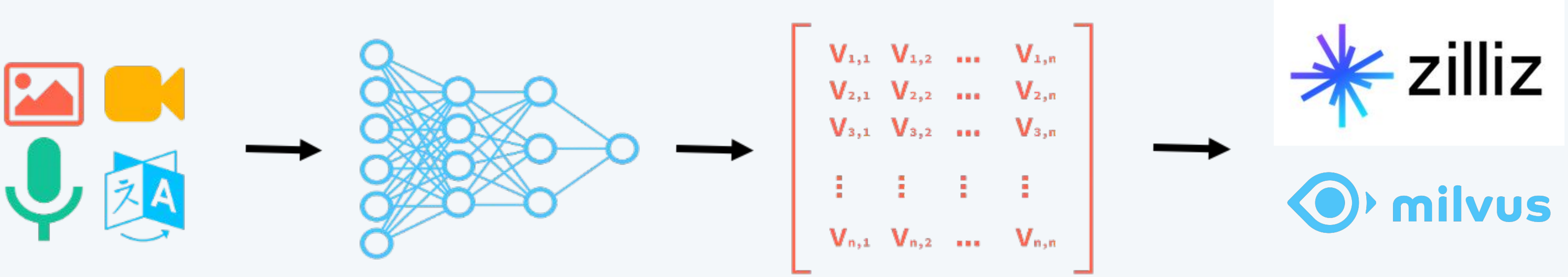
zilliz

Sparse and Dense Embeddings

Frank Liu

A Quick Refresher

Vectors unlock unstructured data



Knowledge Base (Documents)

