

Trove: A Flexible Toolkit for Dense Retrieval

Reza Esfandiarpour Max Zuo Stephen H. Bach

Department of Computer Science, Brown University
{reza_esfandiarpour, zuo, stephen_bach}@brown.edu

Demo: <https://youtu.be/rThGH0w3wS8>



[ir-trove.dev](https://github.com/ir-trove)

Abstract

We introduce Trove, an easy-to-use open-source retrieval toolkit that simplifies research experiments without sacrificing flexibility or speed. For the first time, we introduce *efficient* data management features that load and process (filter, select, transform, and combine) retrieval datasets on the fly, with just a few lines of code. This gives users the flexibility to easily experiment with different dataset configurations without the need to compute and store multiple copies of large datasets. Trove is highly customizable: in addition to many built-in options, it allows users to freely modify existing components or replace them entirely with user-defined objects. It also provides a low-code and unified pipeline for evaluation and hard negative mining, which supports multi-node execution without any code changes. Trove’s data management features reduce memory consumption by a factor of 2.6. Moreover, Trove’s easy-to-use inference pipeline incurs no overhead, and inference times decrease linearly with the number of available nodes. Most importantly, we demonstrate how Trove simplifies retrieval experiments and allows for arbitrary customizations, thus facilitating exploratory research.

1 Introduction

Influential toolkits such as those provided by Hugging Face (HF) simplify Machine Learning (ML) pipelines and support extensive customization with minimal effort, thus facilitating exploratory research (Gugger et al., 2022; Lhoest et al., 2021; Wolf et al., 2020). Similarly, existing retrieval toolkits have significantly improved Information Retrieval (IR) pipelines (Gao et al., 2022; Reimers and Gurevych, 2019). However, IR experiments still require a considerable amount of engineering effort for many tasks like efficient data management or model customization. Here, we introduce a novel open-source toolkit that simplifies various stages of retrieval pipelines, enabling ef-

ficient data management, flexible modeling, and easy distributed evaluation. Our design prioritizes customization and makes it easy to freely modify or entirely replace each component.

General-purpose toolkits are not directly applicable in retrieval pipelines. Retrieval is uniquely different from most ML problems in that instances of retrieval tasks are not self-contained. For example, while solving an image classification task only involves one image, a single retrieval task involves one query and the *entire corpus*. This makes retrieval experiments more challenging. Since most data management tools like HF Datasets (Lhoest et al., 2021) process each instance isolated from the rest of the dataset, they cannot be directly used in retrieval pipelines (Gao et al., 2022). Distributed evaluation is also more challenging. Instances of retrieval tasks share a lot of the computation (i.e., encoding the corpus), and we cannot simply evaluate each instance on a separate device and aggregate the results. Finally, HF transformers models only provide the encoder, and we cannot directly use them for retrieval without additional modeling.

Existing toolkits have recognized these issues and offer initial solutions. However, these solutions are often not as flexible or easy to use. Since naive on-the-fly data preparation for retrieval is memory-intensive, current toolkits rely on large pre-processed dataset files (Gao et al., 2022), often duplicating a lot of data for variations of a single dataset (Fig. 1 top). Although evaluating retrieval tasks is computationally more demanding, currently distributed evaluation is either limited to a single node (Muennighoff et al., 2022; Reimers and Gurevych, 2019) or involves several steps and more engineering effort (Gao et al., 2022). For modeling, current frameworks wrap transformers models in fixed classes, and customizations are limited to a set of pre-defined options. As a result, exploratory experiments require significant engineering effort, which slows down novel research.

In this work, we introduce Trove, an open-source library that simplifies dense retrieval experiments without sacrificing flexibility. Trove is the first toolkit to provide features for efficiently managing and pre-processing retrieval data on the fly (Fig. 1 bottom). Trove provides a simple interface for multi-node/GPU inference and is fully compatible with HF transformers ecosystem. Our modeling approach provides direct access to model components and allows arbitrary customizations. In general, our design increases flexibility at three levels. 1) Trove provides various built-in options to customize experiments. 2) Our transparent design allows users to override many methods with custom logic. 3) Trove’s modular structure allows users to entirely replace many components with arbitrary objects. In summary:

- Trove is built around the unique characteristics of the retrieval task and, for the first time, offers fast and memory-efficient operations for loading and pre-processing (filter, transform, combine, etc.) retrieval data *on the fly*.
- Trove provides a simple and unified interface for evaluation and hard negative mining, which supports both multi-node and multi-GPU inference without additional code.
- Trove allows for direct customization of all modeling components or even replacing them with arbitrary modules, while maintaining compatibility with HF transformers ecosystem.
- Trove is designed with customization in mind. The codebase is heavily documented and easy to understand. We provide ample guides and examples to facilitate customization.

