Ferdinand Schlatt Maik Fröbe Matthias Hagen

🔍 Lightning IR: Straightforward Fine-tuning and Inference of Transformer-based Language Models for Information Retrieval

Lightning IR Design Goals

- Unified interface for different model types
- Easily accessible datasets
- Support all stages of a retrieval pipeline
  Fine-Tuning, Indexing & Retrieval, Re-Ranking
- Reproducible experiments and evaluation

Extending PyTorch Lightning for IR

Four main components:

- Model: Wrappers around HF 🤗 models
- Dataset: Direct integration with `ir-datasets`
- Trainer: PyTorch Lightning Trainer with IR-specific features
- CLI: All components are configurable via YAML files

Models

Bi-/Cross-encoder models can be initialized from any HF backbone

```python
from lightning_ir import BiEncoderModel, CrossEncoderModel
bi_encoder = BiEncoderModel.from_pretrained("{HF_MODEL}")
cross_encoder = CrossEncoderModel.from_pretrained("{HF_MODEL}")
```

The behavior of the models can be controlled via the configuration

For example, to configure a ColBERT model simply deactivate pooling of query / document tokens

```python
from lightning_ir import BiEncoderConfig, BiEncoderModel
config = BiEncoderConfig(
  query_pooling_strategy=None, doc_pooling_strategy=None, embedding_dim=128
)
colbert = BiEncoderModel.from_pretrained("bert-base-uncased", config=config)
```

For ease-of-use, a *Module* combines a *Model* and a *Tokenizer*

```python
from lightning_ir import BiEncoderModule, CrossEncoderModule
bi_encoder = BiEncoderModule("webis/bert-bi-encoder")
cross_encoder = CrossEncoderModule("webis/monoelectra-base")
query = "What is the capital of Germany?"
docs = ["Berlin is the capital of Germany.", "Paris is the capital of France."]
print(bi_encoder.score(query, docs).scores.numpy().round(2)) # [39.37 31.4]
print(cross_encoder.score(query, docs).scores.numpy().round(2)) # [ 7.81 -4.13]
```

Datasets

Lightning IR provides four different dataset types for different tasks

- Tuple Fine-Tuning
- Document Indexing
- Query Retrieval
- Run Re-Ranking

```python
from lightning_ir import DocDataset, QueryDataset, RunDataset, TupleDataset
print(next(iter(TupleDataset("msmarco-passage/train/triples-small"))))
# RankSample(query_id='400296', query='...', doc_ids=('1540783', '3518497'),
#            docs=('...', '...'), targets=tensor([1., 0.]),)
print(next(iter(DocDataset("msmarco-passage/train"))))
# QuerySample(query_id='121352', query='define extreme')
print(next(iter(QueryDataset("msmarco-passage/train"))))
# DocSample(doc_id='0', doc='The presence of communication ...')
print(RunDataset("msmarco-passage/trec-dl-2019", depth=3)[0])
# RankSample(query_id='1037798', query='who is robert gray',
#            doc_ids=('7134595', '7134596', '8402859'), docs=(..., ..., ...))
```

DataModules combine multiple *Datasets* for fine-tuning and inference

```python
from lightning_ir import LightningIRDataModule, RunDataset, TupleDataset
data_module = LightningIRDataModule(
  train_dataset=TupleDataset("msmarco-passage/train/triples-small"),
  inference_datasets=[
    RunDataset("msmarco-passage/trec-dl-2019"),
    RunDataset("msmarco-passage/trec-dl-2020"),
  ]
)
```

Fine-Tuning and Inference Using the CLI

- Fine-Tuning is supported out-of-the-box by PyTorch Lightning
- IR-specific inference steps use custom commands and callbacks

Fine-Tuning

```
# > train.yml
# trainer: ... # hyperparameters
# model:
#   class_path: BiEncoderModule
#   init_args: ... # hyperparameters
# data:
#   class_path: LightningIRDataModule
#   init_args: ... # hyperparameters
# optimizer:
#   class_path: torch.optim.AdamW
#   init_args: ... # hyperparameters
lightning-ir fit --config train.yml
```

Indexing

```
# > index.yml
# trainer:
#   callbacks:
#   - class_path: IndexCallback
#     init_args: ... # hyperparameters
# model:
#   class_path: BiEncoderModule
#   init_args: ... # hyperparameters
# data:
#   class_path: LightningIRDataModule
#   init_args: ... # hyperparameters
lightning-ir index --config index.yml
```

Retrieval

```
# > search.yml
# trainer:
#   callbacks:
#   - class_path: SearchCallback
#     init_args: ... # hyperparameters
# model:
#   class_path: BiEncoderModule
#   init_args: ... # hyperparameters
# data:
#   class_path: LightningIRDataModule
#   init_args: ... # hyperparameters
lightning-ir search --config search.yml
```

Re-Ranking

```
# > rerank.yml
# trainer:
#   callbacks:
#   - class_path: ReRankCallback
#     init_args: ... # hyperparameters
# model:
#   class_path: CrossEncoderModule
#   init_args: ... # hyperparameters
# data:
#   class_path: LightningIRDataModule
#   init_args: ... # hyperparameters
lightning-ir re_rank --config rerank.yml
```

External Model Support

Lightning IR supports several external models (more coming soon)

Bi-Encoders

- SBERT
- ColBERT
- SPLADE

Cross-Encoders

- mono(BERT / ELECTRA)
- monoT5
- RankT5

Reproducibility Study

Model	TREC DL 2019	TREC DL 2020
SBERT (Ours)	0.705	0.696
SBERT (Original)	0.705	0.726
SPLADE (Ours)	0.760†	0.720†
SPLADE (Original)	0.722	0.754
ColBERT (Ours)	0.738	0.726
ColBERT (Original)	0.722	0.723

Next Steps

- Support more models
  XTR, LITE, COIL, SpaDE, MVR, Set-Encoder, ...
- Large-scale comparison of models under equal conditions
- Adding pre-built indexes for common collections / models

Resources

https://webis.de/lightning-ir/

https://github.com/webis-de/lightning-ir

https://webis.de/publications.html#schlatt_2025a