ack to Articles

ain 400x faster Static Embedding Models with ntence Transformers

Upvote 126

shed January 15, 2025

late on GitHub

tomaarsen
Tom Aarsen

;DR

blog post introduces a method to train static embedding models that run 100x to 400x r on CPU than state-of-the-art embedding models, while retaining most of the quality. unlocks a lot of exciting use cases, including on-device and in-browser execution, edge puting, low power and embedded applications.

ipply this recipe to train two extremely efficient embedding models: sentence-sformers/static-retrieval-mrl-en-v1 for English Retrieval, and sentence-sformers/static-similarity-mrl-multilingual-v1 for Multilingual Similarity tasks. These els are **100x to 400x faster on CPU** than common counterparts like all-mpnet-base-v2 multilingual-e5-small, while reaching at least **85%** of their performance on various :hmarks.

ly, we are releasing:

- The two models (for English retrieval and for multilingual similarity) mentioned above.
- The detailed training strategy we followed, from ideation to dataset selection to

implementation and evaluation.

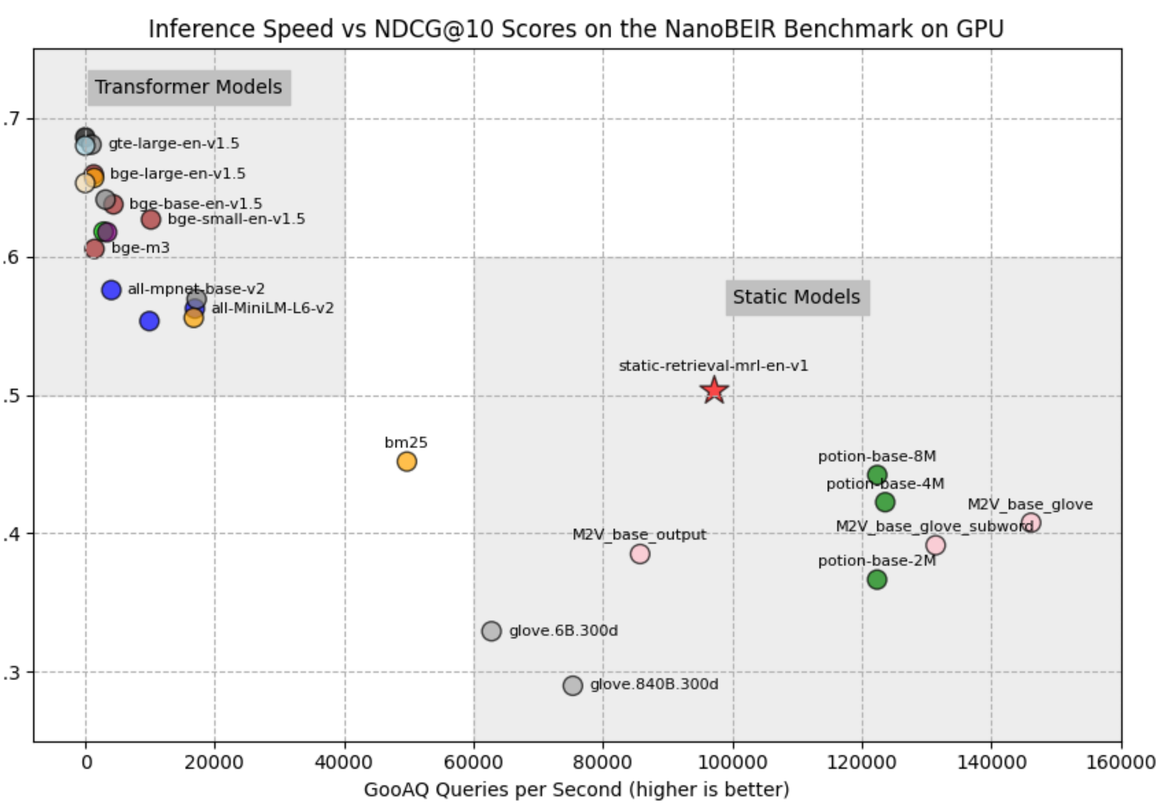
Two training scripts, based on the open-source sentence transformers library.

Two Weights and Biases reports with training and evaluation metrics collected during training.

The detailed list of datasets we used: 30 for training and 13 for evaluation.

ılso discuss potential enhancements, and encourage the community to explore them build on this work!

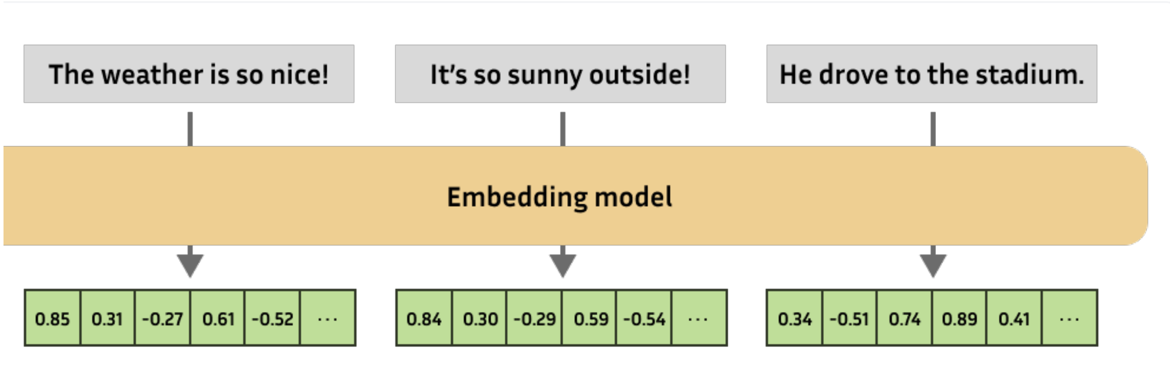
ick to see Usage Snippets for the released models



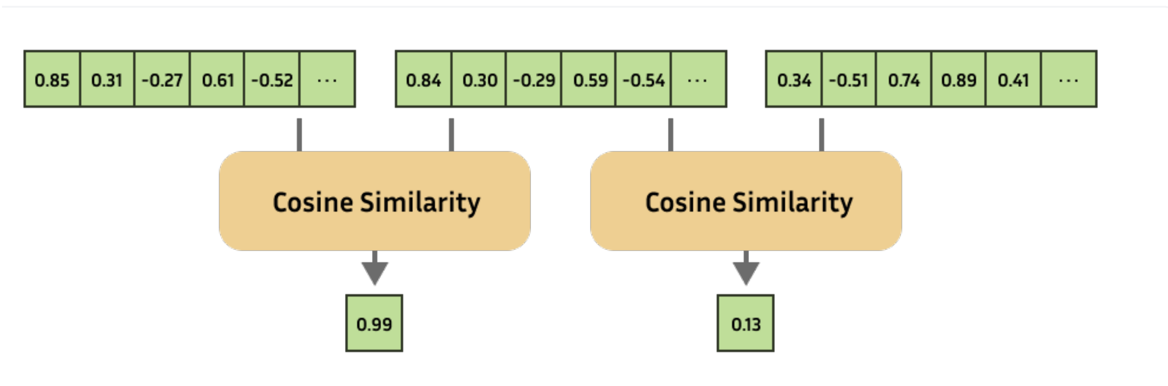
ɔle of Contents

hat are Embeddings?

eddings are one of the most versatile tools in natural language processing, enabling titioners to solve a large variety of tasks. In essence, an embedding is a numerical esentation of a more complex object, like text, images, audio, etc.



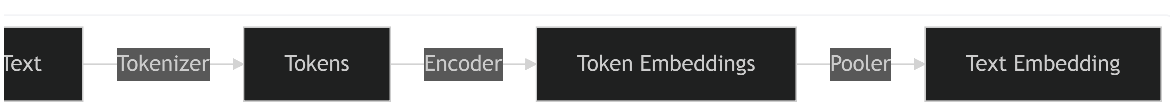
embedding model will always produce embeddings of the same fixed size. You can compute the similarity of complex objects by computing the similarity of the ective embeddings.



has a large amount of use cases, and serves as the backbone for recommendation ems, retrieval, outlier detection, one-shot or few-shot learning, similarity search, ering, paraphrase detection, classification, and much more.

dern Embeddings

y of today's embedding models consist of a handful of conversion steps. Following e steps is called "inference".