2 Background and Challenges

There is a growing body of work on transformer-based dense retrievers. Many works improve the training data through techniques like using mined hard negatives or a large number of random in-batch negatives (Karpukhin et al., 2020; Moreira et al., 2024; Qu et al., 2021; Rekabsaz et al., 2021; Xiong et al., 2020; Zerveas et al., 2022, 2023; Zhan et al., 2021). Several works also use synthetic data for training (Alaofi et al., 2023; Bonifacio et al., 2022; Dai et al., 2022; Jeronymo et al., 2023; Lee et al., 2024; Li et al., 2024). With the introduction of RepLLaMA (Ma et al., 2024), large

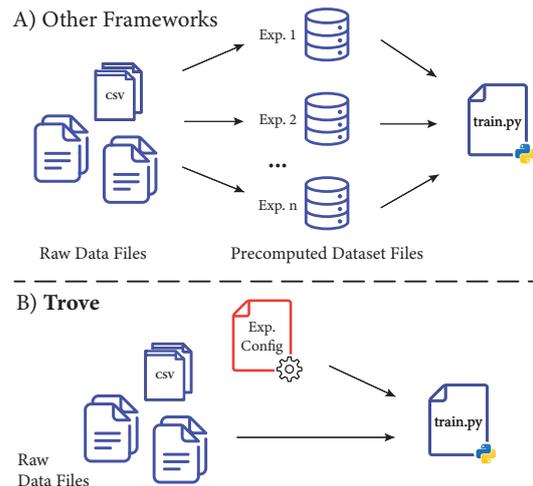


Figure 1: A) Existing toolkits require manually creating and maintaining large pre-processed data files for each experiment. B) Trove processes datasets on the fly based on the given configuration options.

decoder-only LLMs and PEFT techniques (Hu et al., 2022) have become popular for retrieval models (Wang et al., 2024). There are also new variants of the retrieval task itself, such as retrieval instructions (Asai et al., 2022; Weller et al., 2024).

Despite the dynamic research landscape, developing IR pipelines is still challenging. To illustrate the challenges of developing IR pipelines, we compare the development experience for IR to other ML tasks for three common operations.

Data Management For most ML problems, tools like HF Datasets load, pre-process, and combine multiple datasets on the fly with just a few lines of code and minimal memory overhead. In IR, even generating training samples for standard datasets like MS MARCO (Bajaj et al., 2018) requires a lot of memory and extra code. Existing tools do not offer any data management functionality and instead rely on large pre-processed data files, which require maintaining many large files with duplicate data. In addition to the cumbersome process, data changes are not trackable by VCS with this approach, hurting reproducibility.

Modeling Usually, users have full control over the model and can apply *arbitrary* customizations (e.g., add LoRA adapters or change the loss). However, current retrieval toolkits wrap the encoder in custom classes, without providing direct access to the transformers backbone. As a result, any customization requires explicit support from the library¹. The interactions between different com-

¹Example: [sentence-transformers/issues/2575](https://github.com/huggingface/sentence-transformers/issues/2575)

ponents also limit flexibility. For example, it is not possible to train with graduated relevance labels using Tevatron (Gao et al., 2022) even if we overwrite the loss function.

Inference The evaluation pipeline often involves a simple script that executes the forward pass and calculates the metrics, all in one job step. For distributed evaluation, users just need to execute the same script with a distributed launcher like Accelerate (Gugger et al., 2022), with minimal changes. Although IR evaluation is computationally more demanding, multi-node execution is not straightforward. The evaluation process with SentenceTransformers and MTEB² packages is easy but limited to only a single node. Tevatron supports multi-node evaluation, but it needs to manually launch multiple jobs to encode each dataset shard separately, and then launch another job to retrieve related documents and another job to calculate the metrics. See Section B for discussion of other libraries.

3 System Design

Here, we first explain Trove experiment workflows (Fig. 2) and then describe its major components.