Embedding Models

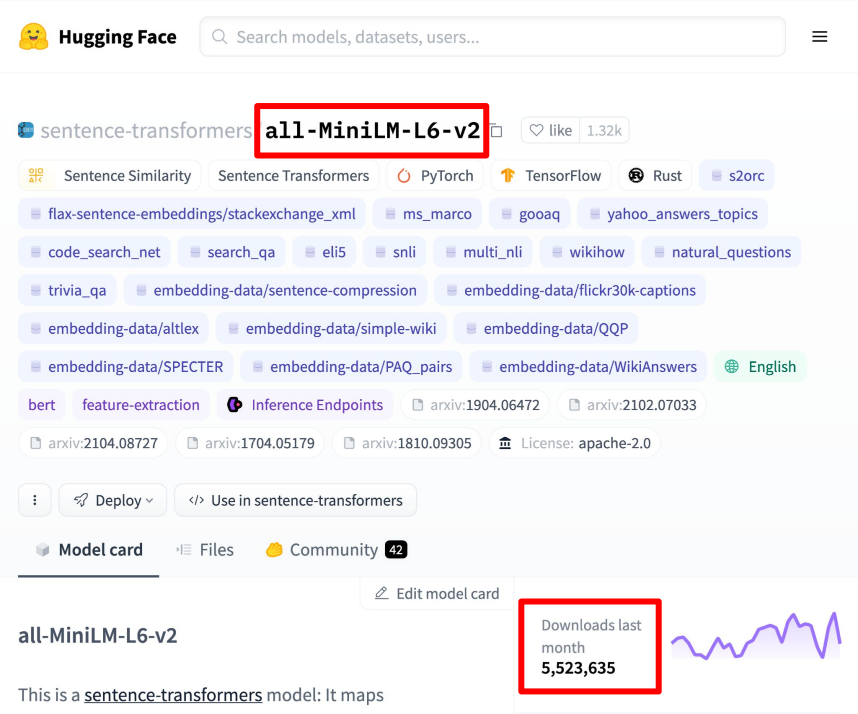
Vectors

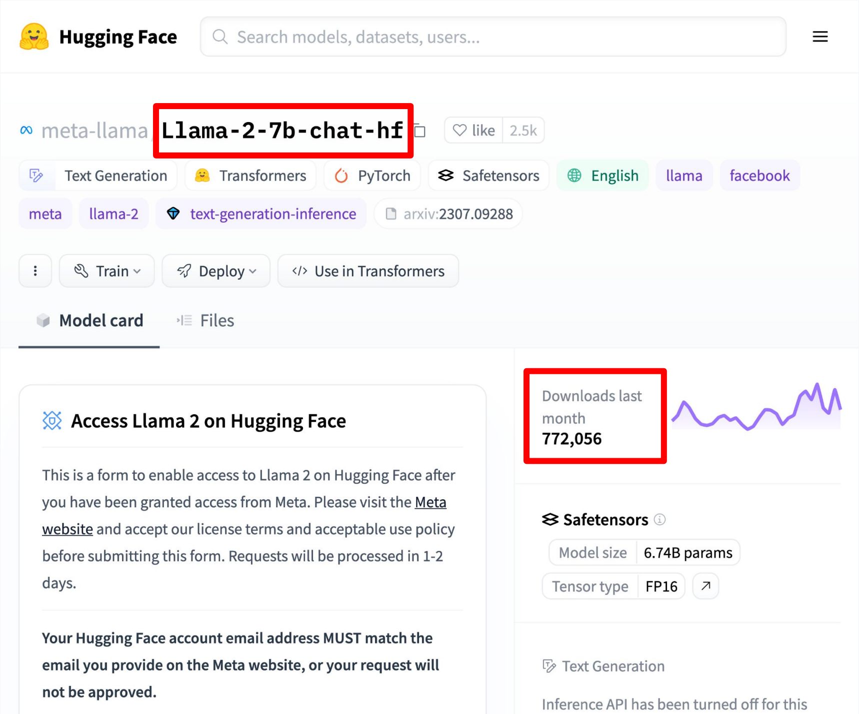
Vector Databases

 |

zilliz

Embeddings models are workhorses of AI apps





zilliz

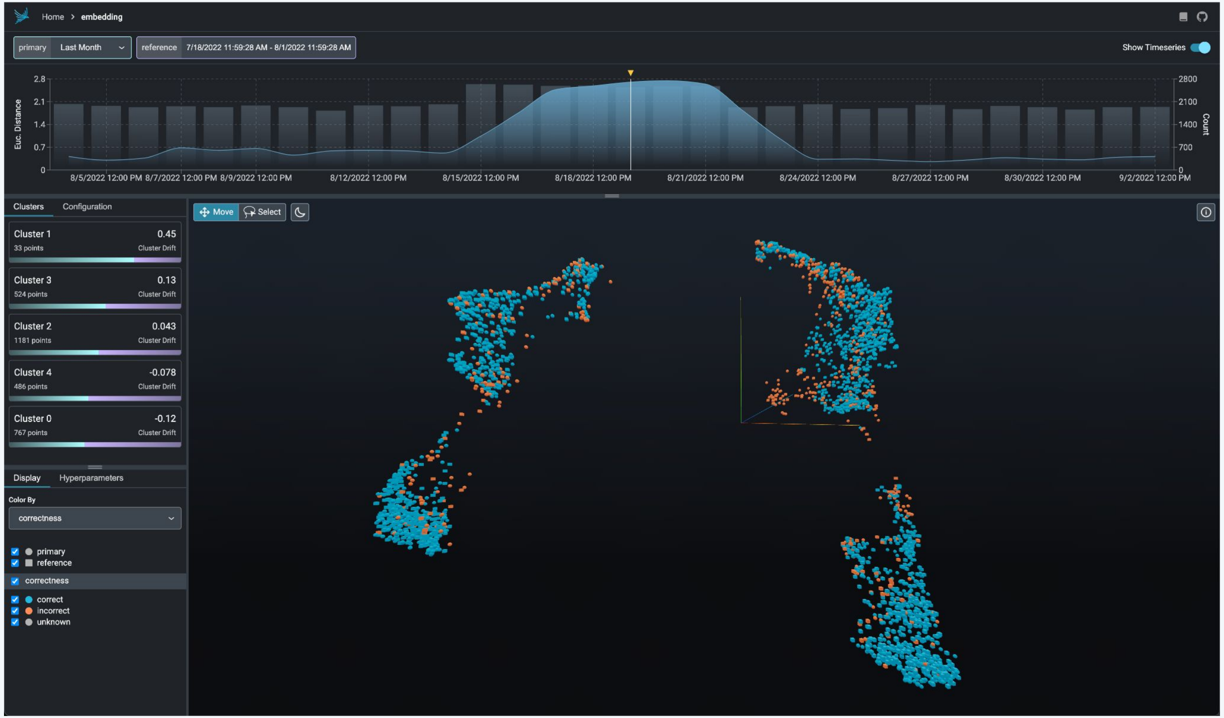
There are lots of embedding models out there

