RAG Data Ingestion in Apache Beam

Jasper Van den Bossche & Konstantin Buschmeier

BEAM SUMMIT

September 4-5, 2024

Sunnyvale, CA. USA

ML6

Empowering Businesses with AI & Machine Learning

Specialized in:

- Strategic Guidance AI Adoption and AI governance
- End-to-end ML/AI application development

BEAM SUMMIT

What will be covered in this talk?

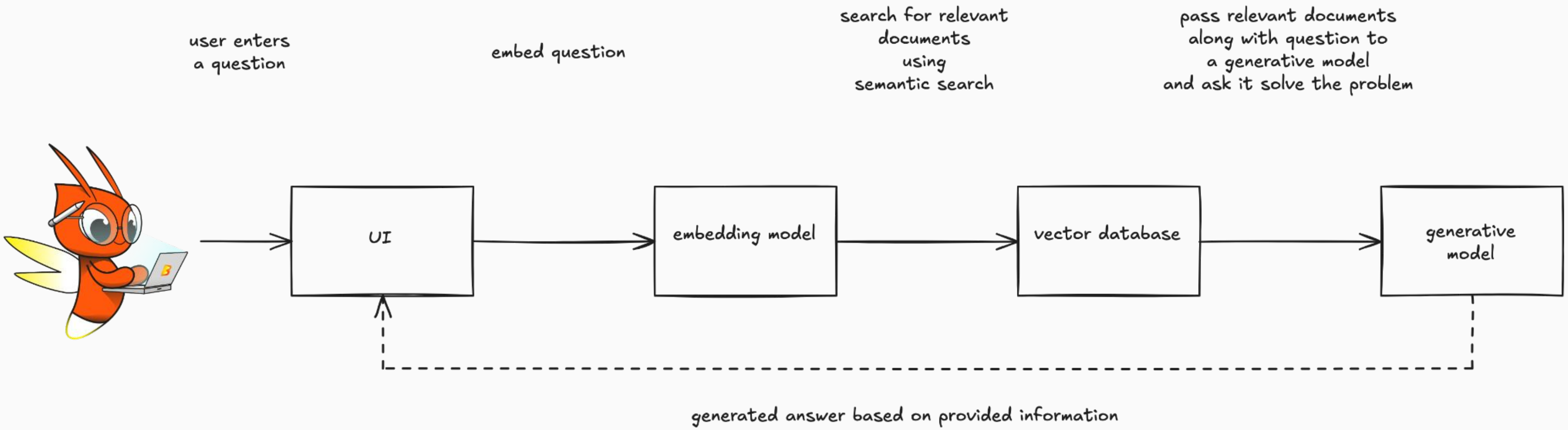
An Introduction to RAG

Building a RAG Ingestion Pipeline

Remarks

A Gentle Introduction to RAG

Retrieval Augmented Generation



Lexical Search

What is Apache Beam?

$\boxed{\text{Apache}}$ $\boxed{\text{Beam}}$ is a unified model for defining both batch and streaming data-parallel processing pipelines

$\boxed{\text{Apache}}$ Flink is an open source stream processing framework with powerful stream- and batch-processing capabilities.

cuDF (pronounced "KOO-dee-eff") is a GPU DataFrame library for loading, joining, aggregating, filtering, and otherwise manipulating data.

BEAM SUMMIT

Text Embeddings

Apache Beam is a unified model for defining both batch and streaming data-parallel processing pipelines

$$\longrightarrow \begin{bmatrix} 0.1279 \\ -0.8841 \\ 0.7683 \\ \cdots \\ 0.7724 \\ -0.4621 \\ -0.3327 \end{bmatrix}$$

BEAM SUMMIT

Transform Documents in a Collection of Embedding Vectors

I'm a fan of hip-hop music.

Al pastor tacos always hit the spot.

I enjoy listening to hip-hop.

Al pastor tacos are one of my favorite dishes.

Hip-hop is one of my favorite genres.

Al pastor tacos are super tasty.

Cycling is one of my favorite weekend activities.

I love riding my bike during weekends.

I really enjoy bike rides during the weekends.

BEAM SUMMIT

Nearest Neighbors Similarity Search

Store Embeddings in a Vector Database

original document	summary	ingestion timestamp	embedding
<doc 1>	<summarized doc 1>	24/08/2024 17:18	[0.267, 0.312, ... 0.972]
<doc 2>	<summarized doc 2>	24/08/2024 17:21	[0.358, 0.614, ... 0.587]
...	...	...	...
<doc N>	<summarized doc N>	29/08/2024 09:54	[0.111, 0.227, ... 0.379]