3.1 Workflow

Trove experiment workflows are simple and based on configuration objects. Users create one or more instances of `MaterializedQRelConfig` to specify how raw input files (query, corpus, qrels) should be loaded and processed (e.g., filtered). We also create a `DataArguments` object with dataset-level details (e.g., sequence length). Users then use these objects to instantiate one of the main dataset classes (`BinaryDataset` or `MultiLevelDataset`). Next, we create a `ModelArguments` instance with model details like name, pooling type, and LoRA configuration. We instantiate the main retriever (e.g., `BiEncoderRetriever`) from the argument object. Finally, we use these components in addition to an instance of `RetrievalTrainingArguments` (`EvaluationArguments`) to instantiate `RetrievalTrainer` (`RetrievalEvaluator`), which is responsible for the main training (evaluation) loop. Our design allows instantiating configuration objects from command-line arguments, which makes it easier to run diverse experiments. Moreover, our design increases flexibility by exposing the main components of the pipeline: users can easily customize or entirely replace each

²gh/embeddings-benchmark/mteb

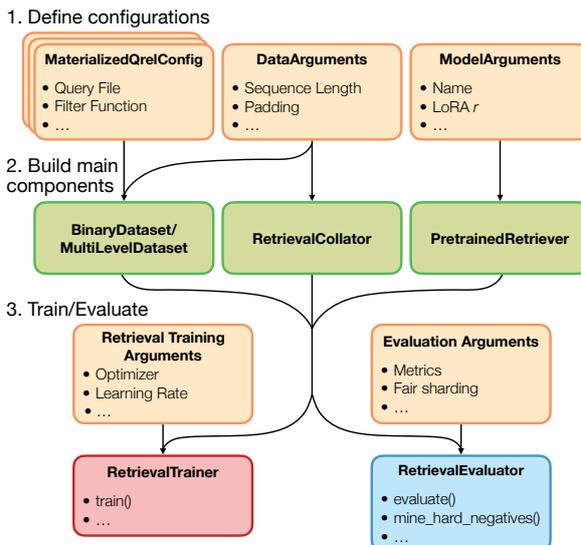


Figure 2: Training and evaluation workflow with Trove.

module without changing Trove’s source code or even major changes to their experiment workflow.

3.2 Data Management

Large IR datasets are often made of three sets of files: query, corpus, and qrels (i.e., annotations). Creating training instances is simple: for each query, find the related document IDs from the qrel files and then replace these IDs with the content from the query and corpus files. Efficiently implementing this logic is challenging for large datasets like MS MARCO with 500K queries and 8M documents. Just loading all the query and corpus records consumes a lot of memory. Moreover, with multiple datasets or more complex pipelines, pre-processing or merging millions of qrel records slows down the program.

3.2.1 Internal Representation

We implement `MaterializedQRel`, an efficient container for IR data that holds query, corpus, and qrel records. We use the Polars library³ to efficiently group qrel triplets by query ID, which significantly speeds up pre-processing and lookup operations for related documents. We convert query, corpus, and grouped qrel records to memory-mapped Apache Arrow tables that are indexable by ID. `MaterializedQRel` only works with IDs, *without loading the actual data*. For each training instance, we load the data at the very last step and even then only load the necessary records for the current instance. As a result, `MaterializedQRel` significantly reduces memory consumption.

³<https://pola.rs>

MaterializedQRel enables users to process the data just by setting a few config values. For instance, users can easily filter qrel triplets, select a subset of queries, or change the labels. See Section 4 for examples.

3.2.2 User-facing Classes

These benefits are available to users through the MultiLevelDataset and BinaryDataset classes. Trove datasets are made of one or more MaterializedQRel instances, which allows defining complex data pipelines. For instance, users can apply different pre-processing steps to each data source (e.g., real and synthetic) before combining them. To create a dataset, users just need to initialize the dataset class with config objects that specify data loading and processing details. As a result, Trove’s data pipelines are trackable with VCS.

Our EncodingDataset prepares the data for encoding during inference and simplifies embedding cache management. We implement the embedding cache as memory-mapped Apache Arrow tables, indexed by ID. The cache supports *lazy loading*, loading each cached vector only when needed. The interface is simple: users call `cache_records(ids, vectors)` to write the vectors and their IDs to cache. Later, when accessing each item (i.e., `dataset[i]`), the dataset returns the cached embedding instead of raw text if available. We also implement the RetrievalCollator, which tokenizes and batches examples for retrieval.

3.2.3 Additional Features

Callbacks for Flexibility We implement frequently changing operations as callback functions for easier customization. For example, users can load qrel triplets from custom file formats by registering a loader with `@register_loader`. Users can customize input formatting (e.g., add instructions) by passing `format_query` and `format_passage` callbacks to the dataset. Similarly, the `filter_fn` option in `MaterializedQRelConfig` allows users to filter the qrel triplets with custom functions.

Reliability We cache intermediate artifacts in the first run and track changes using a fast fingerprinting method. We also use atomic write operations to guard against corrupted files. As a result, datasets are very fast after the first run and reliably generate the same data in all runs.