`Tokenizer` and `Pooler` are responsible for pre- and post-processing for the `Encoder`, ectively. The former chops texts up into tokens (a.k.a. words or subwords) which can nderstood by the `Encoder`, whereas the latter combines the embeddings for all tokens one embedding for the entire text.

ıin this pipeline, the `Encoder` is often a language model with attention layers, which vs each token to be computed within the *context* of the other tokens. For example, `bank` ıt be a token, but the token embedding for that token will likely be different if the text 's to a "river bank" or the financial institution.

e encoder models with a lot of attention layers will be effective at using the context to uce useful embeddings, but they do so at a high price of slow inference. Notably, in the line, the `Encoder` step is generally responsible for almost all of the computational time.

tic Embeddings

c Embeddings refers to a group of `Encoder` models that don't use large and slow ıtion-based models, but instead rely on pre-computed token embeddings. Static eddings were used years before the transformer architecture was developed. Common ıples include GLoVe and word2vec. Recently, Model2Vec has been used to convert pre-ıed embedding models into Static Embedding models.

Static Embeddings, the `Encoder` step is as simple as a dictionary lookup: given the n, return the pre-computed token embedding. Consequently, inference is suddenly no er bottlenecked by the `Encoder` phase, resulting in **speedups of several orders of nitude**. This blogpost shows that the hit on quality can be quite small!

r Method

et out to revisit Static Embeddings models, using modern techniques to train them. of our gains come from the use of a contrastive learning loss function, as we'll explain tly. Optionally, we can get additional speed improvements by using Matryoshka resentation Learning, which makes it possible to use truncated versions of the edding vectors.

l be using the Sentence Transformers library for training. For a more general overview ow this library can be used to train embedding models, consider reading the Training Finetuning Embedding Models with Sentence Transformers v3 blogpost or the ence Transformers Training Overview documentation.

ıining Details

objective with these reimagined Static Embeddings is to experiment with modern edding model finetuning techniques on these highly efficient embedding models. In icular, unlike GLoVe and word2vec, we will be using:

Contrastive Learning: With most machine learning, you take input X and expect output Y, and then train a model such that X fed through the model produces something close to Y. For embedding models, we don't have Y: we don't know what a good embedding would be beforehand.

Instead, with Contrastive Learning, we have multiple inputs X_1 and X_2, and a similarity. We feed both inputs through the model, after which we can *contrast* the two embeddings resulting in a predicted similarity. We can then push the embeddings further apart if the true similarity is low, or pull the embeddings closer together if the true similarity is high.

Matryoshka Representation Learning (MRL): Matryoshka Embedding Models (blogpost) is a clever training approach that allows users to truncate embedding models to smaller dimensions at a minimal performance hit. It involves using the contrastive loss function not just with the normal-sized embedding, but also with truncated versions of them. Consequently, the model learns to store information primarily at the start of the embeddings.

Truncated embeddings will be faster with downstream applications, such as retrieval, classification, and clustering.

uture research, we leave various other modern training approaches for improving data ity. See Next Steps for concrete ideas.

ining Requirements

ıown in the Training Overview documentation in Sentence Transformers, training ists of 3 to 5 components:

Dataset

Loss Function

Training Arguments (Optional)

Evaluator (Optional)

Trainer

e following sections, we'll go through our thought processes for each of these.

del Inspiration

ır experience, embedding models are either used 1) exclusively for retrieval or 2) for y task under the sun (classification, clustering, semantic textual similarity, etc.). We ut to train one of each.

he retrieval model, there is only a limited amount of multilingual retrieval training available, and hence we chose to opt for an English-only model. In contrast, we ded to train a multilingual general similarity model because multilingual data was h easier to acquire for this task.

hese models, we would like to use the `StaticEmbedding` module, which implements an ient `tokenize` method that avoids padding, and an efficient `forward` method that takes of computing and pooling embeddings. It's as simple as using a `torch` `EmbeddingBag`, h is nothing more than an efficient `Embedding` (i.e. a lookup table for embeddings) with n pooling.

an initialize it in a few ways: `StaticEmbedding.from_model2vec` to load a Model2Vec el, `StaticEmbedding.from_distillation` to perform Model2Vec-style distillation, or alizing it with a `Tokenizer` and an embedding dimension to get random weights.

d on our findings, the last option works best when fully training with a large amount ıta. Matching common models like all-mpnet-base-v2 or bge-large-en-v1.5, we are ısing an embedding dimensionality of 1024, i.e. our embedding vectors consist of 1024 es each.

lish Retrieval

he English Retrieval model, we rely on the `google-bert/bert-base-uncased` tokenizer. ıch, initializing the model looks like this:

```
rom sentence_transformers import SentenceTransformer
rom sentence_transformers.models import StaticEmbedding
rom tokenizers import Tokenizer

okenizer = Tokenizer.from_pretrained("google-bert/bert-base-uncased")
tatic_embedding = StaticEmbedding(tokenizer, embedding_dim=1024)

odel = SentenceTransformer(modules=[static_embedding])
```

first entry in the `modules` list must implement `tokenize`, and the last one must produce ed embeddings. Both is the case here, so we're good to start training this model.

tilingual Similarity

he Multilingual Similarity model, we instead rely on the `google-bert/bert-base-ilingual-uncased` tokenizer, and that's the only thing we change in our initialization :

```
rom sentence_transformers import SentenceTransformer
rom sentence_transformers.models import StaticEmbedding
rom tokenizers import Tokenizer

okenizer = Tokenizer.from_pretrained("google-bert/bert-base-multilingual-uncasec
tatic_embedding = StaticEmbedding(tokenizer, embedding_dim=1024)
```

```
odel = SentenceTransformer(modules=[static_embedding])
```

ining Dataset Selection

gside dozens of Sentence Transformer models, the Sentence Transformers nization on Hugging Face also hosts 70+ datasets (at the time of writing):

Embedding Model Datasets

ond that, many datasets have been tagged with `sentence-transformers` to mark that 're useful for training embedding models:

Datasets with the `sentence-transformers` tag

lish Retrieval

he English Retrieval datasets, we are primarily looking for any dataset with:

question-answer pairs, optionally with negatives (i.e. wrong answers) as well, and

no overlap with the BEIR benchmark, a.k.a. the Retrieval tab on MTEB. Our goal is to avoid training on these datasets so we can use MTEB as a 0-shot benchmark.

elected the following datasets:

gooaq

msmarco - the "triplet" subset

squad

s2orc - the "title-abstract-pair" subset

allnli - the "triplet" subset

paq

trivia_qa

msmarco_10m

swim_ir - the "en" subset

pubmedqa - the "triplet-20" subset

miracl - the "en-triplet-all" subset

mldr - the "en-triplet-all" subset

mr_tydi - the "en-triplet-all" subset

tilingual Similarity

he Multilingual Similarity datasets, we aimed for datasets with:

parallel sentences across languages, i.e. the same text in multiple languages, or

positive pairs, i.e. pairs with high similarity, optionally with negatives (i.e. low similarity).

elected the following datasets as they contain parallel sentences:

wikititles

tatoeba

talks

europarl

global_voices

muse

wikimatrix

opensubtitles

these datasets as they contain positive pairs of some kind:

stackexchange - the "post-post-pair" subset

quora - the "triplet" subset

[wikianswers_duplicates]

[all_nli] - the "triplet" subset

[simple_wiki]

[altlex]

[flickr30k_captions]

[coco_captions]

[nli_for_simcse]