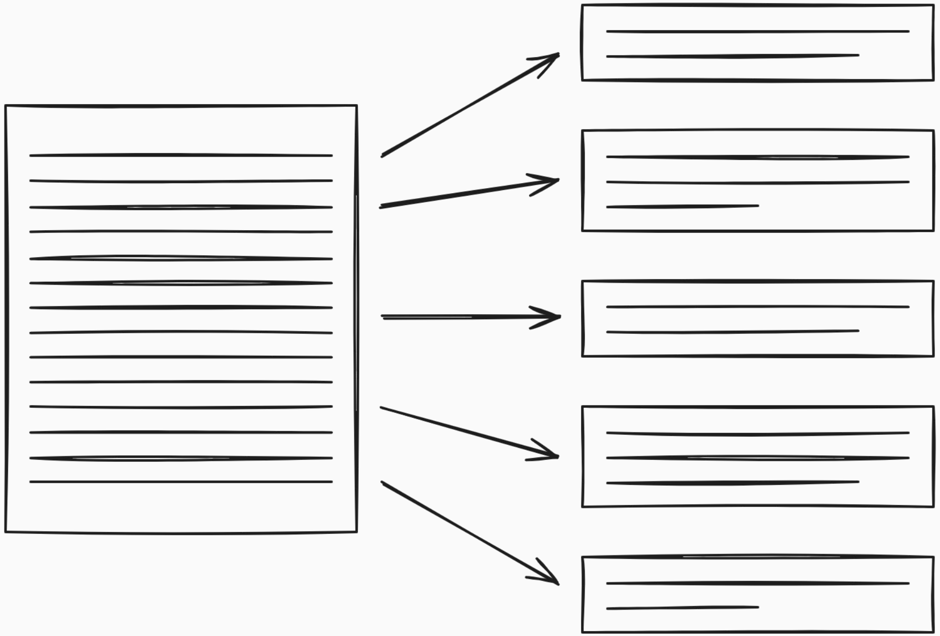
What makes a good embedding?

important context

What services does $\boxed{\text{ML6}}$ offer?

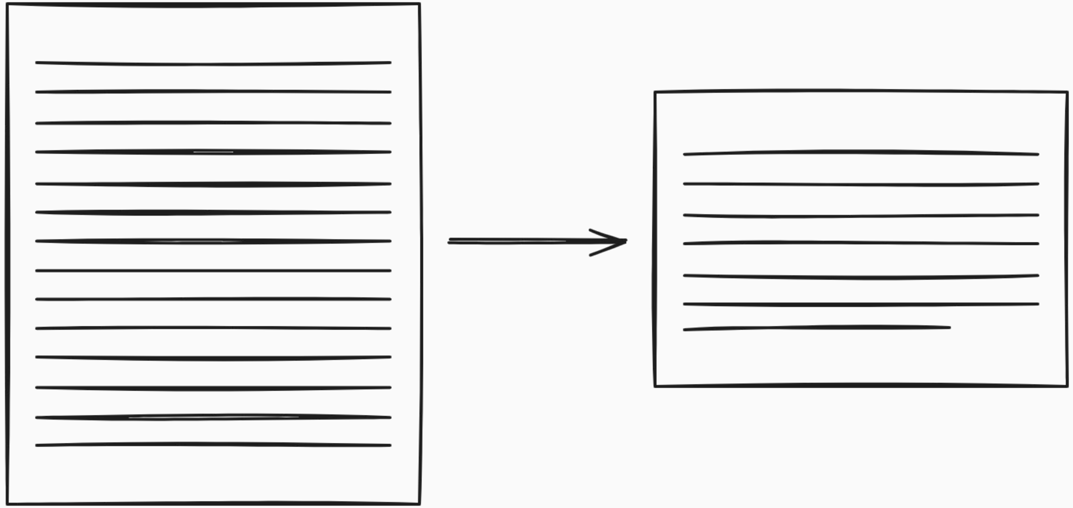
We're offering advice on how to effectively build AI solutions within your organization, implementing an a AI governance strategy.
Besides advice we also build custom end to end AI solutions.

Preprocessing: Chunking



Chunking

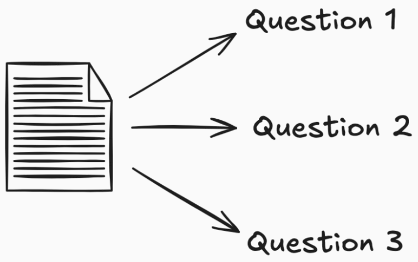
Preprocessing: Summarization



Summarization

Advanced Techniques to Improve Retrieval

retrieved documents	relevance score
doc 14	0.97
doc 87	0.91
...	...
doc 2	0.54



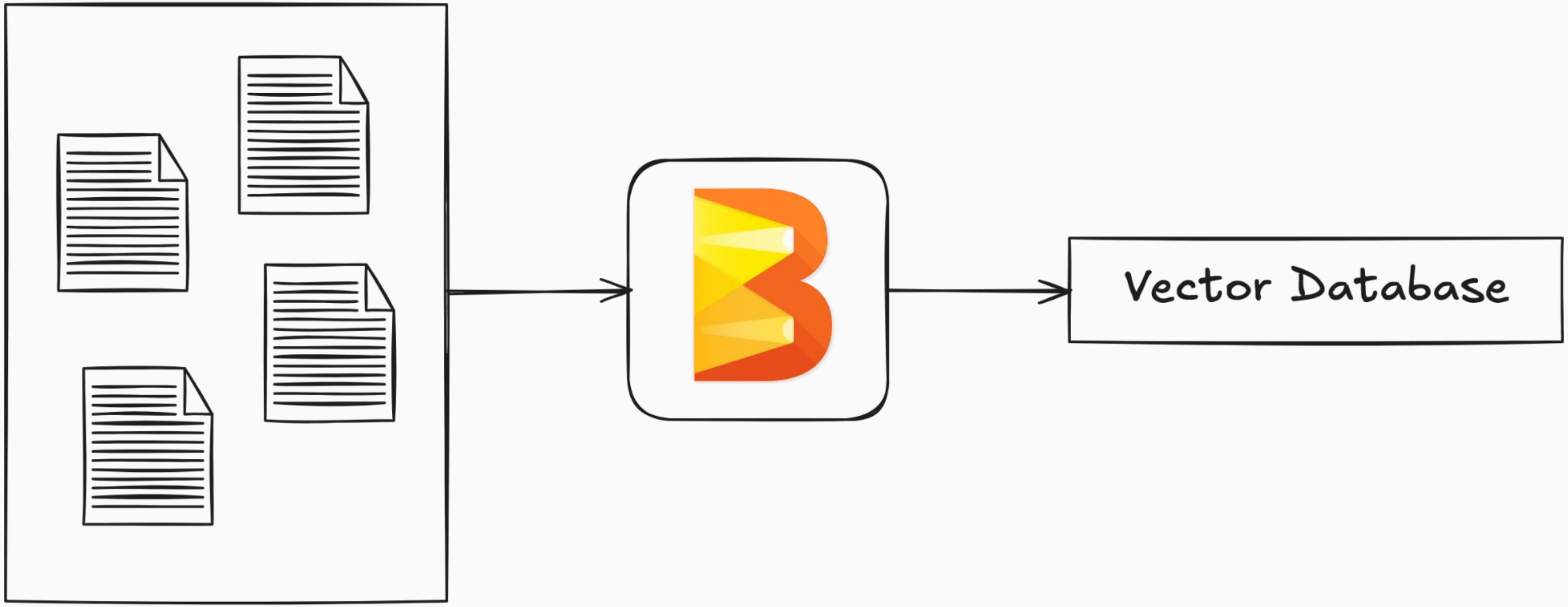
Generate Hypothetical Questions from Documents