By default, Trove supports datasets in the widely adopted BEIR format (Thakur et al., 2021). We also provide decorators for registering new loader

functions for custom file formats. Trove can also load file URIs directly from HuggingFace Hub.

3.3 Modeling

Trove’s modeling is divided into three main components (retriever, encoder, and loss) and allows users to customize each component independently.

Retriever Subclasses of `PretrainedRetriever` are the main model class in Trove and consist of an encoder, loss function, and the retrieval logic. `PretrainedRetriever` can use all HF transformers models as encoder and supports common pooling and normalization operations, as well as LoRA adapters and quantization. It provides the `from_model_args()` method that creates the correct encoder and loss function based on the given options. To allow arbitrary customizations, we encapsulate all details related to transformers models (e.g., quantization) in the encoder and allow users to use arbitrary `nn.Module` objects as the encoder.

Users can subclass `PretrainedRetriever` and overwrite the `forward()` method to implement custom retrieval logics. Trove already comes with `BiEncoderRetriever`, which implements the dual-encoder retrieval logic with support for cross-device in-batch negatives.

Encoder To experiment with new encoding methods (e.g., different pooling or PEFT techniques), users can implement custom encoder wrappers as `PretrainedEncoder` subclasses. Compared to using arbitrary `nn.Module` objects as encoder, this allows us to swap encoder wrappers without changing the code, which simplifies user scripts. Users just need to instantiate the retriever with different options (e.g., `encoder_class="MyEncoderClass"`).

Loss Function Trove implements the InfoNCE and KL Divergence losses. Users can implement new loss functions as `RetrievalLoss` subclasses and choose the correct loss through retriever options (e.g., `loss="MyLossClass"` or `"kl"`).

3.4 Training

Inspired by Tevatron (Gao et al., 2022), we ensure all Trove components are compatible with HF transformers and directly use its Trainer module for training, with minimal changes.

Trove makes it possible to approximate IR metrics like nDCG during training by ranking a small number of annotated documents for each development query, similar to a reranking task. We

```

1 from transformers import AutoTokenizer, HfArgumentParser
2 from trove import *
3
4 parser = HfArgumentParser((RetrievalTrainingArguments, ModelArguments, DataArguments))
5 train_args, model_args, data_args = parser.parse_args_into_dataclasses()
6
7 tokenizer = AutoTokenizer.from_pretrained(...)
8 model = BiEncoderRetriever.from_model_args(...)
9 collator = RetrievalCollator(data_args, tokenizer, append_eos=False)
10
11 pos = MaterializedQRelConfig(min_score=1, qrel_path="qrels/train.tsv", ...)
12 neg = MaterializedQRelConfig(group_random_k=2, qrel_path="mined_neg.tsv", ...)
13 dataset = BinaryDataset(data_args, model.format_query, model.format_passage, pos, neg)
14
15 trainer = RetrievalTrainer(model, train_args, collator, dataset)
16 trainer.train()

```

Figure 3: Training with Mined Hard Negatives

introduce IRMetrics, which can be used as the `compute_metric` callback to efficiently calculate approximate IR metrics during training for small instances of `MultiLevelDataset`.

3.5 Inference

`RetrievalEvaluator` class implements a simple and unified interface for evaluation and hard negative mining. Inference is as easy as creating an instance of `RetrievalEvaluator` and calling the `evaluate()` or `mine_hard_negatives()` method. Trove supports logging to Wandb and can integrate other experiment trackers using callback functions.

For distributed inference, we simply need to launch the same script without any changes using a distributed launcher. `RetrievalEvaluator` automatically distributes the computation across available *nodes* and GPUs. We also introduce a *fair sharding* feature that allows mixing GPUs with different capabilities without stalling the faster devices. Trove adjusts the shard sizes based on GPU throughput, assigning more samples to faster devices.

Trove introduces `FastResultHeapq`, a Pytorch alternative to the standard Python `heapq` that uses fast matrix operations and GPU acceleration for tracking the top-k documents for each query⁴. Existing frameworks often use Python’s `heapq`, which is a major bottleneck and stalls GPU cycles during evaluation (Muennighoff et al., 2022). `FastResultHeapq` is 16x and 600x faster than Python `heapq` for online and cached embeddings, respectively.

4 Demonstration

Here, we demonstrate Trove’s flexibility and ease of use and benchmark its efficiency.

⁴This is not a full `heapq`. It just mimics some functionalities to keep track of topk documents for each query.