[negation]

e

ling these datasets is rather simple, e.g.:

```
rom datasets import load_dataset, Dataset

ooaq_dataset = load_dataset("sentence-transformers/gooaq", split="train")
ooaq_dataset_dict = gooaq_dataset.train_test_split(test_size=10_000, seed=12)
ooaq_train_dataset: Dataset = gooaq_dataset_dict["train"]
ooaq_eval_dataset: Dataset = gooaq_dataset_dict["test"]

rint(gooaq_train_dataset)
""
ataset({
   features: ['question', 'answer'],
   num_rows: 3002496
)
""

rint(gooaq_eval_dataset)
""
ataset({
   features: ['question', 'answer'],
   num_rows: 10000
)
```

```
""
```

gooaq dataset doesn't already have a train-eval split, so we can make one with
n_test_split. Otherwise, we can just load a precomputed split with e.g. `split="eval"`.

that `train_test_split` does mean that the dataset has to be loaded into memory, reas it is otherwise just kept on disk. This increased memory is not ideal when training, s recommended to 1) load the data, 2) split it, and 3) save it to disk with `save_to_disk`. re training, you can then use `load_from_disk` to load it again.

s Function Selection

in Sentence Transformers, your loss model must match your training data format. The Overview is designed as an overview of which losses are compatible with which ats.

rticular, we currently have the following formats in our data:

(anchor, positive) pair, no label

(anchor, positive, negative) triplet, no label

(anchor, positive, negative_1, …, negative_n) tuples, no label

hese formats, we have some excellent choices:

`MultipleNegativesRankingLoss` (MNRL): Also known as in-batch negatives loss or InfoNCE loss, this loss has been used to train modern embedding models for a handful of years. In short, the loss optimizes the following:

> *"Given an anchor (e.g. a question), assign the highest similarity to the corresponding positive (i.e. answer) out of all positives and negatives (e.g. all answers) in the batch."*

If you provide the optional negatives, they will only be used as extra options (also known as in-batch negatives) from which the model must pick the correct positive.

Within reason, the harder this "picking" is, the stronger the model will become. Because of this, higher batch sizes result in more in-batch negatives, which then increase performance (to a point).

`CachedMultipleNegativesRankingLoss` (CMNRL): This is an extension of MNRL that implements GradCache, an approach that allows for arbitrarily increasing the batch size without increasing the memory.

This loss is recommended over MNRL *unless* you can already fit a large enough batch size in memory with just MNRL. In that case, you can use MNRL to save the 20% training speed cost that CMNRL adds.

`GISTEmbedLoss` (GIST): This is also an extension of MNRL, it uses a `guide` Sentence Transformer model to remove potential false negatives from the list of options that the model must "pick" the correct positive from.

False negatives can hurt performance, but hard true negatives (texts that are close to correct, but not quite) can help performance, so this filtering is a fine line to walk.

use these static embedding models are extremely small, it is possible to fit our desired h size of 2048 samples on our hardware: a single RTX 3090 with 24GB, so we don't l to use CMNRL.

tionally, because we're training such fast models, the `guide` from the `GISTEmbedLoss` ld make the training much slower. Because of this, we've opted to use `ipleNegativesRankingLoss` for our models.

were to try these experiments again, we would pick a larger batch size, e.g. 16384 CMNRL. If you try, please let us know how it goes!

e

usage is rather simple:

```
rom sentence_transformers import SentenceTransformer
rom sentence_transformers.losses import MultipleNegativesRankingLoss
```

```
 Prepare a model to train
okenizer = Tokenizer.from_pretrained("google-bert/bert-base-uncased")
tatic_embedding = StaticEmbedding(tokenizer, embedding_dim=1024)
odel = SentenceTransformer(modules=[static_embedding])

 Initialize the MNRL loss given the model
oss = MultipleNegativesRankingLoss(model)
```

ryoshka Representation Learning

ond regular loss functions, Sentence Transformers also implements a handful of Loss ifiers. These work on top of standard loss functions, but apply them in different ways y and instil useful properties into the trained embedding model.

ry interesting one is the `MatryoshkaLoss`, which turns the trained model into a *ryoshka Model*. This allows users to truncate the output embeddings at a minimal loss erformance, meaning that retrieval or clustering can be sped up due to the smaller ensionalities.

e

`MatryoshkaLoss` is applied on top of a normal loss. It's recommended to also include normal embedding dimensionality in the list of `matryoshka_dims`:

```
rom sentence_transformers import SentenceTransformer
rom sentence_transformers.losses import MultipleNegativesRankingLoss, Matryoshka

 Prepare a model to train
okenizer = Tokenizer.from_pretrained("google-bert/bert-base-uncased")
tatic_embedding = StaticEmbedding(tokenizer, embedding_dim=1024)
odel = SentenceTransformer(modules=[static_embedding])

 Initialize the MNRL loss given the model
ase_loss = MultipleNegativesRankingLoss(model)
oss = MatryoshkaLoss(model, base_loss, matryoshka_dims=[1024, 768, 512, 256, 128
```

ining Arguments Selection

ence Transformers supports a lot of training arguments, the most valuable of which been listed in the Training Overview > Training Arguments documentation.

ısed the same core training parameters to train both models:

`num_train_epochs`: 1

- We have sufficient data, should we want to train for more, then we can add more data instead of training with the same data multiple times.

`per_device_train_batch_size` / `per_device_eval_batch_size`: 2048

- 2048 dimensions fit comfortably on our RTX 3090. Various papers (Xiao et al., Li et al.) show that even larger batch sizes still improve performance. For future versions, we will apply `CachedMultipleNegativesRankingLoss` with a larger batch size, e.g. 16384.

`learning_rate`: 2e-1

- Note! This is *much* larger than with normal embedding model training, which often uses a loss around 2e-5.

`warmup_ratio`: 0.1

- 0.1 or 10% is a pretty standard warmup ratio to smoothly introduce the high learning rate to the model.

`bf16`: True

- If your GPU(s) support(s) `bf16` - it tends to make sense to train with it. Otherwise you can use `fp16=True` if that's supported instead.

`batch_sampler`: `BatchSamplers.NO_DUPLICATES`

- All losses with in-batch negatives (such as MNRL) benefit from this batch sampler that avoids duplicates within the batch. Duplicates often result in false negatives, weakening the trained model.

multi_dataset_batch_sampler: MultiDatasetBatchSamplers.PROPORTIONAL

- When you're training with multiple datasets, it's common that not all datasets are the same size. When that happens, you can either:
  - Round Robin: sample the same amount of batches from each dataset until one is exhausted. You'll have an equal distribution of data, but not all data will be used.
  - Proportional: sample each dataset until all are exhausted. You'll use up all data, but you won't have an equal distribution of data. We chose this one as we're not too concerned with a data imbalance.

›nd these core arguments, we also set a few training arguments for tracking and ıgging: `eval_strategy`, `eval_steps`, `save_strategy`, `save_steps`, `save_total_limit`, `ing_steps`, `logging_first_step`, and `run_name`.

