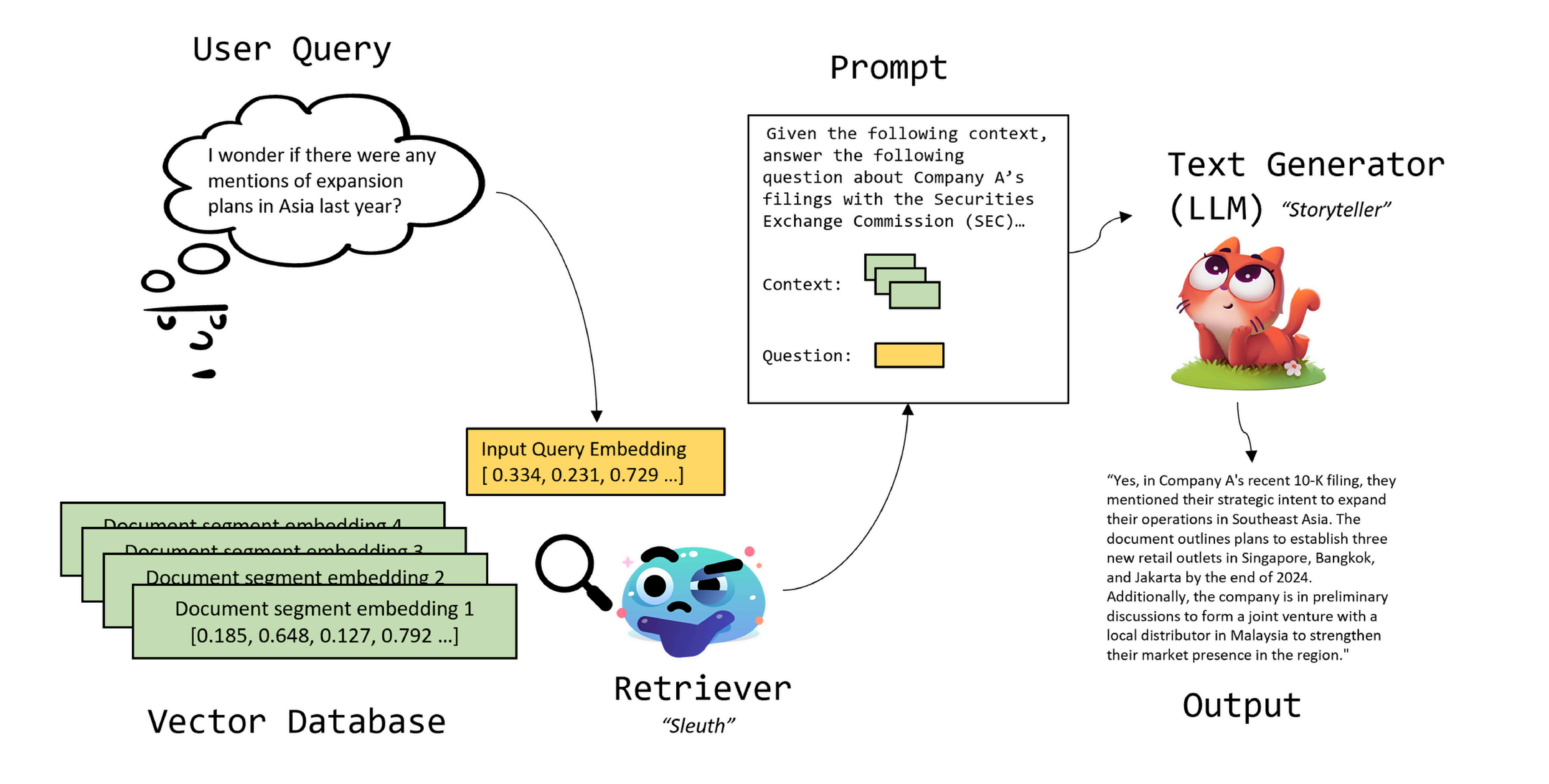
Enhancing Document Retrieval

FINE-TUNING TEXT EMBEDDING FOR RAG

Hieu Ngo

QUICK RECAP


User Query
I wonder if there were any mentions of expansion plans in Asia last year?
Input Query Embedding
[0.334, 0.231, 0.729 ...]
Document segment embedding 4
Document segment embedding 3
Document segment embedding 2
Document segment embedding 1
[0.185, 0.648, 0.127, 0.792 ...]
Vector Database
Retriever
"Sleuth"
Prompt
Given the following context, answer the following question about Company A's filings with the Securities Exchange Commission (SEC)…
Context:
Question:
Text Generator (LLM) "Storyteller"
"Yes, in Company A's recent 10-K filing, they mentioned their strategic intent to expand their operations in Southeast Asia. The document outlines plans to establish three new retail outlets in Singapore, Bangkok, and Jakarta by the end of 2024. Additionally, the company is in preliminary discussions to form a joint venture with a local distributor in Malaysia to strengthen their market presence in the region."
Output

CONCEPT

- Text Embedding
- Semantic Search
- BERT
- Sentence BERT
- Bi-encoder

TEXT EMBEDDING

An embedding is a vector (list) of numbers.