4.1 Flexibility and Ease of Use

We have already used Trove for large-scale research experiments in our earlier work, SyCL (Esfandiarpour et al., 2025). Below, we outline the pipeline for two key SyCL experiments. Trove can be easily installed from PyPI:

```
$ pip install ir-trove
```

Trove greatly reduces the engineering effort required for common training setups. Figure 3 shows the complete code needed to train dense retrievers with mined hard negatives using InfoNCE loss. This simple code already supports multi-node/GPU training, standard pooling and normalization operations, LoRA adapters, and quantization.

Now, we modify the code to train on a mix of multi-level synthetic data (labels in {0, 1, 2, 3}), annotated positives, and mined hard negatives. To do this, we simply replace lines 11–13 in Fig. 3 with the following:

```

syn = MaterializedQRelConfig(...,
  qrel_path="synth_qrel.tsv",
  corpus_path="synth_corpus.jsonl",
  query_subset_path="qrels/orig_train.tsv")
pos = MaterializedQRelConfig(...,
  qrel_path="qrels/train.tsv",
  score_transform=3,
  min_score=1)
neg = MaterializedQRelConfig(...,
  qrel_path="mined_neg.tsv",
  score_transform=1,
  group_random_k=2)
dataset = MultiLevelDataset([syn, pos, neg], ...)

```

This snippet processes each data source differently and combines the results. `syn` collection selects only synthetic data for training queries, using query IDs from `qrels/train.tsv` file. `pos` collection filters for documents with relevance labels ≥ 1 (i.e., positives), then assigns them a new label of 3. And,

neg randomly selects two of the hard negatives per query and assigns them a new label of 1.

In SyCL, we also explore the Wasserstein distance as loss function. For this, we just implement the loss function as the following and use `--loss=ws` to run the training script.

```
class WSLoss(RetrievalLoss):
    _alias = "ws"

    def forward(self, logits, label):
        loss = ... # calculate the loss value
        return loss
```

4.1.1 Model Customization

Trove provides different methods for customizing the model. As described in Section 3.3, Trove comes with many built-in options for choosing different pooling operations, applying embedding normalization, adding LoRA adapters, or quantization. Here, we provide several examples of how users can further customize models beyond these options.

Input Formatting For convenience, Trove encoders include two methods for proper input formatting for a given model. The code below, modifies these methods to add instructions to input queries and passages, similar to Wang et al. (2023).

```
class EncoderWithInstructions(DefaultEncoder):
    _alias = "encoder_with_inst"

    def format_query(self, text, dataset,
        ↪ **kwargs):
        if dataset == "msmarco":
            inst = "Instruct: Given a web search
            ↪ query, retrieve relevant passages
            ↪ that answer the query\nQuery: "
        else:
            inst = "Query: "
        return inst + text

    def format_passage(self, text, title=None,
        ↪ **kwargs):
        return f"Passage: {title} {text}"
```

Then, in the user scripts (e.g., Fig. 3), they just need to pass `--encoder_class="encoder_with_inst"` to use this modified encoder wrapper.

Pooling Method Here is an example of how users can modify the default encoder wrapper to implement different pooling operations.

```
class EncoderWithNewPooling(DefaultEncoder):
    _alias = "encoder_new_pooling"

    def encode(self, inputs) -> torch.Tensor:
        output = self.model(**inputs,
            ↪ return_dict=True)
        embs = ... # custom pooling and
            ↪ normalization operations
        return embs
```

Similar to above, users can use `--encoder_class="encoder_new_pooling"` to use the above encoder wrapper.

New Encoder Wrappers Users can also directly subclass `PretrainedEncoder`, which gives them control over how the model is loaded, saved, and used for calculating the embeddings.

```
class CustomEncoder(PretrainedEncoder):
    _alias = 'custom_encoder'

    def __init__(self, args:
        ↪ trove.ModelArguments, **kwargs):
        self.model = AutoModel.from_pretrained(
            args.model_name_or_path)
        # Arbitrary Model Customizations (LoRA,
        ↪ Quantization, etc.):
        # ...

    def save_pretrained(self, *args, **kwargs):
        ...

    def encode_query(self, inputs):
        ...

    def encode_passage(self, inputs):
        ...
```

These new encoder wrappers are also automatically registered and available through configuration options (e.g., `--encoder_class="custom_encoder"`).