.e

.e end, we used these `SentenceTransformerTrainingArguments` for the two models:

```
un_name = "static-retrieval-mrl-en-v1"
 or
 run_name = "static-similarity-mrl-multilingual-v1"

rgs = SentenceTransformerTrainingArguments(
   # Required parameter:
   output_dir=f"models/{run_name}",
   # Optional training parameters:
   num_train_epochs=1,
   per_device_train_batch_size=2048,
   per_device_eval_batch_size=2048,
   learning_rate=2e-1,
   warmup_ratio=0.1,
   fp16=False,  # Set to False if you get an error that your GPU can't run on FF
   bf16=True,  # Set to True if you have a GPU that supports BF16
```

```
batch_sampler=BatchSamplers.NO_DUPLICATES,  # MultipleNegativesRankingLoss be
multi_dataset_batch_sampler=MultiDatasetBatchSamplers.PROPORTIONAL,
# Optional tracking/debugging parameters:
eval_strategy="steps",
eval_steps=1000,
save_strategy="steps",
save_steps=1000,
save_total_limit=2,
logging_steps=1000,
logging_first_step=True,
run_name=run_name,  # Used if `wandb`, `tensorboard`, or `neptune`, etc. is i
```

luator Selection

: provide an evaluation dataset to the Sentence Transformer Trainer, then upon uation we will get an evaluation loss. This'll be useful to track whether we're overfitting ot, but not so meaningful when it comes to real downstream performance.

use of this, Sentence Transformers additionally supports Evaluators. Unlike the ing loss, these give qualitative metrics like NDCG, MAP, MRR for Information ieval, Spearman Correlation for Semantic Textual Similarity, or Triplet accuracy nber of samples where `similarity(anchor, positive)` > `similarity(anchor, tive)`).

to its simplicity, we will be using the `NanoBEIREvaluator` for the retrieval model. This uator runs Information Retrieval benchmarks on the NanoBEIR collection of datasets. dataset is a subset of the much larger (and thus slower) BEIR benchmark, which is monly used as the Retrieval tab in the MTEB Leaderboard.

e

use all datasets are already pre-defined, we can load the evaluator without any ments:

```
rom sentence_transformers import SentenceTransformer
rom sentence_transformers.evaluation import NanoBEIREvaluator

 Load an example pre-trained model to finetune further
odel = SentenceTransformer("sentence-transformers/all-MiniLM-L6-v2")

 Initialize the NanoBEIR Evaluator
valuator = NanoBEIREvaluator()

 Run it on any Sentence Transformer model
valuator(model)
```

rdware Details

e training these models on consumer-level hardware, specifically:

GPU: RTX 3090

CPU: i7-13700K

RAM: 32GB

erall Training Scripts

section contains the final training scripts for both models with all of the previously ribed components (datasets, loss functions, training arguments, evaluator, trainer) bined.

lish Retrieval

ick to expand

script produced sentence-transformers/static-retrieval-mrl-en-v1 after 17.8 hours of ing. In total, it consumed 2.6 kWh of energy and emitted 1kg of CO_2. That is roughly valent to the amount of CO_2 an average person exhales per day.

our Weights and Biases report for the training and evaluation metrics collected during ing.

tilingual Similarity

ick to expand

script produced sentence-transformers/static-similarity-mrl-multilingual-v1 after 3.1 s of training. In total, it consumed 0.5 kWh of energy and emitted 0.2kg of CO_2. That ughly 20% of the CO_2 that an average person exhales per day.

our Weights and Biases report for the training and evaluation losses collected during ing.

age

usage of these models is very straightforward, identical to the normal Sentence sformers flow:

glish Retrieval

```
rom sentence_transformers import SentenceTransformer

 Download from the 🤗 Hub
odel = SentenceTransformer("sentence-transformers/static-retrieval-mrl-en-v1", c
 Run inference
entences = [
   'Gadofosveset-enhanced MR angiography of carotid arteries: does steady-state 
   'To evaluate the diagnostic accuracy of gadofosveset-enhanced magnetic resona
   'In a longitudinal study we investigated in vivo alterations of CVO during ne

mbeddings = model.encode(sentences)
rint(embeddings.shape)
 [3, 1024]
```

```
 Get the similarity scores for the embeddings
imilarities = model.similarity(embeddings[0], embeddings[1:])
rint(similarities)
 tensor([[0.7649, 0.3279]])
```

upcoming Performance > English Retrieval section will show that these results are
 solid, within 15% of commonly used Transformer-based encoder models like all-
et-base-v2.

`SentenceTransformer` API Reference.

`SentenceTransformer.encode` API Reference.

`SentenceTransformer.similarity` API Reference.

ltilingual Similarity

```
rom sentence_transformers import SentenceTransformer

 Download from the 🤗 Hub
odel = SentenceTransformer("sentence-transformers/static-similarity-mrl-multilin
 Run inference
entences = [
   'It is known for its dry red chili powder.',
   'It is popular for dried red chili powder.',
   'These monsters will move in large groups.',

mbeddings = model.encode(sentences)
rint(embeddings.shape)
 [3, 1024]

 Get the similarity scores for the embeddings
imilarities = model.similarity(embeddings, embeddings)
rint(similarities)
 tensor([[ 1.0000,  0.8388, -0.0012],
         [ 0.8388,  1.0000,  0.0445],
```

```
        [-0.0012,  0.0445,  1.0000]])
```

model only loses about 8% of performance compared to the popular but much slower ilingual-e5-small, as shown in the upcoming Performance > Multilingual Similarity on.

`SentenceTransformer` API Reference.

`SentenceTransformer.encode` API Reference.

`SentenceTransformer.similarity` API Reference.

tryoshka Dimensionality Truncation

educe the dimensionality of your calculated embeddings, you can simply pass the `cate_dim` parameter. This works for all Sentence Transformer models.

```
rom sentence_transformers import SentenceTransformer

 Download from the 🤗 Hub
odel = SentenceTransformer(
   "sentence-transformers/static-retrieval-mrl-en-v1",
   device="cpu",
   truncate_dim=256,

 Run inference
entences = [
   'Gadofosveset-enhanced MR angiography of carotid arteries: does steady-state
   'To evaluate the diagnostic accuracy of gadofosveset-enhanced magnetic resona
   'In a longitudinal study we investigated in vivo alterations of CVO during ne

mbeddings = model.encode(sentences)
rint(embeddings.shape)
 [3, 256]

 Get the similarity scores for the embeddings
```

```
imilarities = model.similarity(embeddings[0], embeddings[1:])
rint(similarities)
 tensor([[0.7844, 0.3561]])
```

rd Party libraries

 model also works out of the box in various third party libraries, for example
Chain, LlamaIndex, Haystack, and txtai.

gChain

```
 pip install langchain langchain_huggingface
rom langchain_huggingface import HuggingFaceEmbeddings

odel_name = "sentence-transformers/static-retrieval-mrl-en-v1"
odel_kwargs = {'device': 'cpu'} # you can use 'truncate_dim' here
odel = HuggingFaceEmbeddings(
   model_name=model_name,
   model_kwargs=model_kwargs,
```

HuggingFaceEmbeddings documentation.

naIndex

```
 pip install llama-index llama-index-embeddings-huggingface
rom llama_index.core import Settings
rom llama_index.embeddings.huggingface import HuggingFaceEmbedding

 Set up the HuggingFaceEmbedding class with the required model to use with llama
odel_name = "sentence-transformers/static-retrieval-mrl-en-v1"
evice = "cpu"
mbed_model = HuggingFaceEmbedding(
   model_name=model_name,
```

```
    device=device,
    # truncate_dim=256, # you can use 'truncate_dim' here

ettings.embed_model = embed_model
```

HuggingFaceEmbedding documentation and API Reference.

stack

```
 pip install haystack sentence-transformers
rom haystack.components.embedders import (
    SentenceTransformersDocumentEmbedder,
    SentenceTransformersTextEmbedder,


odel_name = "sentence-transformers/static-retrieval-mrl-en-v1"
evice = "cpu"
ocument_embedder = SentenceTransformersDocumentEmbedder(
    model=model_name,
    device=device,
    # truncate_dim=256, # you can use 'truncate_dim' here

ext_embedder = SentenceTransformersTextEmbedder(
    model=model_name,
    device=device,
    # truncate_dim=256, # you can use 'truncate_dim' here
```

SentenceTransformersDocumentEmbedder documentation.

SentenceTransformersTextEmbedder documentation.

i

```
 pip install txtai sentence-transformers
```

```
rom txtai import Embeddings

odel_name = "sentence-transformers/static-retrieval-mrl-en-v1"
nbeddings = Embeddings(path=model_name)
```

`Embeddings` documentation

rformance

glish Retrieval

r training, we've evaluated the final model sentence-transformers/static-retrieval-mrl-1 on NanoBEIR (normal dimensionality and with Matryoshka dimensions) as well as EIR.

oBEIR

e evaluated sentence-transformers/static-retrieval-mrl-en-v1 on NanoBEIR and ed it against the inference speed computed on our hardware. For the inference speed , we calculated the number of computed query embeddings of the GooAQ dataset per nd, either on CPU or GPU.

valuate against 3 types of models:

Attention-based dense embedding models, e.g. traditional Sentence Transformer models like `all-mpnet-base-v2`, `bge-base-en-v1.5`, and `gte-large-en-v1.5`.