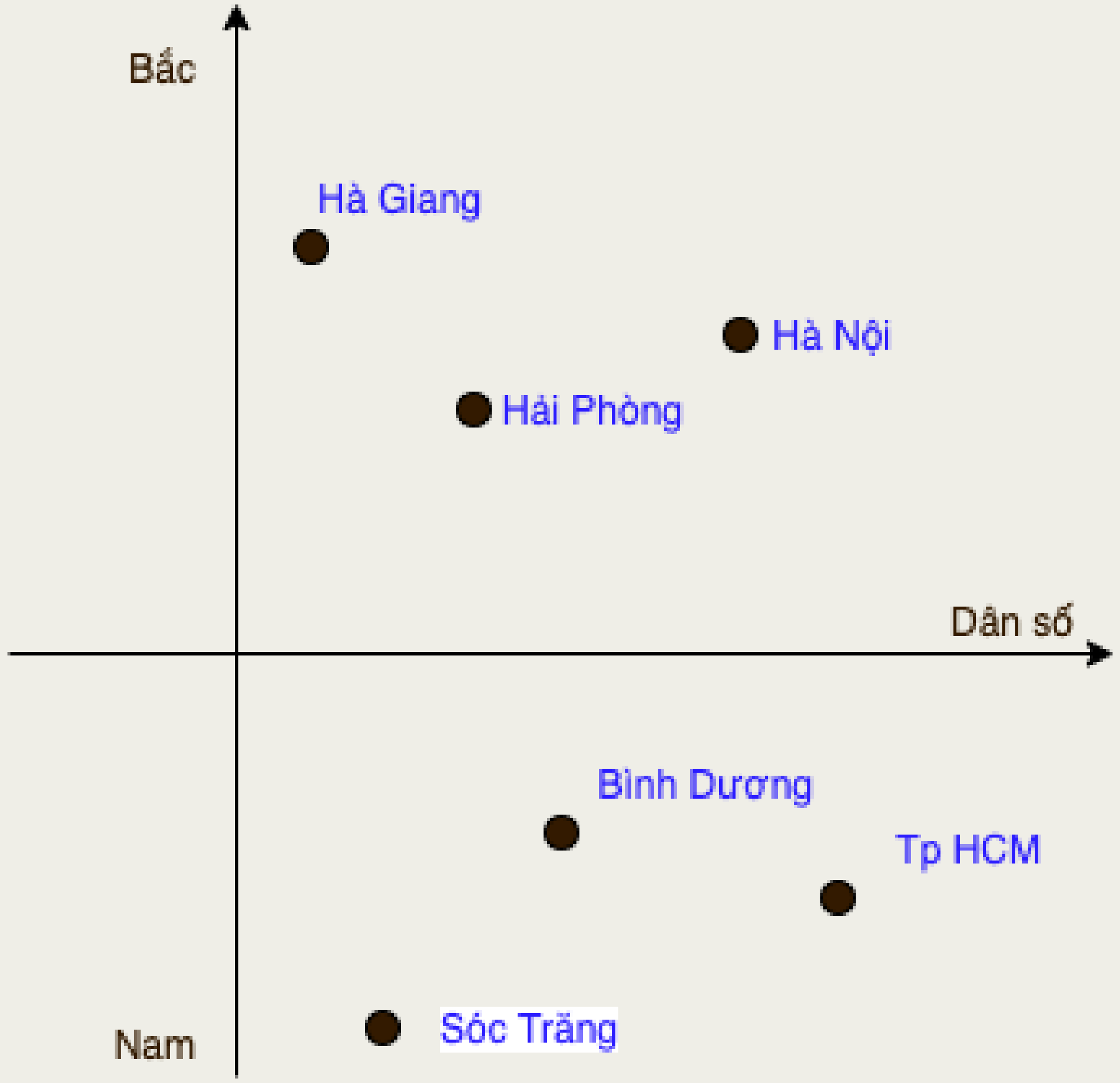
	Không gian one-hot						Không gian embedding	
Hà Nội	1	0	0	0	0	0	0.8	0.8
Hải Phòng	0	1	0	0	0	0	0.2	0.7
Tp HCM	0	0	1	0	0	0	0.9	-0.8
Bình Dương	0	0	0	1	0	0	0.25	-0.7
Hà Giang	0	0	0	0	1	0	0.08	1
Sóc Trăng	0	0	0	0	0	1	0.12	-0.9