User-provided Encoder Objects Users can also directly instantiate `BiEncoderRetriever` with any `nn.Module` object as the encoder.

```
custom_encoder: torch.nn.Module = ... # any
↪ encoder module
model =
↪ BiEncoderRetriever(encoder=custom_encoder,
↪ model_args)
```

4.2 Efficiency

Here, we benchmark the impact of Trove's optimizations for data management and inference. Our main goal is to benchmark Trove's efficiency when handling large-scale datasets. Therefore, as a representative example of large-scale datasets, we use MS MARCO for benchmarking. Moreover, to further stress-test Trove, we also increase the size of the original MS MARCO dataset by adding synthetic documents.

Data Management Table 1 shows the memory required to prepare MS MARCO data for training. The naive baseline loads the entire data in memory. Trove cuts memory usage by 2.6x. Since Trove's main data is on disk and memory-mapped, increasing the data size only marginally increases

	Real		Real w/ Synth.	
	1x GPU	8x GPU	1x GPU	8x GPU
Naive	8.85	70.80	11.30	90.40
Trove	3.34	26.72	4.07	32.56

Table 1: Memory consumption in GB

the memory usage: when combining synthetic and real data (Esfandiarpour et al., 2025), Trove adds 0.73 GB of memory to load the additional 2M synthetic passages, far less than the naive approach, which uses 2.45 GB. This efficiency is critical for distributed training, where each process loads its own data. On a machine with 8 GPUs, the naive method consumes 90 GB of RAM just for data loading. We note that most existing frameworks rely on large pre-processed data files for each individual experiment, and Trove is the only library to offer efficient on-the-fly data processing for retrieval.

	Real		Real w/ Synth.	
	TTFS	TTFS _{1st}	TTFS	TTFS _{1st}
Naive	31	31	40	40
Trove	5	39	7	55

Table 2: Time to first sample (TTFS) in seconds for the first and subsequent runs

Table 2 reports time to first sample (TTFS), measuring the time to load and process the data. Thanks to Trove’s internal caching, after the first run, the data is available almost instantaneously. While TTFS has minimal impact on long-running experiments, a short TTFS is critical for efficient debugging and interactive development.

Inference Table 3 shows retrieval times for all queries in MS MARCO using E5-Mistral-Instruct (Wang et al., 2024) in a distributed environment. Inference time decreases linearly with the number of nodes, showing that Trove uses additional nodes with no overhead. Crucially, we just need to run the same script with a distributed launcher, without changing the code.

	1x Node	2x Node	3x Node
Inference Time	14h:20m	7h:12m	4h:48m

Table 3: Inference time on different numbers of nodes. Note that other libraries are limited to 1x Node.

Table 4 compares the performance of Python’s heapq with FastResultHeapq for keeping track of top-k documents at MS MARCO scale. In an on-line setup where we embed a small batch of 256 documents and compare it with queries on the fly, Trove is more than 600x faster than Python’s heapq. When the number of queries grows (e.g., for hard negative mining), Python’s heapq becomes unusable, taking up to 130 hours.

Queries	On The Fly		w/ Cached Embs	
	Standard	Trove	Standard	Trove
6K	1h:9m	7s	21s	1s
500K	130h:40m	11m:45s	30m:17s	1m:52s

Table 4: Performance of Python’s heapq vs Trove’s FastResultHeapq during inference

Even when embeddings are cached and comparisons are made in large batches (e.g., 40,960 documents) on GPU, Trove remains 16x to 21x faster. However, in practice, this speedup is not realized for Python’s heapq. It is often bottlenecked by disk I/O, particularly with the simple caching mechanisms used by existing frameworks.

Training Results For training results and additional experiments using Trove, we refer readers to the SyCL paper (Esfandiarpour et al., 2025) and codebase⁵, which uses Trove for all its experiments.

5 Conclusion

In this work, we introduce Trove, an open-source toolkit for dense retrieval that reduces the engineering effort in research experiments. Trove eliminates the need for large pre-processed data files and, for the first time, provides data management features that load and process retrieval data on the fly, with a small memory footprint. Trove provides full control over modeling and allows users to freely customize different modeling components. Trove provides a simple and unified interface for evaluation and hard negative mining, which supports multi-node inference without any extra code. While Trove provides a simple high-level interface, every component is designed to be configured, modified, or replaced entirely. As a result, Trove provides researchers with a tool to quickly and freely experiment with new ideas.

⁵[gh/BatsResearch/sycl](https://github.com/BatsResearch/sycl)

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. RISE-2425380. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This research is supported in part by the Office of Naval Research (ONR) award N00014-20-1-2115. Disclosure: Stephen Bach is an advisor to Snorkel AI, a company that provides software and services for data-centric artificial intelligence.