Building a RAG Ingestion Pipeline

Why Apache Beam?

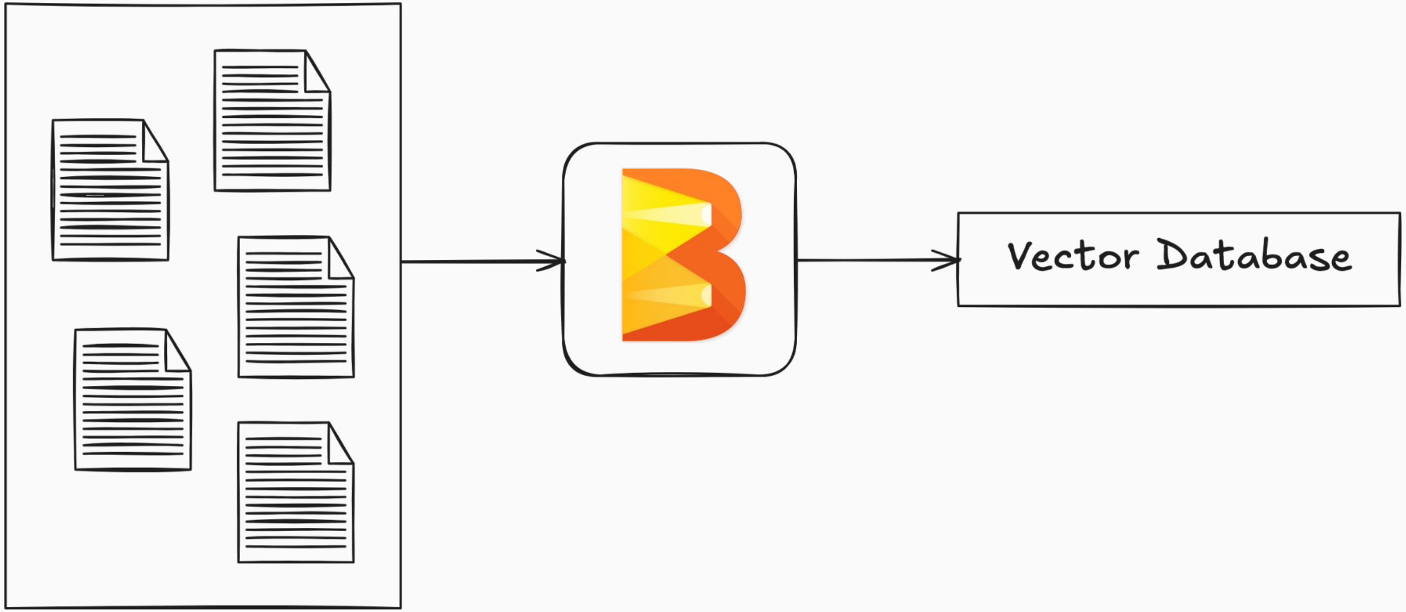
BEAM SUMMIT

Batch Processing



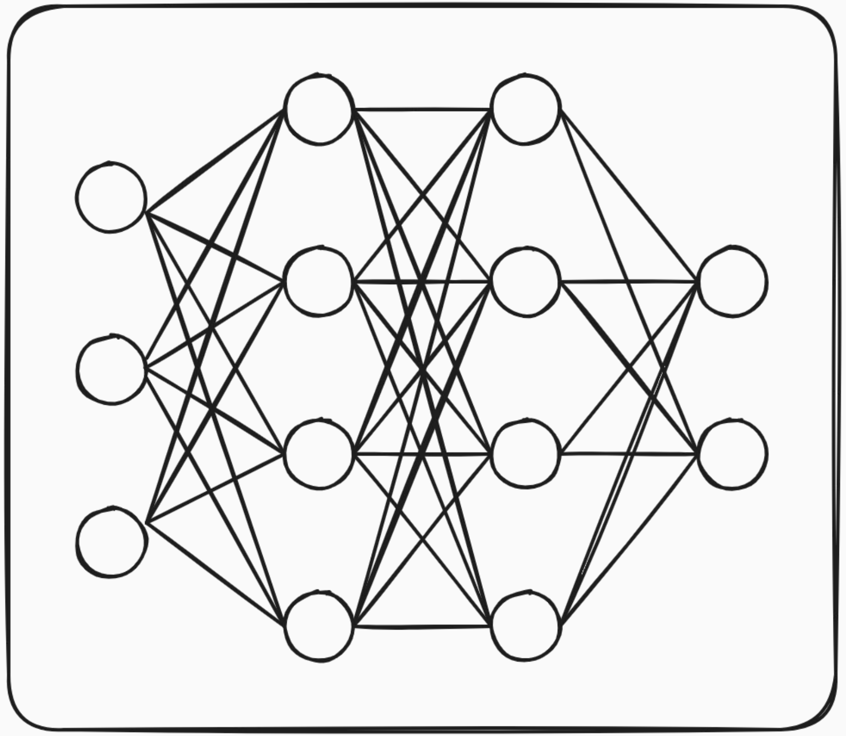
Batch Processing Existing Knowledge Base

Stream Processing



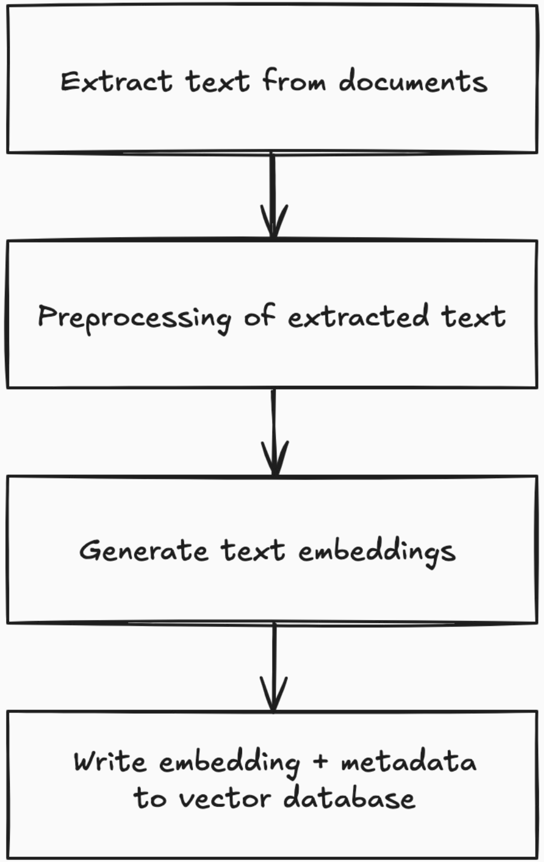
Stream Processing for Updates in Knowledge Base