TEXT EMBEDDING

Distance between two vectors measures their **relatedness.**

Small distances suggest **high relatedness**

Large distances suggest **low relatedness**



TEXT EMBEDDING USE CASE

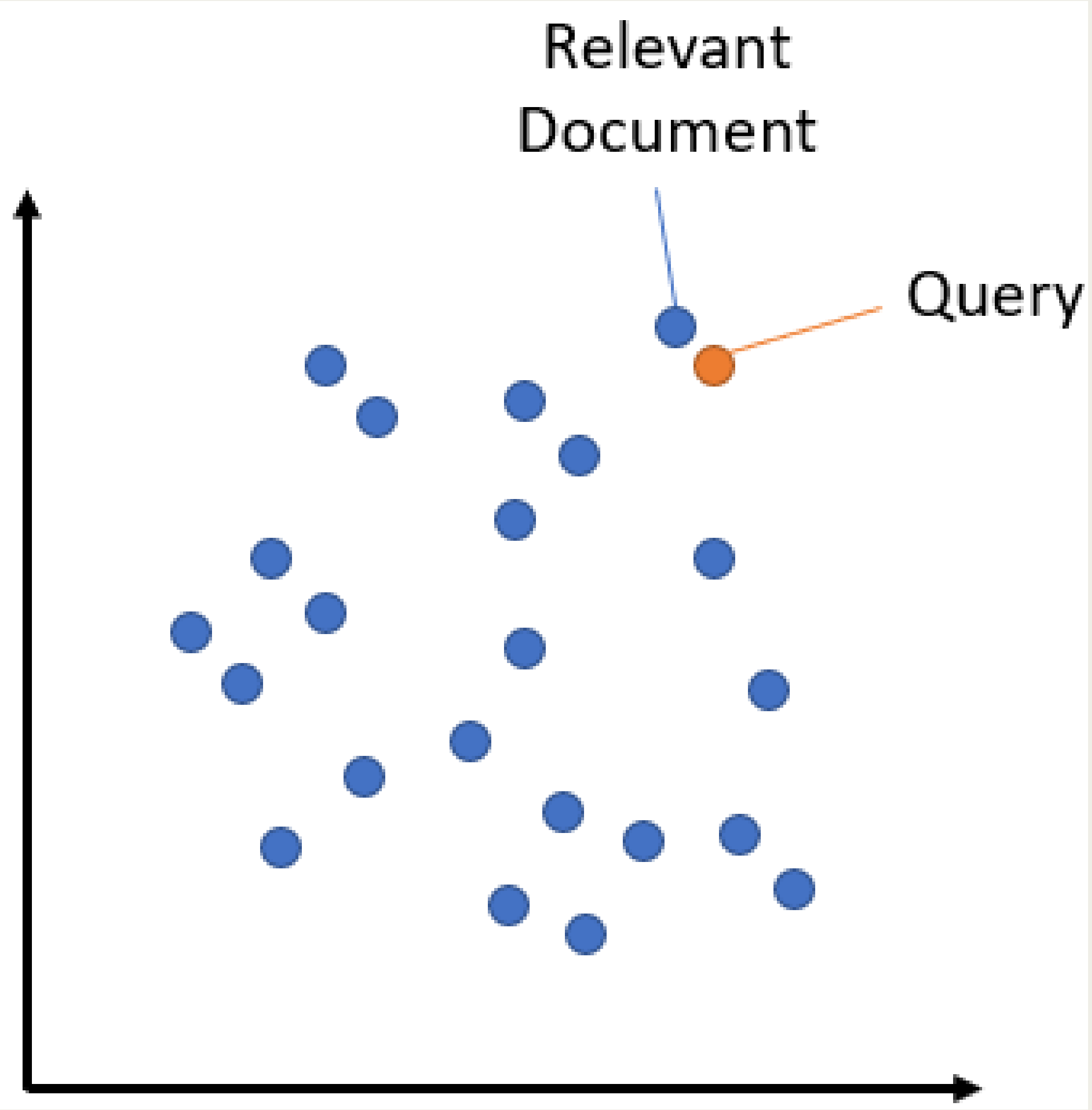
Turn text into numbers, unlocking use cases like **search**.

Embeddings are commonly used for:

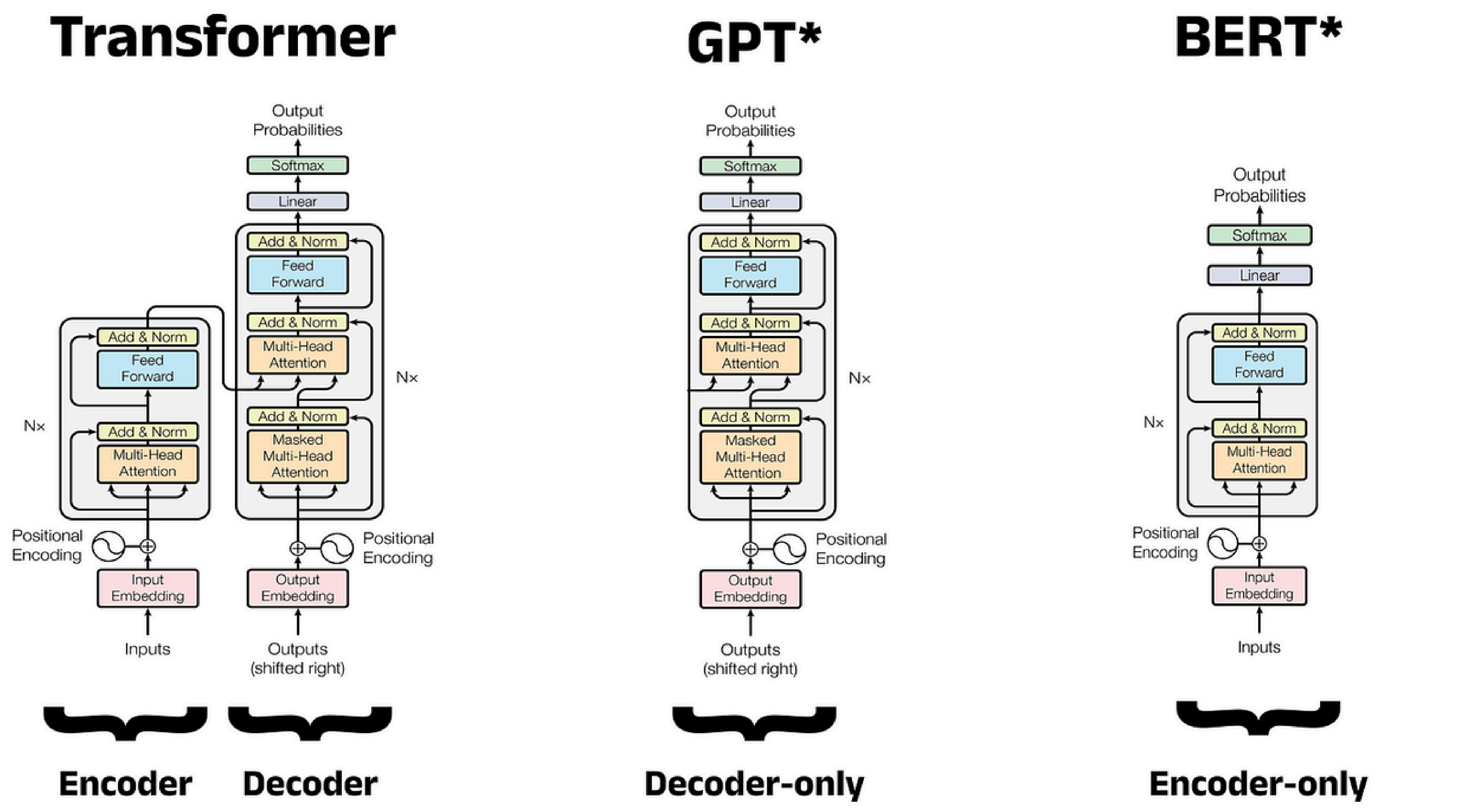
- **Semantic Search (where results are ranked by relevance to a query string) (our focus)**
- Clustering (where text strings are grouped by similarity)
- Recommendations (where items with related text strings are recommended)
- Anomaly detection (where outliers with little relatedness are identified)
- Diversity measurement (where similarity distributions are analyzed)
- Classification (where text strings are classified by their most similar label)

SEMANTIC SEARCH

- **Embeds Documents** into a **vector space**
- Does the same with the **query**
- Finds the **closest embeddings**, ensuring high **semantic similarity** between the **query** and **documents**.