Rank	Model	Average	AskUbuntuDupQuestions	MindSmallReranking	SciDocsRR	StackOverflowDupQuestions
1	e5-mistral-7b-instruct	60.21	66.98	32.6	86.33	54.91
2	ember-v1	60.04	64.46	32.27	87.56	55.85
3	bge-large-en-v1.5	60.03	64.47	32.06	87.63	55.95
4	UAE-Large-V1	59.88	64.2	32.51	87.49	55.32
5	sf_model_e5	59.86	64.32	32.27	87.47	55.4
6	voyage-lite-01-instruct	59.74	65.77	31.69	87.03	54.49
7	all-mpnet-base-v2	59.36	65.85	30.97	88.65	51.98
8	gte-large	59.13	63.06	32.63	87.2	53.63
9	bge-base-en-v1.5-quant	58.94	62.39	31.89	87.05	54.45
10	bge-base-en-v1-5-seqlen-384-bs-1	58.86	62.13	31.2	87.49	54.61
11	bge-base-en-v1.5	58.86	62.13	31.2	87.49	54.61
12	stella-base-en-v2	58.78	62.72	31.91	86.66	53.81

Source: [MTEB Leaderboard]

zilliz

Visualizing dense embeddings

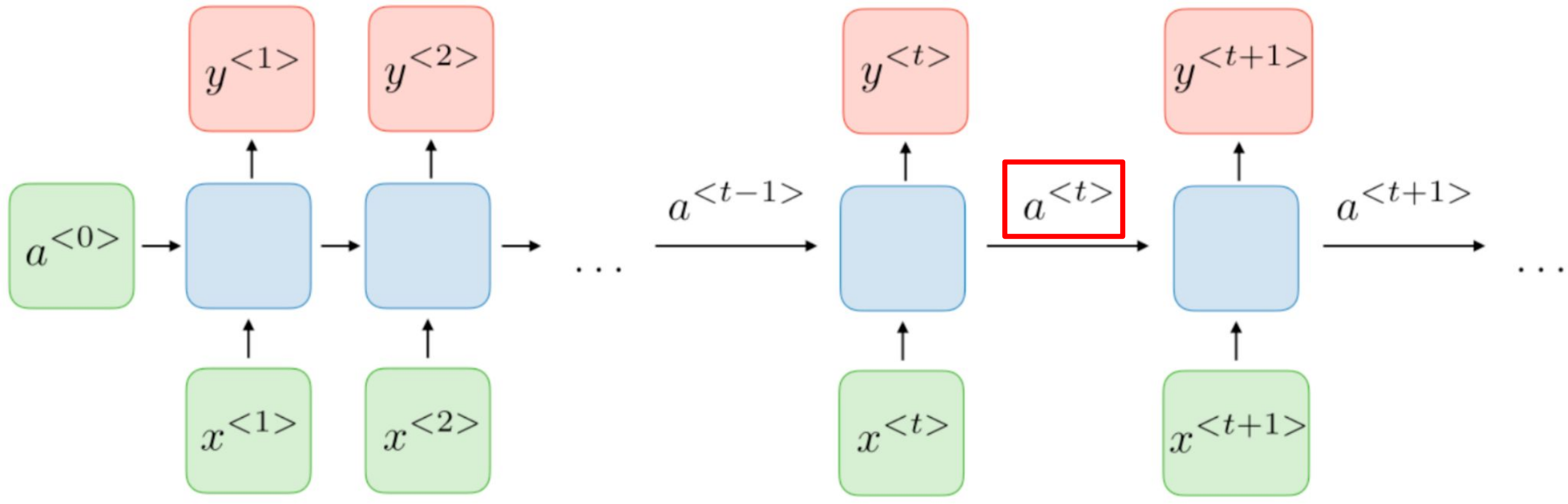


Source: [Arize Phoenix]