Static Embedding-based models, e.g. `static-retrieval-mrl-en-v1`, `potion-base-8M`, `M2V_base_output`, and `glove.6B.300d`.

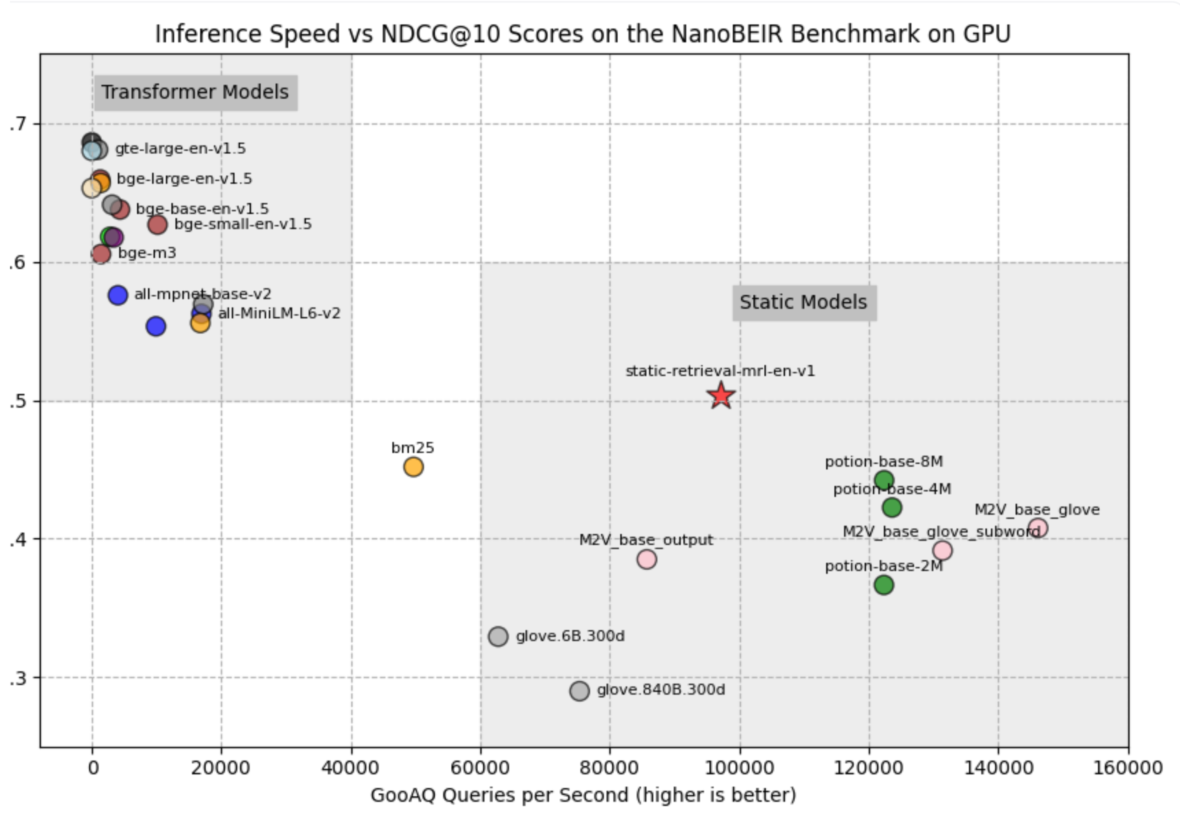
Sparse bag-of-words model, BM25, often a strong baseline.

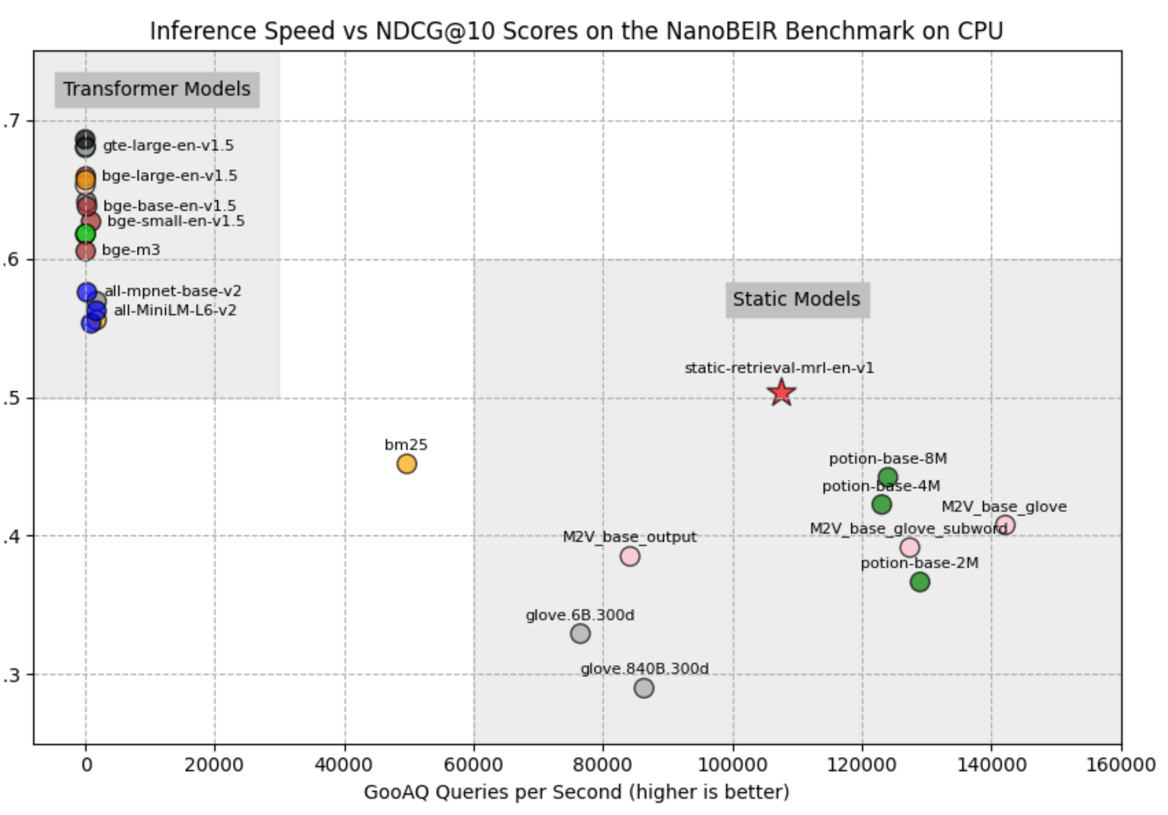
▶ Click to expand BM25 implementation details

'OTE: *Many of the attention-based dense embedding models are finetuned on the training splits of the 'ano)BEIR evaluation datasets. This gives the models an unfair advantage in this benchmark and can sult in lower downstream performance on real retrieval tasks.*

tic-retrieval-mrl-en-v1 is purposefully not trained on any of these datasets."

ick to see a table with all values from the next 2 Figures





:an draw some notable conclusions from these figures:

`static-retrieval-mrl-en-v1` outperforms all other Static Embedding models, like GloVe or Model2Vec.

`static-retrieval-mrl-en-v1` is the only Static Embedding model to outperform BM25.