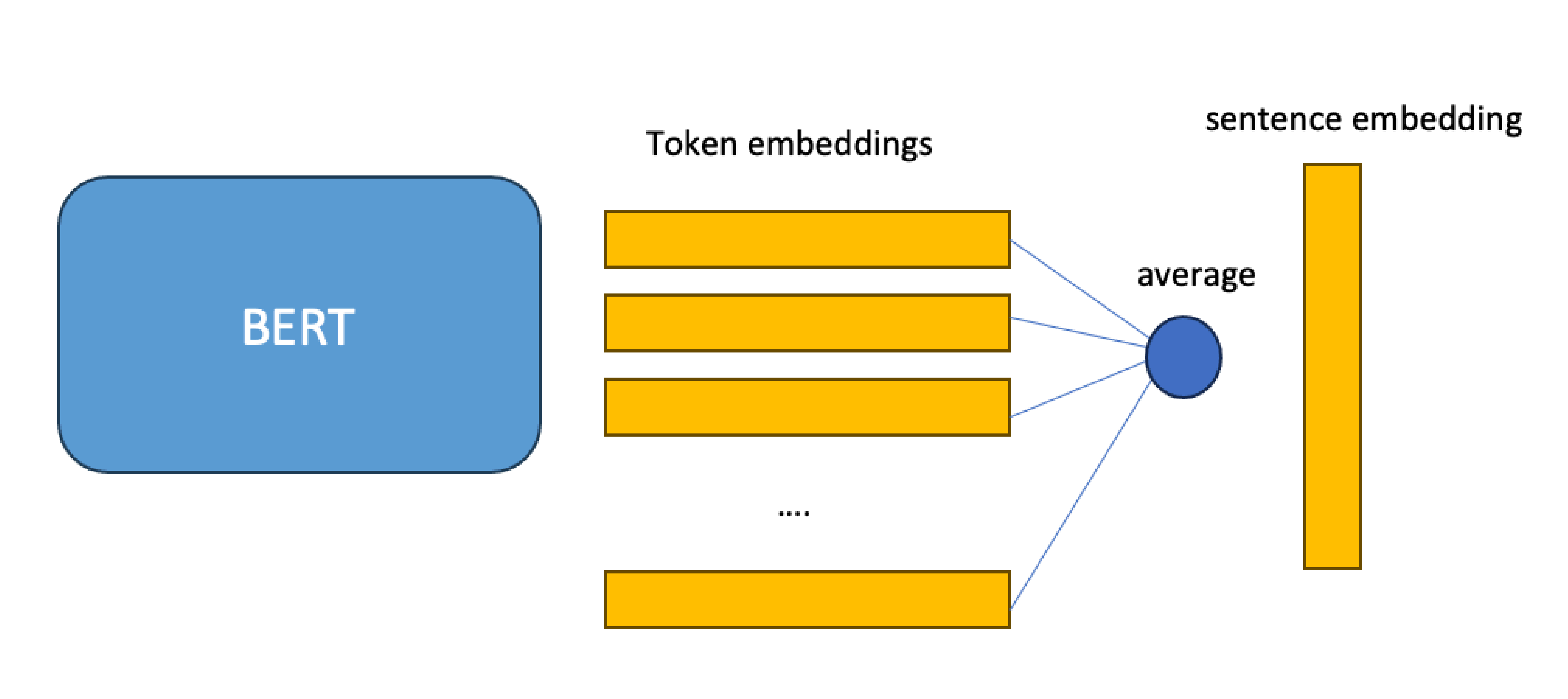


BIDIRECTIONAL ENCODER REPRESENTATIONS FROM TRANSFORMERS



*Illustrative example, exact model architecture may vary slightly

SENTENCE BERT (SBERT)