 |

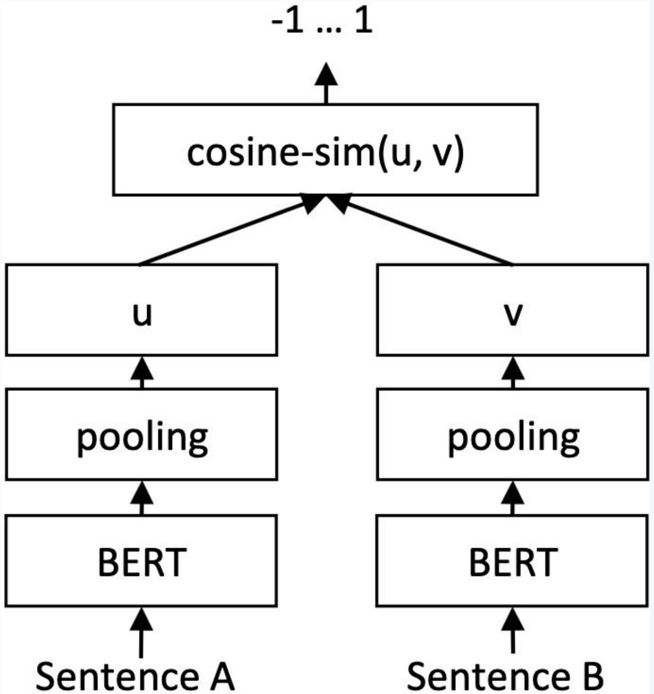
zilliz

Recurrent neural networks



Source: [CS230 notes]

Sentence BERT



Source: [SBERT paper]

Bi-encoder
1 embedding per text

Sparse Embeddings

Dense embeddings are great...

 |

Dense embeddings are great…

… but they lack lexical information

 |

We’re used to searching lexically, not semantically

We’re used to searching lexically, not semantically

- Here are some common Google searches
  - “rubik’s cube algorithm”
  - “cups in a quart”
  - “how to tie a tie”

We’re used to searching lexically, not semantically

- Here are some common Google searches
  - “rubik’s cube algorithm”
  - “cups in a quart”
  - “how to tie a tie”

- Keywords play an important role

 |

We're used to searching lexically, not semantically

- Here are some common Google searches
  - “rubik’s cube algorithm”
  - “cups in a quart”
  - “how to tie a tie”

- Keywords play an important role
  - Lexical search is superior for out-of-domain data

 |

We can combine sparse and dense embeddings

 |

We can combine sparse and dense embeddings

Ranking	BM25 *with title boosting*	BM25 *with content boosting*	Semantic
1	Document-2	Document-3	Document-4
2	Document-3	Document-5	Document-2
3	Document-5	Document-2	Document-5
4	Document-1	Document-1	Document-3
5	Document-4	Document-4	Document-1

Let's calculate RRF for each document and rerank:

Document-1	1/4 + 1/4 + 1/5	= 0.7
Document-2	1/1 + 1/3 + 1/2	= 0.83
Document-3	1/2 + 1/1 + 1/4	=1.75
Document-4	1/5 + 1/5 + 1/1	=1.4
Document-5	1/3+ 1/2 + 1/3	=1.16

⇒

Document-2
Document-3
Document-4
Document-5
Document-1

Source: Sowmiya Jaganathan

 |

zilliz

Sparse Embedding Algorithms

zilliz

Pure lexical sparse embeddings (e.g. TF-IDF)

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term *x* within document *y*