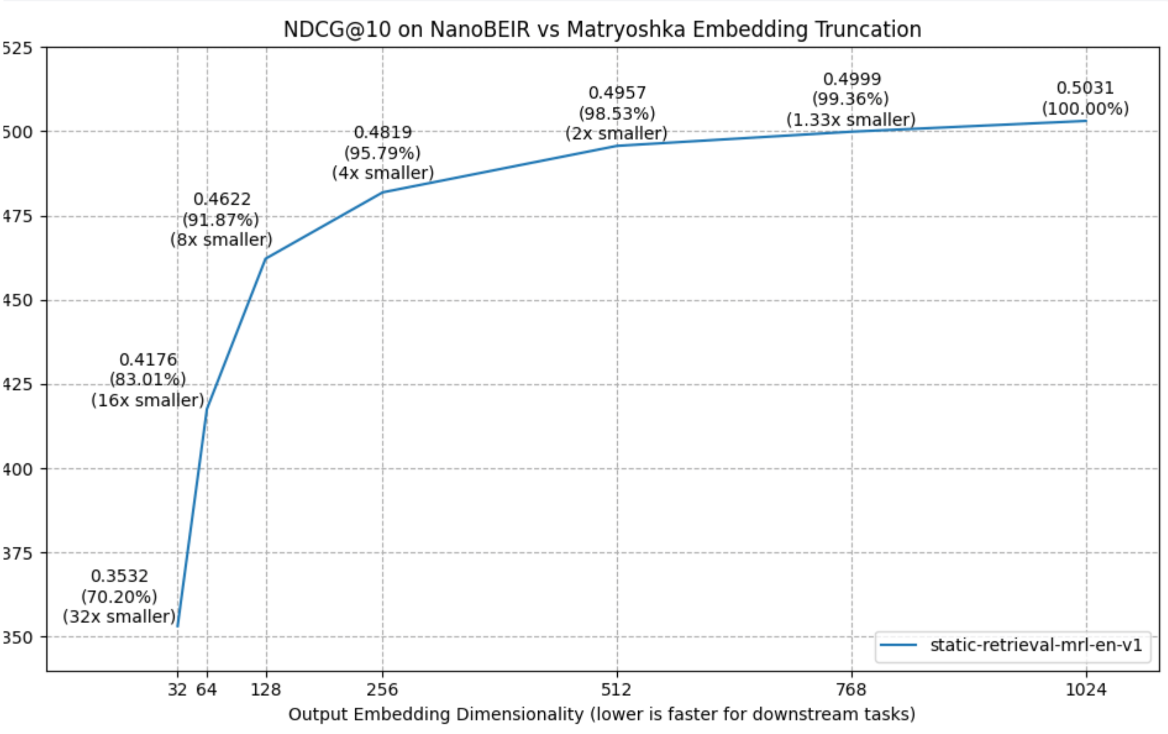
`static-retrieval-mrl-en-v1` is

- **87.4%** as performant as the commonly used `all-mpnet-base-v2`,
- **24x** faster on GPU,
- **397x** faster on CPU.

`static-retrieval-mrl-en-v1` is quicker on CPU than on GPU: This model can run extraordinarily quickly everywhere, including consumer-grade PCs, tiny servers, phones, or in-browser.

ryoshka Evaluation

:tionally, we experimented with the results on NanoBEIR performance when we ormed Matryoshka-style dimensionality reduction by truncating the output eddings to a lower dimensionality.

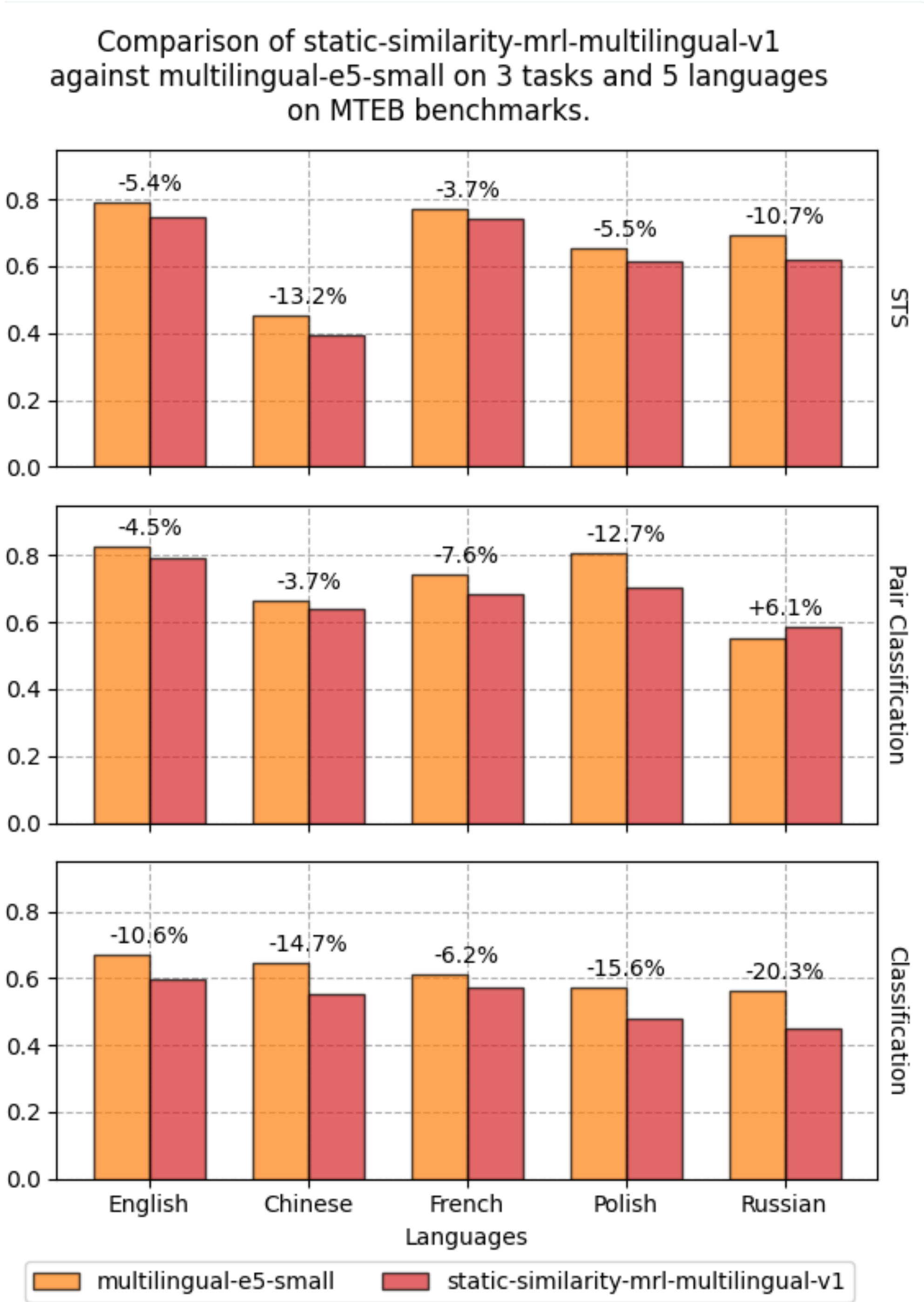


e findings show that reducing the dimensionality by e.g. 2x only has a 1.47% ction in performance (0.5031 NDCG@10 vs 0.4957 NDCG@10), while realistically lting in a 2x speedup in retrieval speed.