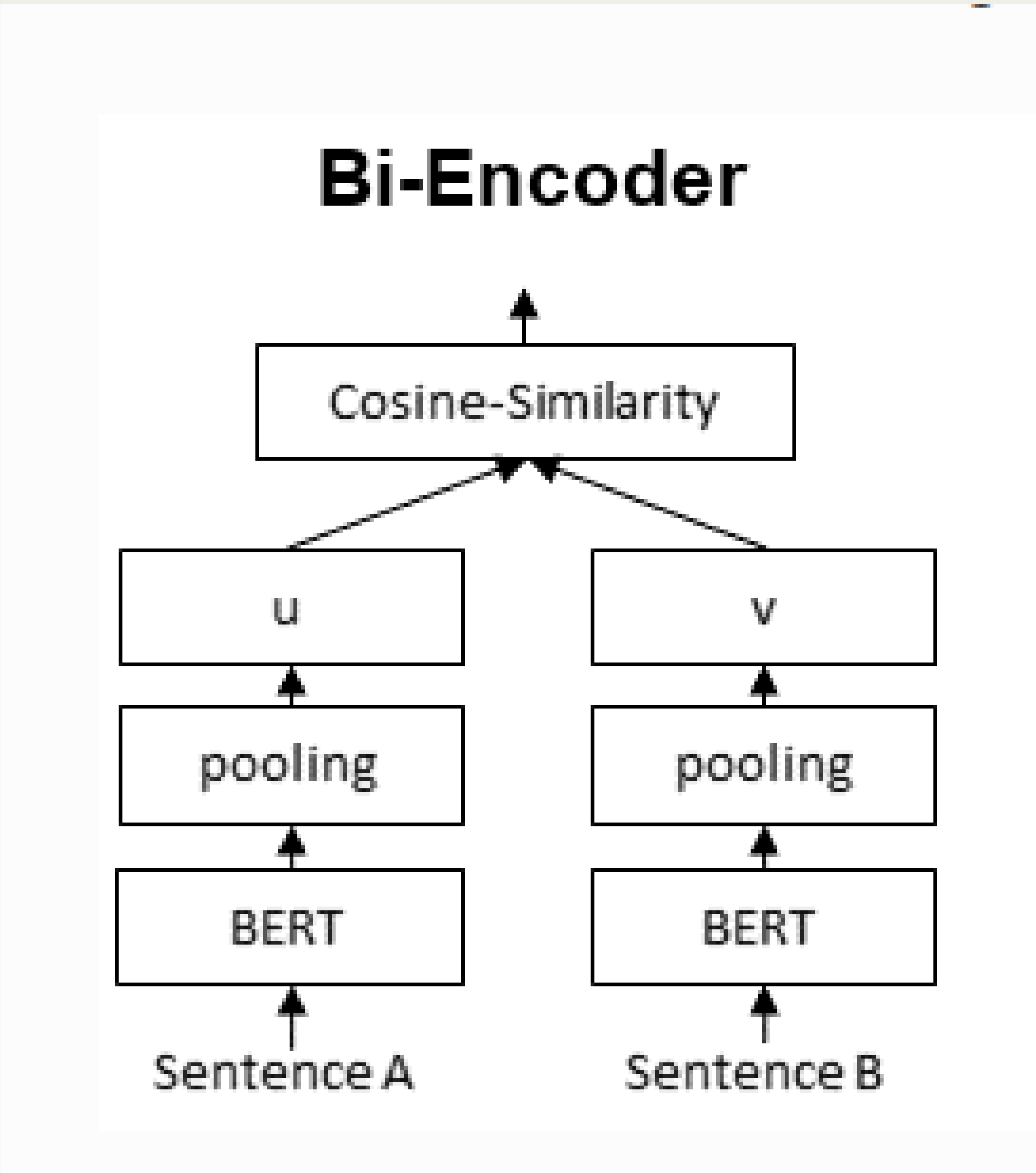


BI-ENCODER

Bi-Encoders produce for **a given sentence** a sentence **embedding**.

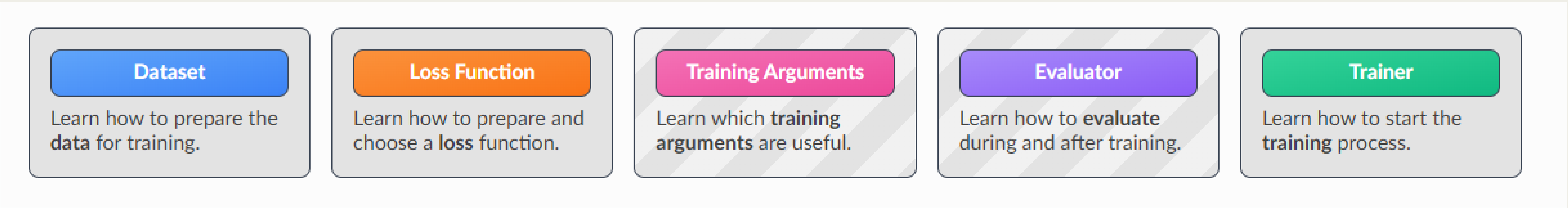
We pass to a BERT independently the sentences **A** and **B**, which result in the **sentence embeddings u** and **v**

These s**entence embedding** can then be **compared** using **cosine similarity**



FINE-TUNING

- Why Fine-tuning
- Dataset
- Loss Function
- Training Arguments
- Evaluator
- Trainer



WHY FINE-TUNING

Finetuning models often heavily **improves** the **performance** of the **model** on **your use case**, because each task requires a **different notion of similarity**.

For example, given news articles:

- “Apple launches the new iPad”
- “NVIDIA is gearing up for the next GPU generation”

Then the following use cases, we may have different **notions of similarity:**

- a model for **classification** of news articles as Economy, Sports, **Technology**, Politics, etc., should produce **similar embeddings** for these texts.
- a model for **semantic textual similarity** should produce **dissimilar embeddings** for these texts, as they have **different meanings.**
- a model for **semantic search** would **not need a notion for similarity** between two documents, as it should only compare **queries and documents.**

DATASET

Pair of Question and Document

The dataset has the following format

```
{"question": "<question>", "context": "<relevant context to answer>"}
{"question": "<question>", "context": "<relevant context to answer>"}
{"question": "<question>", "context": "<relevant context to answer>"}
```

LOSS FUNCTION

MultipleNegativesRankingLoss

$$\mathcal{L} = -\log\left(\frac{\exp(\text{sim}(\mathbf{a},\mathbf{p})/\tau)}{\sum_{i=1}^{N}\exp(\text{sim}(\mathbf{a},\mathbf{n}_i)/\tau)}\right)$$

Where:

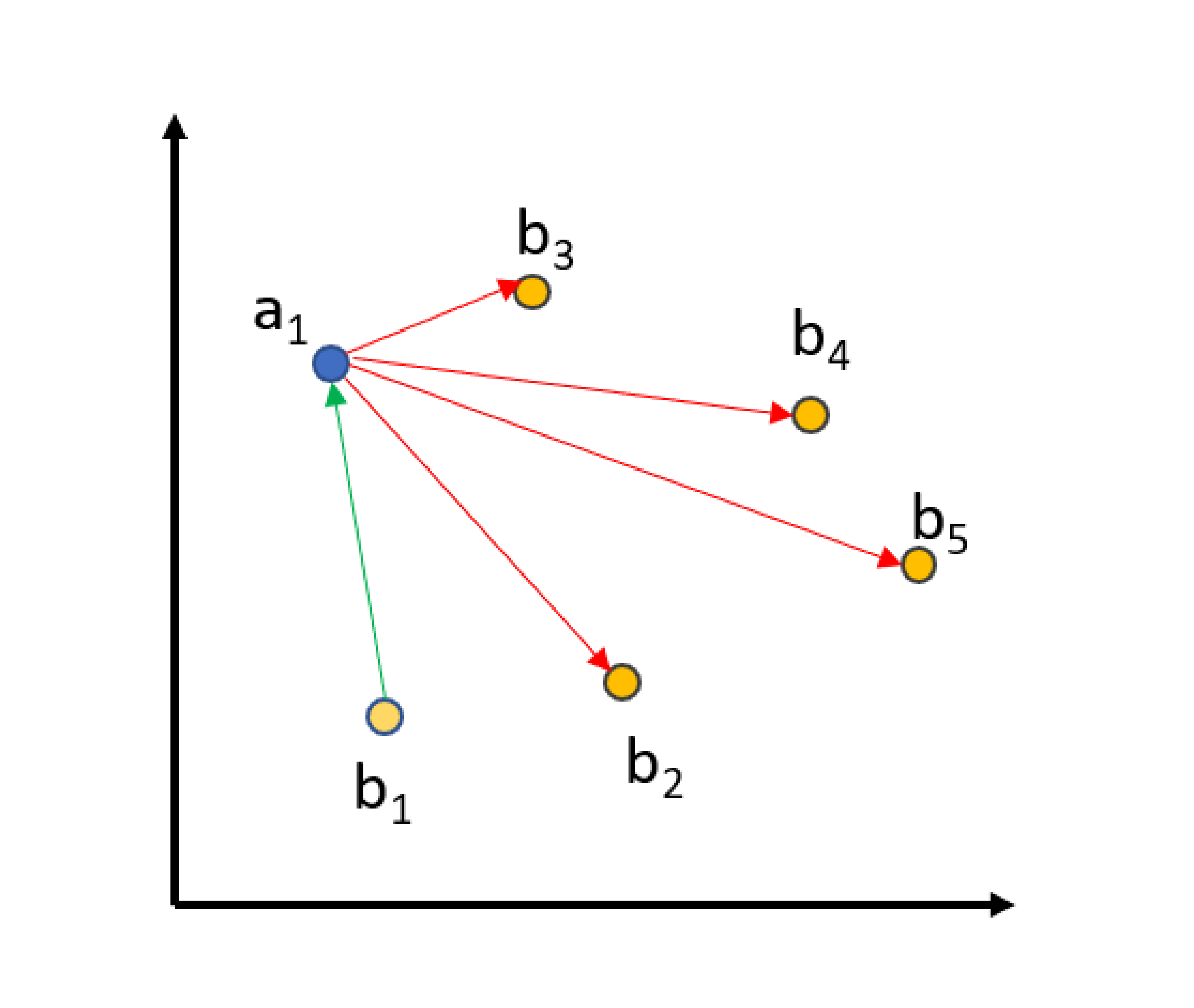
- $\mathbf{a}$ is the anchor (query) embedding.
- $\mathbf{p}$ is the positive (target) embedding.
- $\mathbf{n}_i$ are the negative embeddings.
- $\text{sim}(\mathbf{u},\mathbf{v})$ is the similarity function, often the dot product or cosine similarity between embeddings $\mathbf{u}$ and $\mathbf{v}$.
- τ is a temperature parameter that controls the scaling of the similarities.
- N is the number of negative samples.

MultipleNegativesRankingLoss

pairs [(a_1, b_1), …, (a_n, b_n)]

- (a_i, b_i) are **similar sentences**
- (a_i, b_j) are **dissimilar sentences**

The **distance** between **(a_1, b_1)** is **reduced**
The **distance** between **(a_1, $b_{2…5}$)** will be **increased**