References

- Marwah Alaofi, Luke Gallagher, Mark Sanderson, Falk Scholer, and Paul Thomas. 2023. Can generative llms create query variants for test collections? an exploratory study. In *Proceedings of the 46th international ACM SIGIR conference on research and development in information retrieval*, pages 1869–1873.
- Akari Asai, Timo Schick, Patrick Lewis, Xilun Chen, Gautier Izacard, Sebastian Riedel, Hannaneh Hajishirzi, and Wen-tau Yih. 2022. Task-aware retrieval with instructions. *arXiv preprint arXiv:2211.09260*.
- Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. 2018. **MS MARCO: A Human Generated MACHine Reading COMprehension Dataset**. *arXiv:1611.09268 [cs]*. ArXiv: 1611.09268.
- Luiz Bonifacio, Hugo Abonizio, Marzieh Fadaee, and Rodrigo Nogueira. 2022. Inpars: Data augmentation for information retrieval using large language models. *arXiv preprint arXiv:2202.05144*.
- Zhuyun Dai, Vincent Y Zhao, Ji Ma, Yi Luan, Jianmo Ni, Jing Lu, Anton Bakalov, Kelvin Guu, Keith B Hall, and Ming-Wei Chang. 2022. Promptagator: Few-shot dense retrieval from 8 examples. *arXiv preprint arXiv:2209.11755*.
- Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. **The faiss library**.
- Reza Esfandiarpour, George Zerveas, Ruochen Zhang, Macton Mgonzo, Carsten Eickhoff, and Stephen H Bach. 2025. Beyond contrastive learning: Synthetic data enables list-wise training with multiple levels of relevance. *arXiv preprint arXiv:2503.23239*.
- Luyu Gao, Xueguang Ma, Jimmy J. Lin, and Jamie Callan. 2022. Tevatron: An efficient and flexible toolkit for dense retrieval. *ArXiv*, abs/2203.05765.
- Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, Marc Sun, and Benjamin Bossan. 2022. Accelerate: Training and inference at scale made simple, efficient and adaptable. <https://github.com/huggingface/accelerate>.
- Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. **LoRA: Low-rank adaptation of large language models**. In *International Conference on Learning Representations*.
- Vitor Jeronimo, Luiz Bonifacio, Hugo Abonizio, Marzieh Fadaee, Roberto Lotufo, Jakub Zavrel, and Rodrigo Nogueira. 2023. Inpars-v2: Large language models as efficient dataset generators for information retrieval. *arXiv preprint arXiv:2301.01820*.
- 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)*, pages 6769–6781, Online. Association for Computational Linguistics.
- Jinhyuk Lee, Zhuyun Dai, Xiaoqi Ren, Blair Chen, Daniel Cer, Jeremy R. Cole, Kai Hui, Michael Boratko, Rajvi Kapadia, Wen Ding, Yi Luan, Sai Meher Karthik Duddu, Gustavo Hernandez Abrego, Weiqiang Shi, Nithi Gupta, Aditya Kusupati, Praatek Jain, Siddhartha Reddy Jonnalagadda, Ming-Wei Chang, and Iftekhhar Naim. 2024. **Gecko: Versatile text embeddings distilled from large language models**. *Preprint*, arXiv:2403.20327.
- Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, and 13 others. 2021. **Datasets: A community library for natural language processing**. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.
- Xiaopeng Li, Xiangyang Li, Hao Zhang, Zhaocheng Du, Pengyue Jia, Yichao Wang, Xiangyu Zhao, Huifeng Guo, and Ruiming Tang. 2024. Syneg: Llm-driven synthetic hard-negatives for dense retrieval. *arXiv preprint arXiv:2412.17250*.
- Xueguang Ma, Liang Wang, Nan Yang, Furu Wei, and Jimmy Lin. 2024. Fine-tuning llama for multi-stage text retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2421–2425.
- Craig Macdonald and Nicola Tonellotto. 2020. Declarative experimentation in information retrieval using pyterrier. In *Proceedings of ICTIR 2020*.