Machine Learning Baked In



RunInference

High-Level Overview of RAG Ingestion Pipeline



Step 1: Text Extraction

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

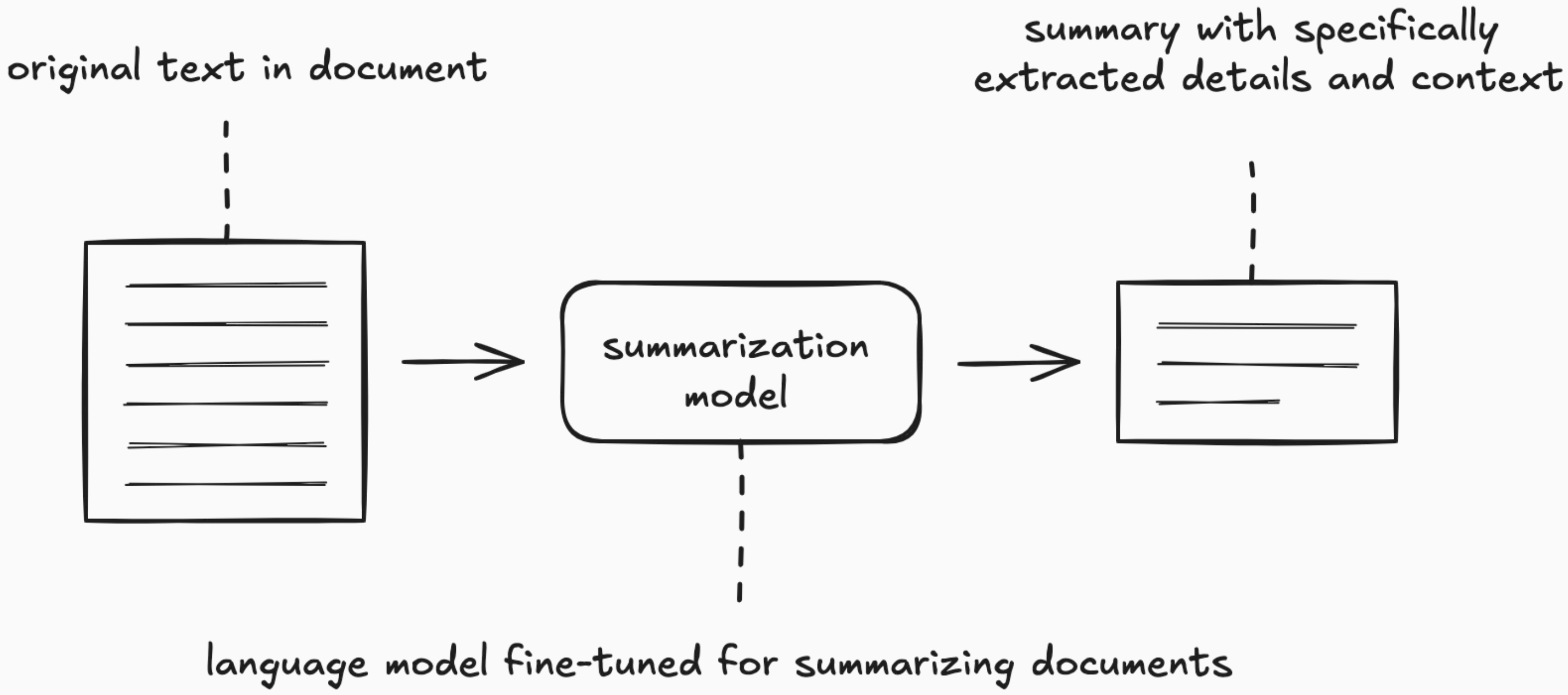
Attention Is All You Need

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 Englishto-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

BEAM SUMMIT

```
class ExtractTextFromPDF(beam.DoFn):
    def process(self, element):
        try:
            with fitz.open(element) as doc:
                text = '\n'.join([page.get_text() for page in 
doc])
                yield text
```

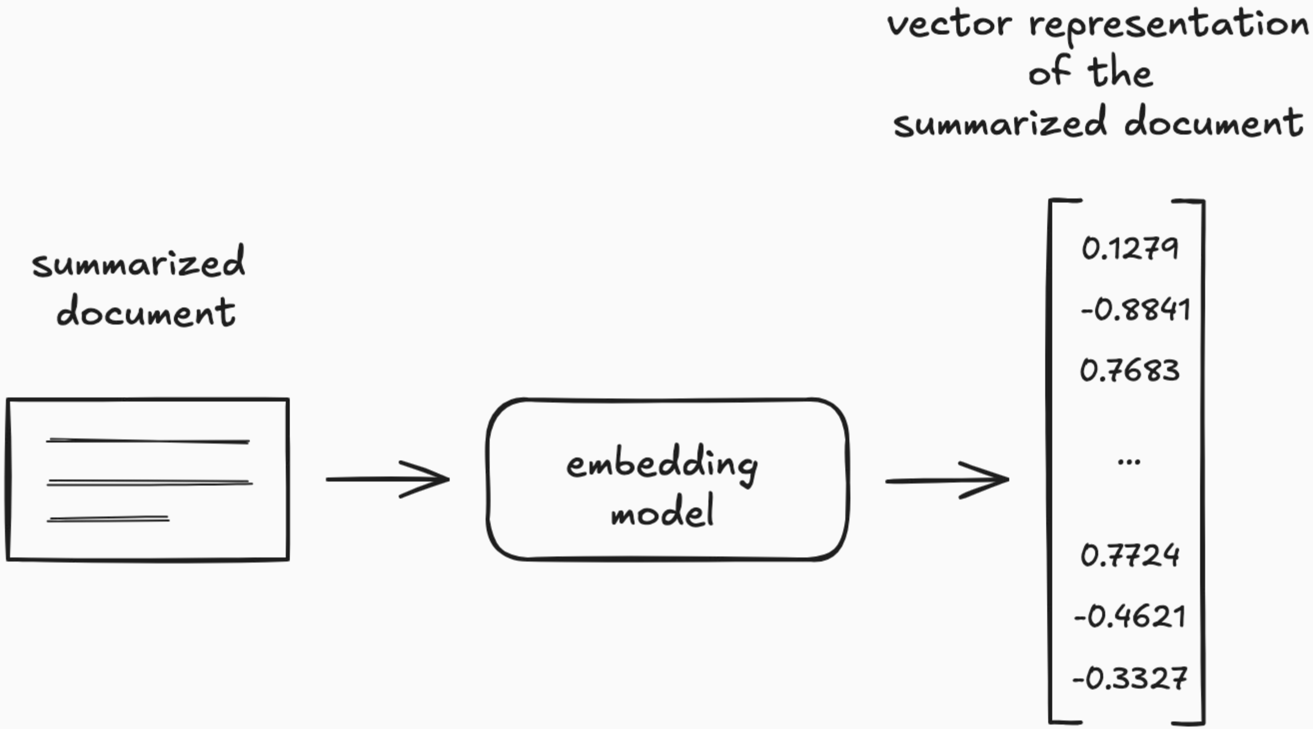
Step 2: Preprocessing Text Data for Embedding



```python
class CohereSummarizationModelHandler(CohereModelHandler):
    ...
    def run_inference(self, batch: Sequence[Dict[str, Any]], model: cohere.Client,
                      inference_args: Optional[Dict[str, Any]] = None) -> Iterable[PredictionResult]:
        inference_args = {} if not inference_args else inference_args
        summaries = []
        for element in batch:
            text = element['text']
            # The Cohere summarization models do not support inputs smaller than 250 character
            if len(text) > 250:
                # Send the text to the summarization model, along with optionally configured parameters
                summary_response = model.summarize(text=text, **self.config)
                # Extract the summary from the response returned by the API
                summary = summary_response.summary
            else:
                # Texts shorter than 250 characters aren't summarized
                summary = text
            summaries.append(summary)

        # Add the summaries to the output dictionaries along the other blogs
        updated_list_of_dicts = [{**element, 'summary': summary} for element, summary in zip(batch,
summaries)]
        # Return the output dictionaries as a batch of PredictionResult objects
        return [PredictionResult(x, y) for x, y in zip(batch, updated_list_of_dicts)]
```