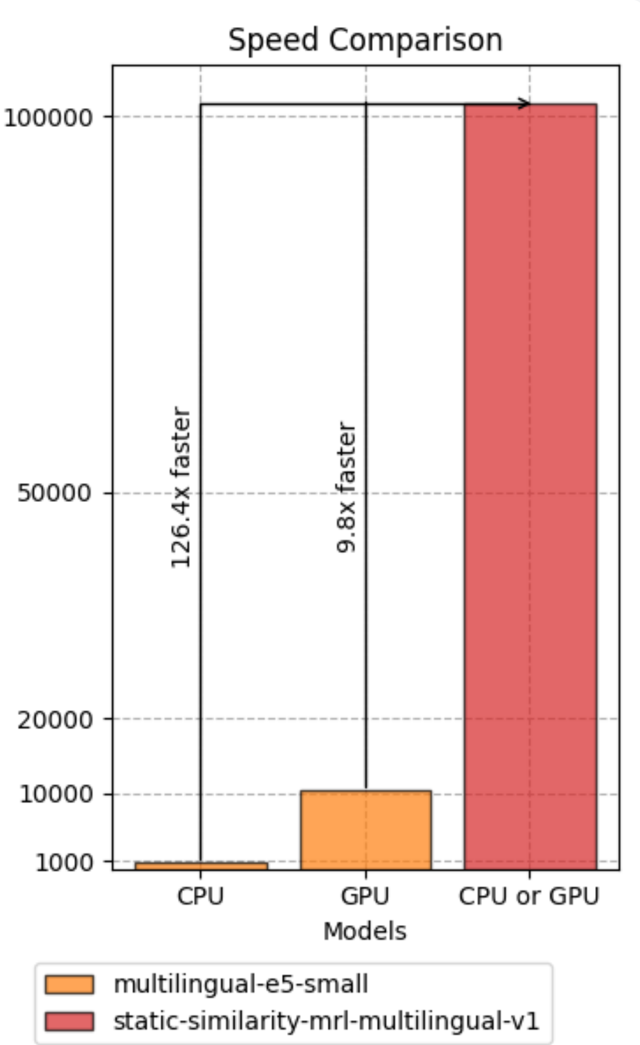
ltilingual Similarity

e additionally evaluated the final sentence-transformers/static-similarity-mrl-ilingual-v1 model on 5 languages which have a lot of benchmarks across various tasks ITEB.

vant to reiterate that this model is not intended for retrieval use cases. Instead, we uate on Semantic Textual Similarity (STS), Classification, and Pair Classification. We pare against the excellent and small multilingual-e5-small model.



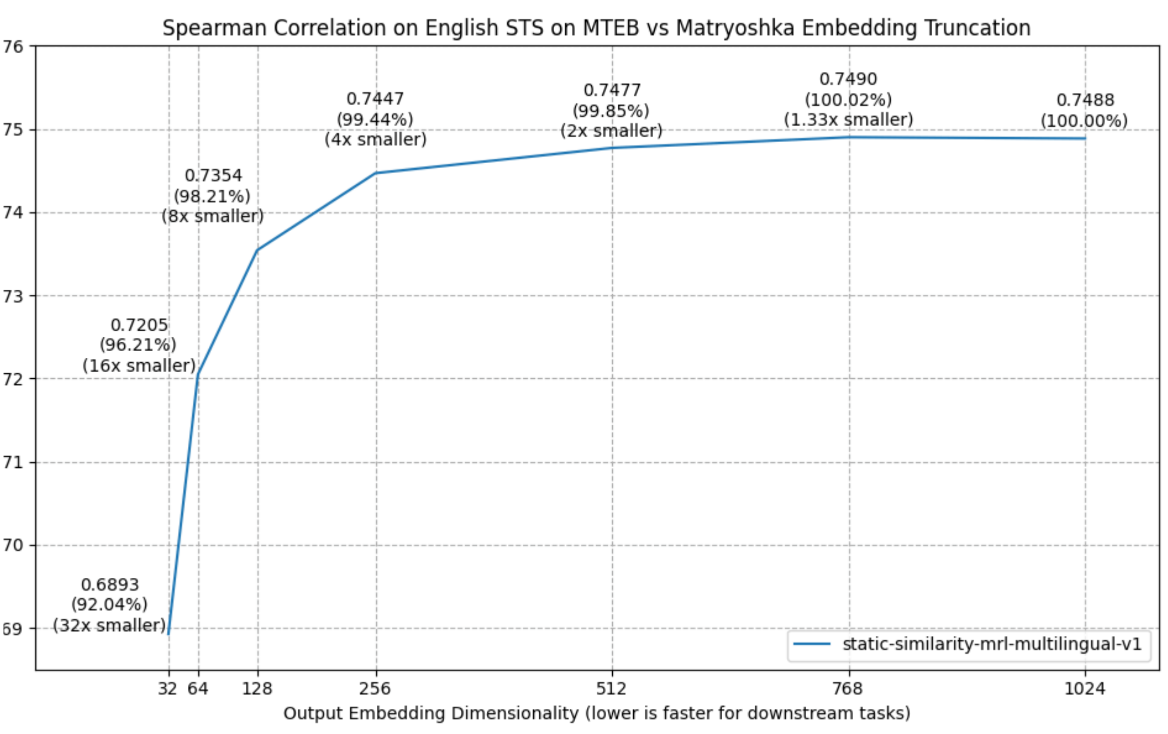
ss all measured languages, static-similarity-mrl-multilingual-v1 reaches an average % for STS, **95.52%** for Pair Classification, and **86.52%** for Classification relative to ilingual-e5-small.



ıake up for this performance reduction, static-similarity-mrl-multilingual-v1 is oximately ~125x faster on CPU and ~10x faster on GPU devices than multilingual-e5-l. Due to the super-linear nature of attention models, versus the linear nature of static edding models, the speedup will only grow larger as the number of tokens to encode ɛases.

ryoshka Evaluation

y, we experimented with the impacts on English STS on MTEB performance when we Matryoshka-style dimensionality reduction by truncating the output embeddings to a r dimensionality.



ou can see, you can easily reduce the dimensionality by 2x or 4x with minor (0.15% or %) performance hits. If the speed of your downstream task or your storage costs are a eneck, this should allow you to alleviate some of those concerns.

nclusion

blogpost described all of the steps that we undertook from ideation to finished els, in addition to details regarding usage and evaluation of the two resulting models: c-retrieval-mrl-en-v1 and static-similarity-mrl-multilingual-v1.

evaluations show that:

- Static Embedding-based models can exceed **85%** of the performance of common attention-based dense models,
- Static Embedding-based models are realistically **10x to 25x faster on GPUs** and **100x to 400x faster on CPUs** than common efficient alternatives like all-mpnet-base-v2 and multilingual-e5-small. This speedup only grows larger with longer texts.
- Training with a Matryoshka Loss allows significant preservation of downstream performance:

- **4x smaller** gives a **0.56% performance decrease** by static-similarity-mrl-multilingual-v1 for English STS, and
- **2x smaller** gives a **1.47% performance decrease** by static-retrieval-mrl-en-v1 for English Retrieval.

ıld you need an efficient CPU-only dense embedding model for your retrieval or larity tasks, then static-retrieval-mrl-en-v1 and static-similarity-mrl-multilingual-v1 be extremely performant solutions at minimal costs that get surprisingly close to the ıtion-based dense models.

xt Steps

t out! If you already use a Sentence Transformer model somewhere, feel free to swap it or static-retrieval-mrl-en-v1 or static-similarity-mrl-multilingual-v1. Or, better yet: ı your own models on data that is representative for the task and language of your est.

hermore, some questions remain about the trained models:

Because Static Embedding-based models aren't bottlenecked by positional embeddings or superlinear time complexity, they can have arbitrarily high maximum sequence lengths. However, at some point the law of large numbers is likely to "normalize" all embeddings for really long documents, such that they aren't useful anymore.

More experiments are required to determine what a good cutoff point is. For now, we leave the maximum sequence length, chunking, etc. to the user.

tionally, there are quite a few possible extensions that are likely to improve the ormance of this model, which we happily leave to other model authors. We are also ı to collaborations:

Hard Negatives Mining: Search for similar, but not quite relevant, texts to improve training data difficulty.

Model Souping: Combining weights from multiple models trained in the same way with different seeds or data distributions.

Curriculum Learning: Train on examples of increasing difficulties.

Guided False In-Batch Negatives Filtering: Exclude false negatives via an efficient pre-trained embedding model.

Seed Optimization for the Random Weight Initialization: Train the first steps with various seeds to find one with a useful weight initialization.