- Gabriel de Souza P Moreira, Radek Osmulski, Mengyao Xu, Ronay Ak, Benedikt Schifferer, and Even Oldridge. 2024. *Nv-retriever: Improving text embedding models with effective hard-negative mining*. *arXiv preprint arXiv:2407.15831*.
- Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2022. *Mteb: Massive text embedding benchmark*. *arXiv preprint arXiv:2210.07316*.
- 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*, pages 5835–5847, Online. Association for Computational Linguistics.
- Nils Reimers and Iryna Gurevych. 2019. *Sentence-bert: Sentence embeddings using siamese bert-networks*. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.
- Navid Reikabsaz, Oleg Lesota, Markus Schedl, Jon Brassey, and Carsten Eickhoff. 2021. *TripClick: The Log Files of a Large Health Web Search Engine*. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2507–2513. Association for Computing Machinery, New York, NY, USA.
- Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. *BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models*. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.
- Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2023. *Improving text embeddings with large language models*. *arXiv preprint arXiv:2401.00368*.
- Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024. *Improving text embeddings with large language models*. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11897–11916, Bangkok, Thailand. Association for Computational Linguistics.
- Orion Weller, Benjamin Van Durme, Dawn Lawrie, Ashwin Paranjape, Yuhao Zhang, and Jack Hessel. 2024. *Promptretriever: Instruction-trained retrievers can be prompted like language models*. *Preprint*, arXiv:2409.11136.
- Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. *Transformers: State-of-the-art natural language processing*. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.
- Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. 2020. *Approximate nearest neighbor negative contrastive learning for dense text retrieval*. *arXiv preprint arXiv:2007.00808*.
- George Zerveas, Navid Reikabsaz, Daniel Cohen, and Carsten Eickhoff. 2022. *CODER: An efficient framework for improving retrieval through CONTEXTUAL Document Embedding Reranking*. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 10626–10644, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.
- George Zerveas, Navid Reikabsaz, and Carsten Eickhoff. 2023. *Enhancing the ranking context of dense retrieval through reciprocal nearest neighbors*. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 10779–10803, Singapore. Association for Computational Linguistics.
- Jingtao Zhan, Jiabin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. 2021. *Optimizing dense retrieval model training with hard negatives*. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*, pages 1503–1512.

A Limitations

Trove is mainly aimed at researchers, and our design emphasizes a simple codebase that is freely customizable. Since embedding models are a major component of IR pipelines and are even frequently used in RAG applications without additional steps like re-ranking, Trove provides extensive support for training and evaluating dense embedding models out of the box. However, while we envision adding support for other major retrieval methods such as sparse or hybrid retrieval, Trove is not designed to support all variants of different retrieval methods out of the box. Instead, Trove’s design makes it easy to implement custom variations of different methods in user scripts. On the other end of the spectrum, there is SentenceTransformers, a great library that makes it easy to get started with IR experiments, with built-in support for many retrieval methods. Inevitably, such a codebase is complex and hard to modify by users, which is what we are trying to avoid with Trove.

Moreover, to facilitate exploration and rapid experiments, we sometimes avoid industry standards and implement our own solutions. For example, we implement a fast embedding cache and our own Pytorch container (i.e., `FastResultHeapq`) to speed up nearest neighbor search instead of using tools such as FAISS (Douze et al., 2024), which are also used by other libraries like Tevatron and PyTerrier. While FAISS has many great features, in our case, creating a large search index that is only used once is not as efficient. Moreover, such dependencies limit flexibility. For example, FAISS only reports the similarity scores of the most similar documents for each query. On the other hand, our method can also track similarity scores for arbitrary documents even if they are not ranked among top-k results but, for example, are useful for answering a specific research question.

B Other Libraries

Here, we discuss Trove’s main differences with two other popular libraries in the information retrieval space.

Pyterrier While Pyterrier (Macdonald and Tonello, 2020) is a mature library for composing general IR pipelines, its support for dense retrieval (via plugins⁶) is limited. On the other hand, Trove mainly focuses on reducing the computational overhead and complexity of IR experiments with dense embedding models. Unlike Pyterrier, Trove supports multi-node evaluation of dense retrieval models out of the box, without any additional effort. Furthermore, Pyterrier mainly relies on Pandas Dataframes to represent the data which consumes significant memory in distributed settings where each process loads a separate copy of the data. By contrast, Trove offers dedicated dataset classes optimized for training, that implement an efficient memory-mapped representation of data by leveraging technologies like Apache Arrow and Polars. Finally, Trove is designed specifically for dense retrieval and offers abstractions that simplify experimenting with different modeling choices like different loss functions and encoder architectures.

ir_datasets Another popular library is `ir_datasets`⁷, which is dedicated to providing a unified interface for loading IR datasets. By contrast, Trove encompasses the full pipeline for training and evaluating dense embedding models,

including data processing, hard negative mining, modeling, training, and evaluation. However, Trove and libraries like `ir_datasets` are not mutually exclusive. Trove can integrate different data sources and provides simple decorators that allow users to register and use custom data loader functions. For example, users can integrate new data loader functions that use `ir_datasets` for reading the data files.

⁶https://github.com/terrierteam/pyterrier_dr

⁷https://github.com/allenai/ir_datasets