Step 3: Embedding the Summarized Documents



```python
class CohereEmbeddingModelHandler(CohereModelHandler):
    ...

    def run_inference(self, batch: Sequence[Dict[str, Any]], model: cohere.Client,
                      inference_args: Optional[Dict[str, Any]] = None) ->
Iterable[PredictionResult]:
        inference_args = {} if not inference_args else inference_args
        # Create a list of inputs that will be sent to the embedding model
        texts = [element[self.input_key] for element in batch]

        # Send the text to the embedding model, along with optionally configured parameters
        response = model.embed(texts=texts, **self.config)

        # Extract the embeddings from the response returned by the API
        embeddings = response.embeddings

        # Return a list of PredictionResult
        return [
            PredictionResult(example=element, inference=embedding)
            for element, embedding
            in zip(batch, embeddings)
        ]
```

Step 4: Writing to a Vector Database

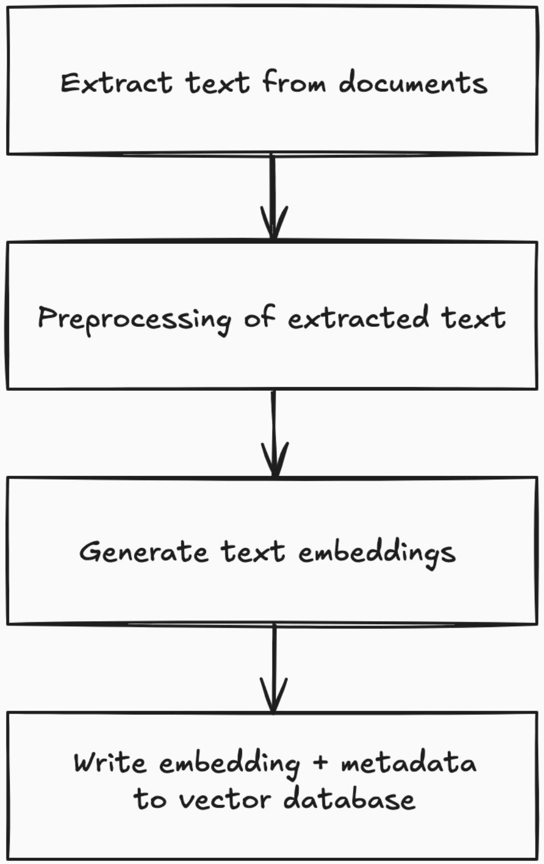
original document	summary	ingestion timestamp	embedding
<doc 1>	<summarized doc 1>	24/08/2024 17:18	[0.267, 0.312, ... 0.972]
<doc 2>	<summarized doc 2>	24/08/2024 17:21	[0.358, 0.614, ... 0.587]
...	...	...	...
<doc N>	<summarized doc N>	29/08/2024 09:54	[0.111, 0.227, ... 0.379]

BEAM SUMMIT

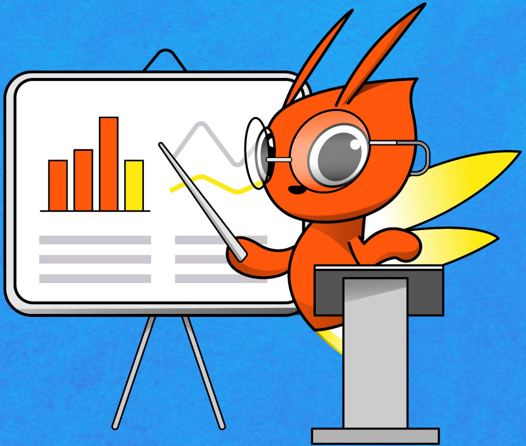
```python
class StoreWeaviate(beam.DoFn):
    ...

    def process(self, prediction_results: List[PredictionResult],
**kwargs):
        collection = self.weaviate_client.collections.get(self.collection)

        with collection.batch.dynamic() as batch:
            for prediction_result in prediction_results:
                batch.add_object(
                    properties=prediction_result.example,
                    vector=prediction_result.inference
                )
```


Extract text from documents
Preprocessing of extracted text
Generate text embeddings
Write embedding + metadata to vector database