Tokenizer Retraining: Retrain a tokenizer with modern texts and learnings.

Gradient Caching: Applying GradCache via `CachedMultipleNegativesRankingLoss` allows for larger batches, which often result in superior performance.

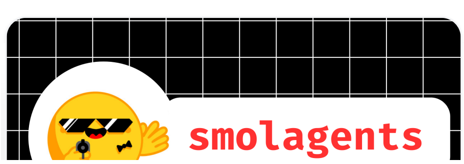
Model Distillation: Rather than training exclusively using supervised training data, we can also feed unsupervised data through a larger embedding model and distil those embeddings into the static embedding-based student model.

knowledgements

uld like to thank Stéphan Tulkens and Thomas van Dongen of The Minish Lab for ging Static Embedding models to my attention via their Model2Vec work. Additionally, uld like to thank Vaibhav Srivastav and Pedro Cuenca for their assistance with this post, and Antoine Chaffin for brainstorming the release checkpoints.

y, a big thanks to all researchers working on embedding models, datasets, and open ce Python packages. You strengthen the industry, and I build on your shoulders. One I hope you build on mine.

More Articles from our Blog



Llamaindex
vdr-2b-multi-v1

Visual Document Retrieval Goes Multilingual

By marco January 9, 2… • △ 64

Introducing smolagents: simple agents that write actions in code.

By m-ric December 30… • △ 531

---

nunity

13 days ago

ıu need to buy GPUs and machines.
r: No, we can just tweak the algorithms.
‹: Look at me buying GPUs, you poor folks.

Reply

\icky 13 days ago

u very much for the post, great work,

ɜady trained some English and Spanish models:

yNicky/StaticEmbedding-MatryoshkaLoss-gemma-2-2b-en-es

yNicky/StaticEmbedding-MatryoshkaLoss-gemma-2-2b-gooaq-en

‹e to know how to increase or decrease the
ȝth example 371'

eck 'print(model.max_seq_length) # -> Inf'.

ble, how? I can't find documentation about it

ı so much

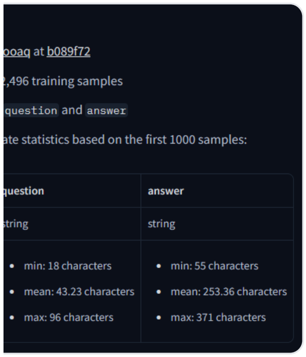
plies · 🔥 3 +

aarsen Article author 13 days ago

rk on those models! Am I correct in understanding that one of those models reaches 0.5623 NDCG@10 oBEIR across all datasets? That's a pretty huge jump from the 0.5032 NDCG@10 for static-retrieval-mrl-en-

uld like to know how to increase or decrease the _length example 371' "

referring to the 'max' in your model card here?



ooaq at b089f72

,496 training samples

question and answer

ate statistics based on the first 1000 samples:

question	answer
tring	string
• min: 18 characters • mean: 43.23 characters • max: 96 characters	• min: 55 characters • mean: 253.36 characters • max: 371 characters

simply some approximate statistics on the training data; taken from the first 1000 samples. Although it's ays recommended to use texts with (much) larger sequence lengths than the training data, the actual um sequence length is indeed infinity. It is defined here: https://github.com/UKPLab/sentence-rmers/blob/cccab8303aaf6e18f069b0da578b3d162bf8442a/sentence_transformers/models/StaticEmbed #L106-L108

: the model will never truncate sequences, because the approach

s linear complexity (2x more data -> 2x slower) unlike Transformer models (2x more data -> (much) slower an 2x).

not beholden to positional embeddings that might impose limitations on the maximum sequence length.

ic Models don't have a maximum sequence length. They just require care by the user to make sure that not feeding documents that are too large, as all documents will eventually embed very similarly if they g enough.

m Aarsen

replies

n thread

queaker 13 days ago

lly cool! I'm surprised you do better than model2vec - is the difference really just the use of a (better) ve loss pretraining formula?

ies · 🧠 2 +

aarsen Article author 13 days ago

e architecture is identical. In fact, the `StaticEmbedding` module that is used for the models described
logpost is actually the same that is used when loading a Model2Vec model in Sentence Transformers:

```
 sentence_transformers import SentenceTransformer
 sentence_transformers.models import StaticEmbedding
 tokenizers import Tokenizer

e-distilled embeddings:
ic_embedding = StaticEmbedding.from_model2vec("minishlab/M2V_base_output")

l = SentenceTransformer(modules=[static_embedding])

ddings = model.encode(["What are Pandas?", "The giant panda (Ailuropoda melanoleuca
larity = model.similarity(embeddings[0], embeddings[1])
nsor([[0.9177]]) (If you use the distilled bge-base)
```

taticEmbedding docs

replies

n thread

sv 13 days ago

work and excellent writing!

+ Reply

ynman 11 days ago

s training on a subset of data (AllNLI, GooAQ, MSMacro, PAQ, S2ORC) with batch size 16384. Took 5 hours.

i.wandb.ai/links/arunarumugam411-sui/dkcwm6gs

. looks cool!

+

Reply

321 11 days ago

great, but

larify idea behind . Do you calculate emceeing for each token and them average them ?

hare link to NanoBEIR ?

· +

aarsen Article author 11 days ago

;, the implementation is just https://pytorch.org/docs/stable/generated/torch.nn.EmbeddingBag.html. In ort: token -> token embedding via lookup -> text embedding via mean pooling (averaging per dimension).

noBEIR

tasets: https://huggingface.co/collections/zeta-alpha-ai/nanobeir-66e1a0af21dfd93e620cd9f6

cumentation:
ps://sbert.net/docs/package_reference/sentence_transformer/evaluation.html#nanobeirevaluator

vas introduced in this blogpost: https://www.zeta-alpha.com/post/fine-tuning-an-llm-for-state-of-the--retrieval-zeta-alpha-s-top-10-submission-to-the-the-mteb-be

ı thread

321 11 days ago

ıve detailed description like paper pls

Reply

321 11 days ago

:larify

ıggingface.co/blog/Pringled/model2vec

e a simple mean over tokens in the space, it is important that the vectors are weighted correctly.
, a sentence transformer would be there to correctly weight all the tokens for us given the context, but we

e that luxury any more. Intuitively, we would like to use something like Inverse Document Frequency ɔwn-weight very frequent or uninteresting words. But we don't have access to a corpus over which to document frequencies.

ıme this, we opt to use a well-known principle from language sciences, which is that, given a frequency t, the frequency of the items in that list follow a power law distribution. This is called Zipf's law. So, if we ıssumption that a vocabulary is ranked by frequency, we can accurately down-weight really frequent hout needing to have access to actual frequencies. As tokenizer vocabularies are sorted by frequency, we ave access to a ranked list, so this optimization can be applied without any additional work

othetical Zipf input
0.7] , [1.2, 0.9,0.2], [0.4, 0.3, 0.2] ,[1.3, 2.4, 3.2]]

according to each vector norm
t
0.2] , [0.2,0.5,0.7] , [1.2, 0.9,0.2],[1.3, 2.4, 3.2]]

e each vector by its norm

0.2]/n1 , [0.2,0.5,0.7]/n2 , [1.2, 0.9,0.2] /n3 ,[1.3, 2.4, 3.2]/n4]

embedding is mean of this down-weighted vectors ?
0.2]/n1 + [0.2,0.5,0.7]/n2 + [1.2, 0.9,0.2] /n3 + [1.3, 2.4, 3.2]/n4) / 4

ct algorithm ?

Reply

onnesoeur 11 days ago

work there. This technique is quite eye opening really ^^

omment on the title of the post though, at first, I believed that this post was about training sentence ng model faster, not about training sentence embedding models that have a faster inference time. Just ıys to knoew.

Reply

otch 8 days ago

antastic approach!

Static Embedding Japanese model (static-embedding-japanese) by incorporating a large amount of datasets, and when we compared it using the Japanese Multilingual Text Embedding Benchmark