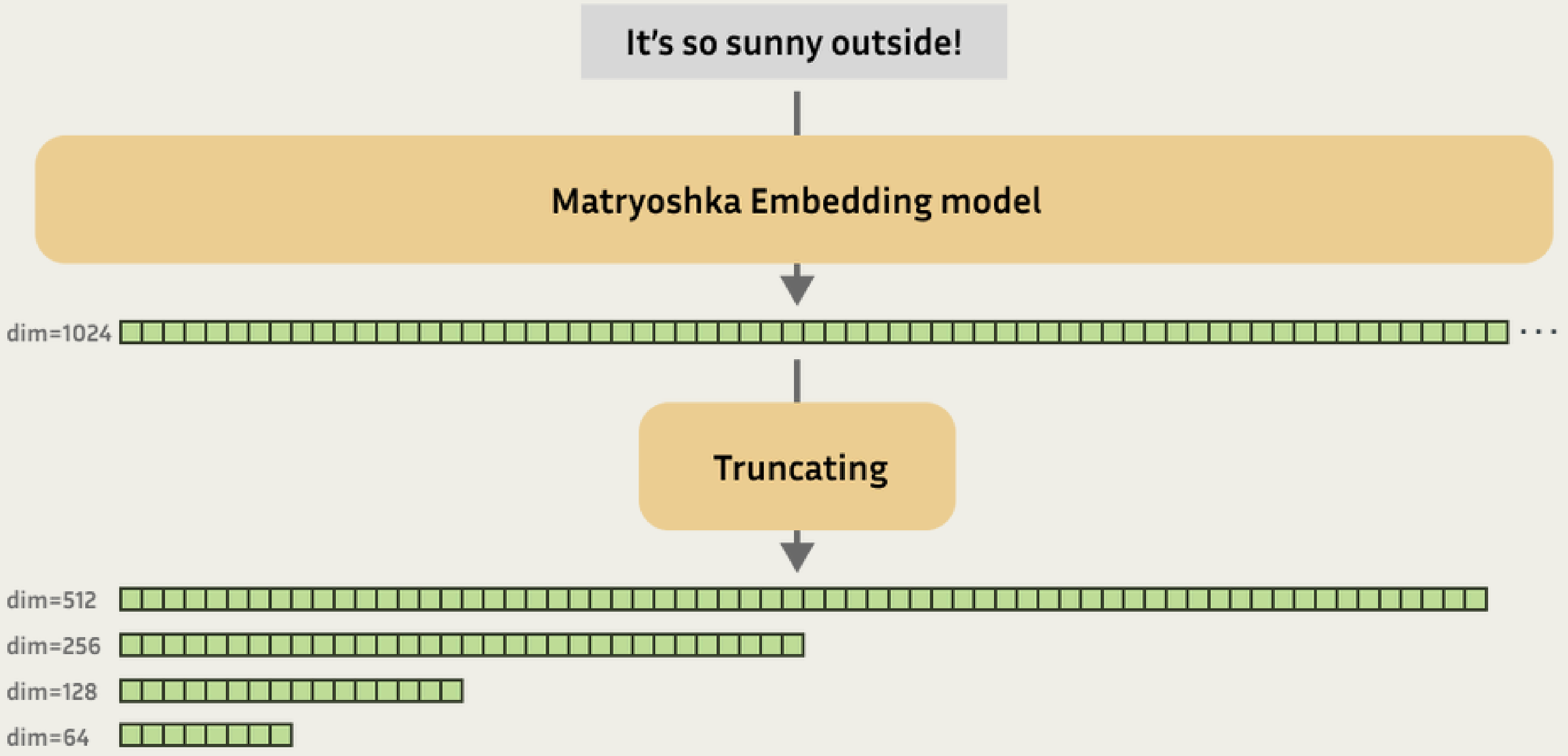
Matryoshka loss

Matryoshka loss function to determine not just the **quality** of your **full-size embeddings**, **but** also the **quality** at various **different embeddings**

For example, output dimensionalities are **768, 512, 256, 128, and 64**.

The **loss values** for each dimensionality are **added together**, resulting in a **final loss value**

The optimizer will then try and adjust the model weights to lower this loss value.



INFORMATION RETRIEVAL EVALUATOR

Example Scenario

- Total documents in dataset: **20**
- Total relevant documents in dataset: **10**
- Documents retrieved by system: **5**
- Relevant documents retrieved by system: **3**
- Example Query: has **10** relevant documents in the dataset.

Example Top K document retrieved

- @1: **[Relevant]**
- @3: **[Relevant, Irrelevant, Relevant]**
- @5: **[Relevant, Irrelevant, Relevant, Irrelevant, Irrelevant]**

Metric	Definition	@1 Calculation	@1 Result	@3 Calculation	@3 Result	@5 Calculation	@5 Result
Accuracy	Measures if at least one relevant document is in the top k.	(Relevant doc in top-1?) / (Total queries)	1 / 1 = 100%	(Relevant doc in top-3?) / (Total queries)	1 / 1 = 100%	(Relevant doc in top-5?) / (Total queries)	1 / 1 = 100%
Precision	Measures the proportion of relevant documents in the top k.	(Relevant docs in top-1) / 1	1 / 1 = 100%	(Relevant docs in top-3) / 3	2 / 3 = 66.67%	(Relevant docs in top-5) / 5	2 / 5 = 40%
Recall	Measures the proportion of relevant documents retrieved out of the total relevant documents.	(Relevant docs in top-1) / 10	1 / 10 = 10%	(Relevant docs in top-3) / 10	2 / 10 = 20%	(Relevant docs in top-5) / 10	2 / 10 = 20%

CONSIDERATIONS

- **Better context length (8192)** -> Nomic Embed
- **LLM Embedding** -> (Salesforce/SFR-Embedding-Mistral, https://huggingface.co/Alibaba-NLP/gte-Qwen1.5-7B-instruct)
- **ColBERT Model** (contextual late interaction) -> jinaai/jina-colbert-v1-en

MY TIPS

- OpenAI Embedding is a good start
- Dataset matters the most (as always)
- Fine-tuned model on proprietary data always outperform open/general model
- Most of mine time improving RAG is to improve internal search engine (hybrid search, retrieval + rerank,...)

Thank you!