Things to Consider



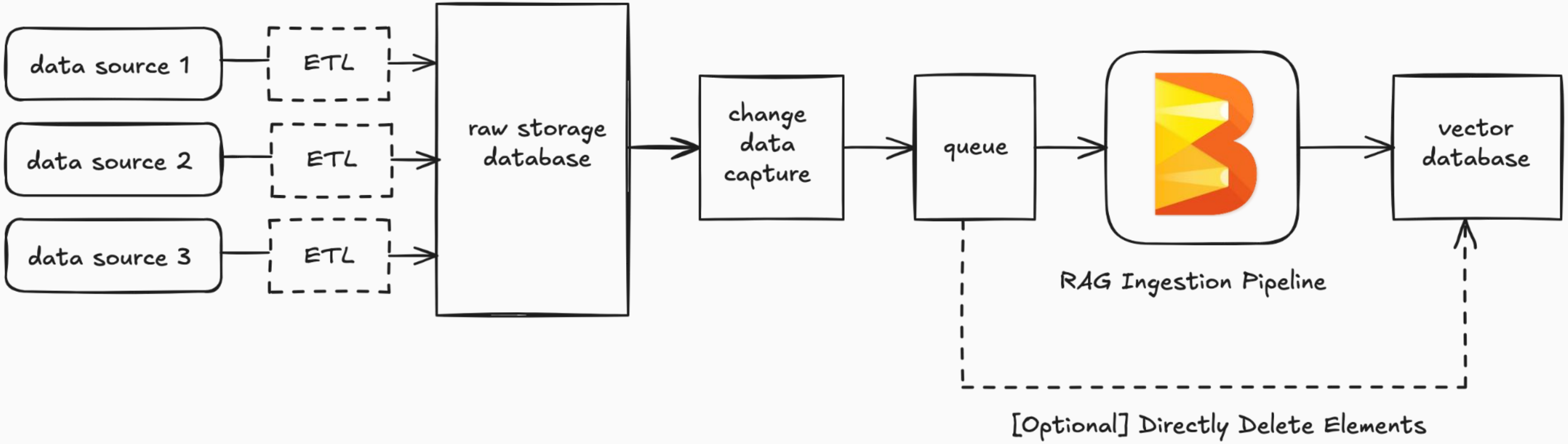
Demand for LLMs

→

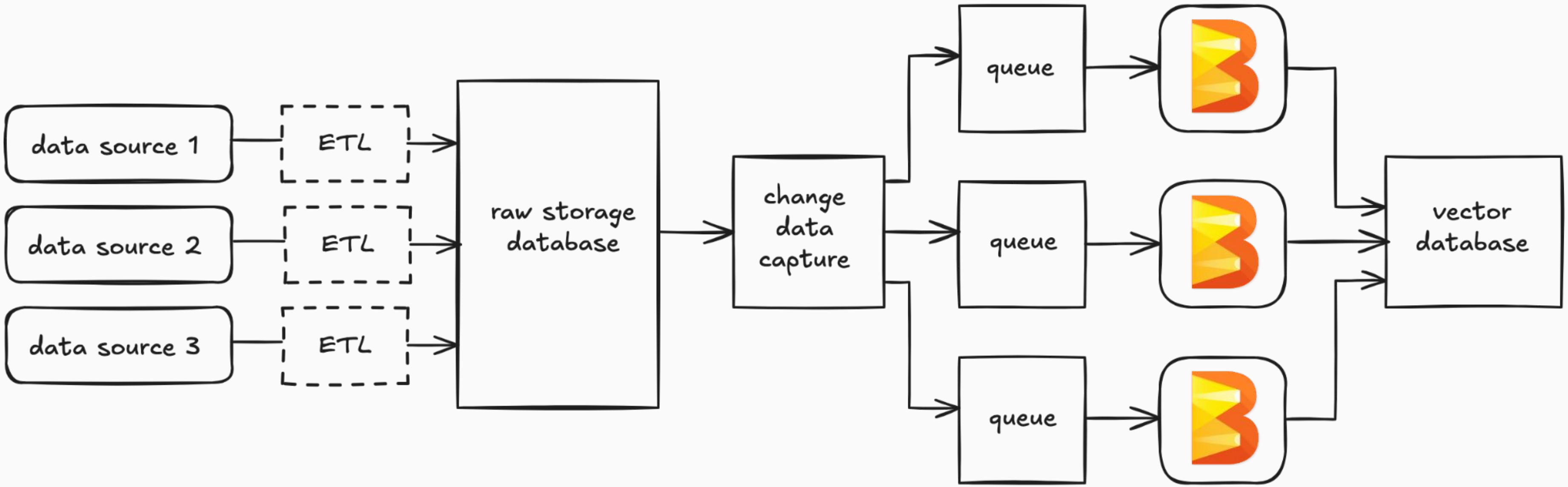
API Quota and Rate Limits
+
Availability of LLM APIs

Bonus: Using the RAG Ingestion Pipeline in Production

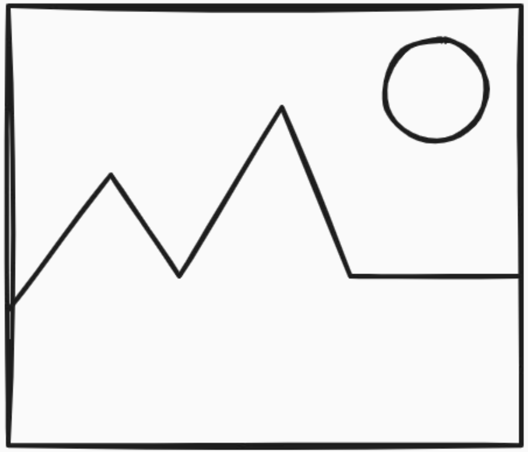
Real Time Data Ingestion: Change Data Capture Architecture



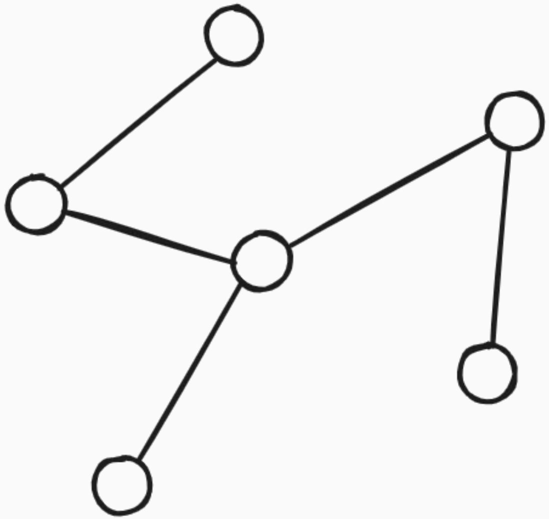
Real Time Data Ingestion: Change Data Capture Architecture