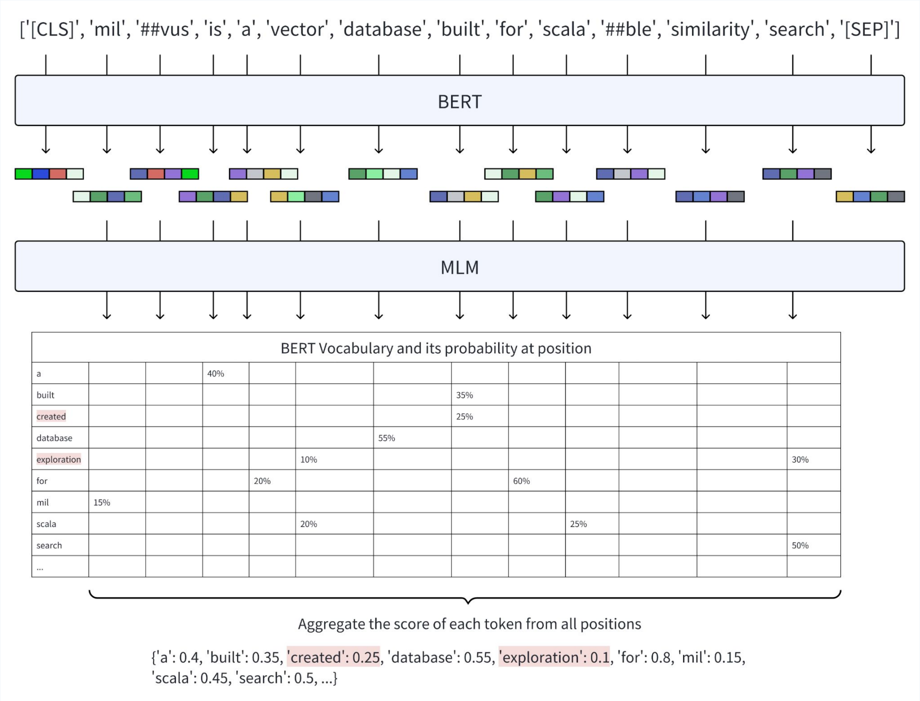
$tf_{x,y}$ = frequency of *x* in *y*

df_x = number of documents containing *x*

N = total number of documents

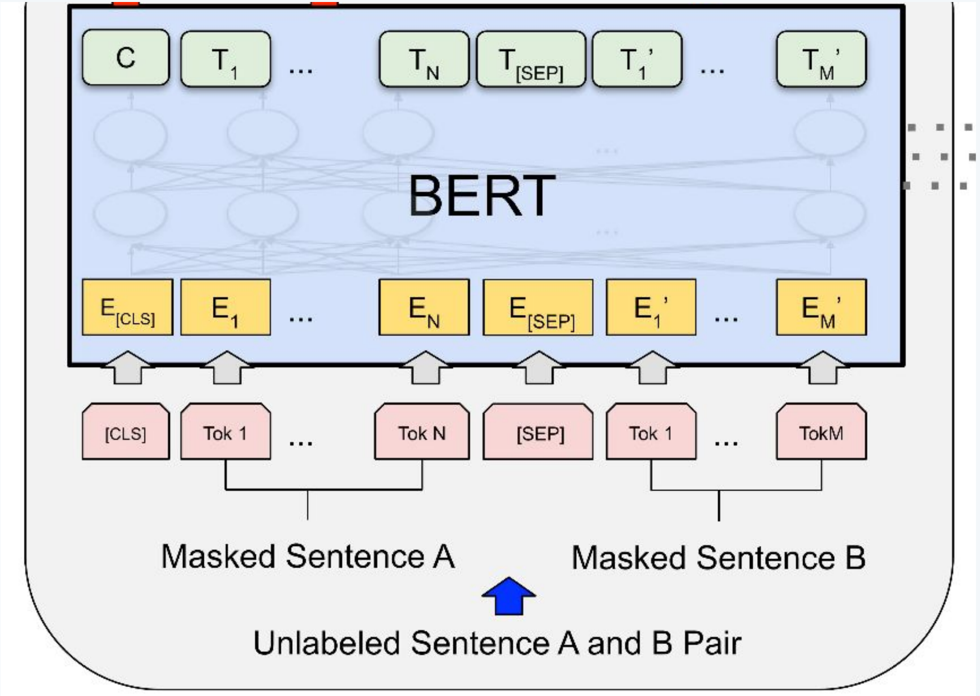
Source: [Ted Mei]

