
    "project": "my-hparam-tuning-project"
})

# Print optuna stats and best run
print_optuna_stats(study)
```

More information on the library is found on the github page, DeBeIR . Any feedback and suggestions are welcome at issues .

References

Armstrong, T., Moffat, A., Webber, W., & Zobel, J. (2009). Improvements that don't add up: Ad-hoc retrieval results since 1998. *CIKM*, 601–610.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 4171–4186. https://doi.org/10.18653/v1/N19-1423

Guo, J., Fan, Y., Ai, Q., & Croft, B. (2017). A deep relevance matching model for ad-hoc retrieval. *Computing Research Repository*, *abs/1711.08611*, 55–64. http://arxiv.org/abs/1711.08611

Hui, K., Yates, A., Berberich, K., & Melo, G. de. (2017). A position-aware deep model for relevance matching in information retrieval. *Computing Research Repository*, *abs/1704.03940*. http://arxiv.org/abs/1704.03940

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 25* (pp. 1097–1105). Curran Associates, Inc.

Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., & Kang, J. (2019). BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, *36*(4), 1234–1240. https://doi.org/10.1093/bioinformatics/btz682

Lin, J. (2019). Neural hype, justified! A recantation. *ACM SIGIR Forum*, *53*. http://sigir.org/wp-content/uploads/2019/december/p088.pdf

Liu, Yang, & Lapata, M. (2019). Text summarization with pretrained encoders. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 3730–3740. https://doi.org/10.18653/v1/D19-1387

Liu, Yinhan, Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A robustly optimized BERT pretraining approach. *Computing Research Repository*, *abs/1907.11692*. http://arxiv.org/abs/1907.11692

Nguyen, V., Rybinski, M., Karimi, S., & Xing, Z. (2022). Search like an expert: Reducing expertise disparity using a hybrid neural index for COVID-19 queries. *Journal of Biomedical Informatics*, *127*, 104005. https://doi.org/https://doi.org/10.1016/j.jbi.2022.104005

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. *EMNLP*, 3982–3992. https://doi.org/10.18653/v1/D19-1410

Robertson, S., Walker, S., Jones, S., Hancock-Beaulieu, M., & Gatford, M. (1995, January). Okapi at TREC-3. *TREC*. https://trec.nist.gov/pubs/trec3/t3/_proceedings.html

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014). *Going deeper with convolutions* (pp. 1–9). https://doi.org/10.1109/CVPR.2015.7298594

Yang, W., Lu, K., Yang, P., & Lin, J. (2019). Critically examining the” neural hype” weak baselines and the additivity of effectiveness gains from neural ranking models. *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1129–1132. https://dl.acm.org/doi/10.1145/3331184.3331340

Yin, W., Schütze, H., Xiang, B., & Zhou, B. (2015). ABCNN: Attention-based convolutional neural network for modeling sentence pairs. *Computing Research Repository*, *abs/1512.05193*. http://arxiv.org/abs/1512.05193