Future of RAG

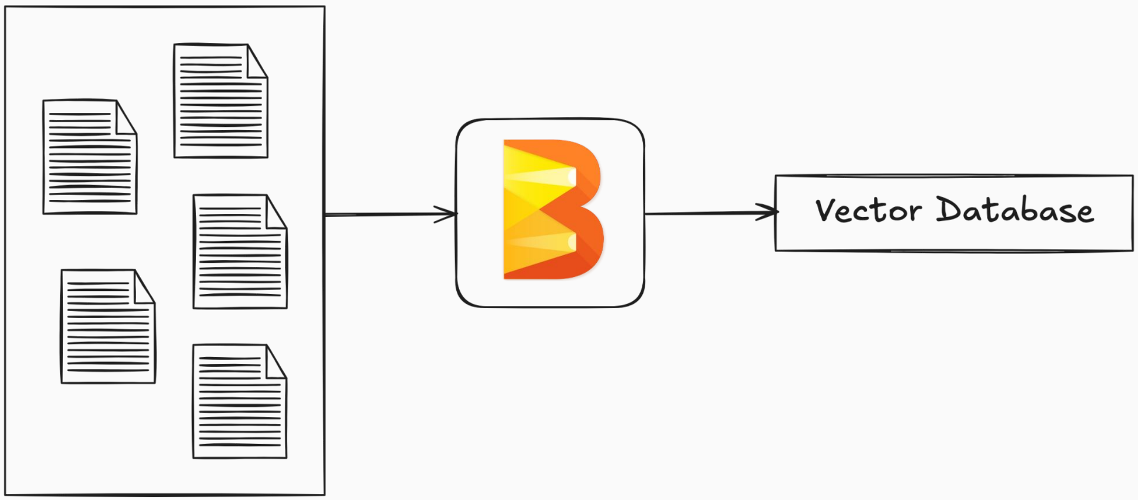


MultiModal RAG: RAG using Images, Videos, ...



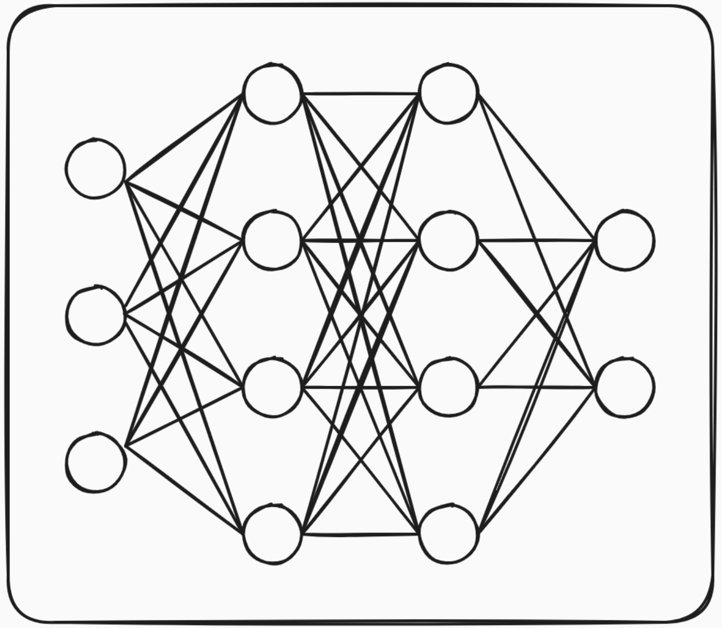
Graph RAG: RAG using Knowledge Graphs

Conclusion

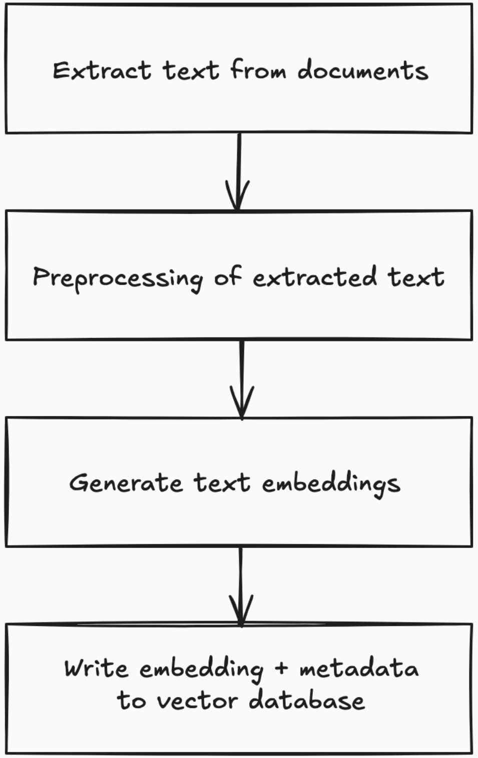


Batch Processing for Ingesting Exesting Datasets to Knowledge Base
+
Stream Processing for Updates in Knowledge Base

BEAM SUMMIT



RunInference SDK



Flexibility to Create Different Transforms

Thank you!

Questions?

https://www.linkedin.com/in/jasper-van-den-bossche/

https://www.linkedin.com/in/konstantin-buschmeier/

https://www.ml6.eu/