“Learned” sparse embeddings (e.g. SPLADE)



Source: [Buqian Zheng]

Bonus: ColBERT



Source: [BERT paper]

 |

Bonus: ColBERT

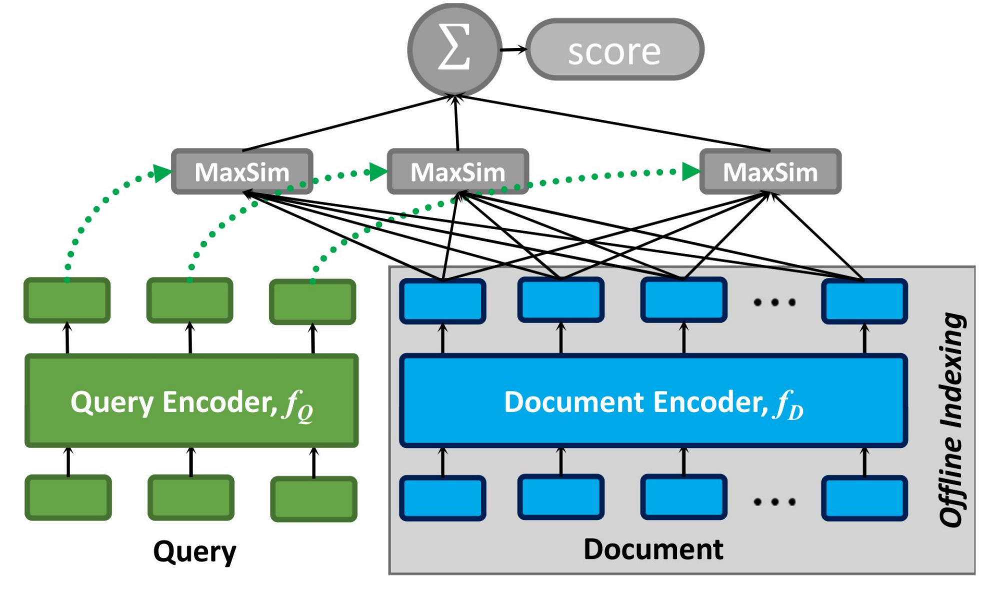


Figure 3: The general architecture of ColBERT given a query q and a document d.

Source: [ColBERT paper]

 |

Demo Time

BGE-M3

- **Multilingual**
  - Supports multiple natural languages
  - Cross-lingual in addition to multi-lingual
- **Multifunctional**
  - Supports both dense and sparse (splade-like) vectors
  - Late interaction model i.e. ColBERT
- **Multigranular**
  - Embeds short phrases as well as long documents
  - Up to 8192 token length

BGE-M3's Sparse Vectors

• **Lexical Retrieval.** The output embeddings are also used to estimate the importance of each term to facilitate lexical retrieval. For each term t within the query (a term is corresponding to a token in our work), the term weight is computed as $w_{q_t} \leftarrow \mathsf{Relu}(\mathbf{W}_{lex}^T \mathbf{H_q}[i]))$, where $\mathbf{W}_{lex} \in \mathcal{R}^{d\times1}$ is the matrix mapping the hidden state to a float number. If a term t appears multiple times in the query, we only retain its max weight. We use the same way to compute the weight of each term in the passage. Based on the estimation term weights, the relevance score between query and passage is computed by the joint importance of the co-existed terms (denoted as $q \cap p$) within the query and passage: $s_{lex} \leftarrow \sum_{t\in q\cap p}(w_{q_t} * w_{p_t})$.

Source: BGE-M3 paper

 |