DeepLearning.AI

Module 3 introduction

Information Retrieval in Production

Introduction

Vector Databases

- Vector databases: optimized for huge quantities of vector data
- Almost synonymous with RAG systems

What You'll Learn

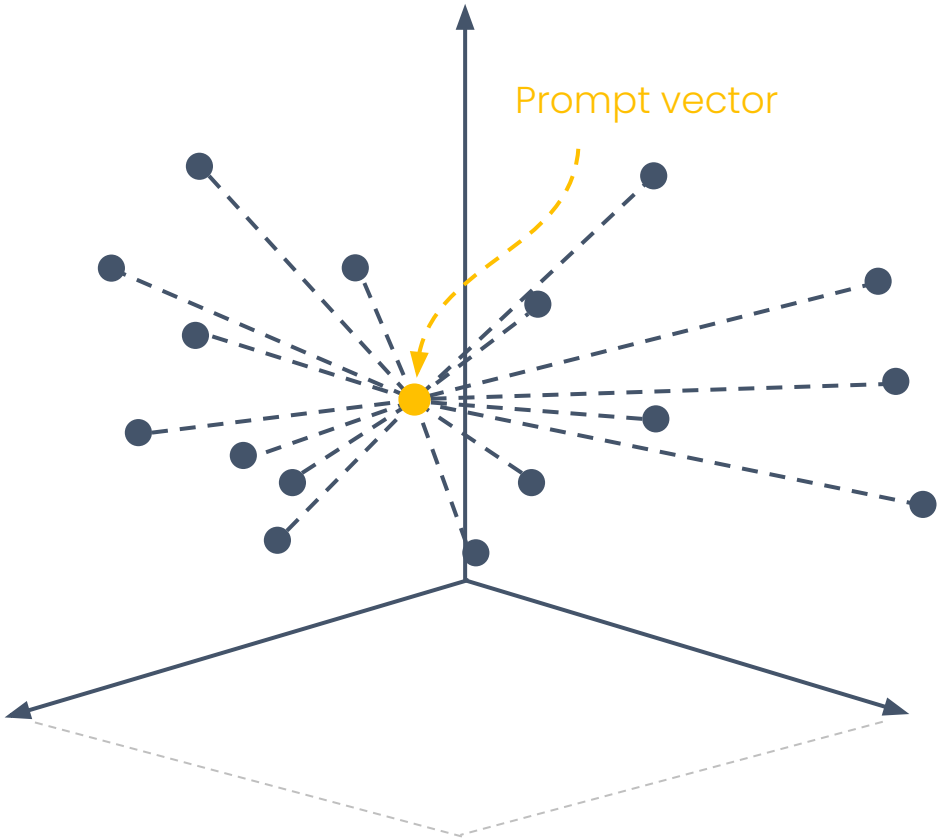
- Hands-on vector database usage
- Production techniques: chunking, query parsing, reranking
- Programming assignment applying all concepts

DeepLearning.AI

Approximate nearest neighbors algorithms (ANN)

Information Retrieval in Production

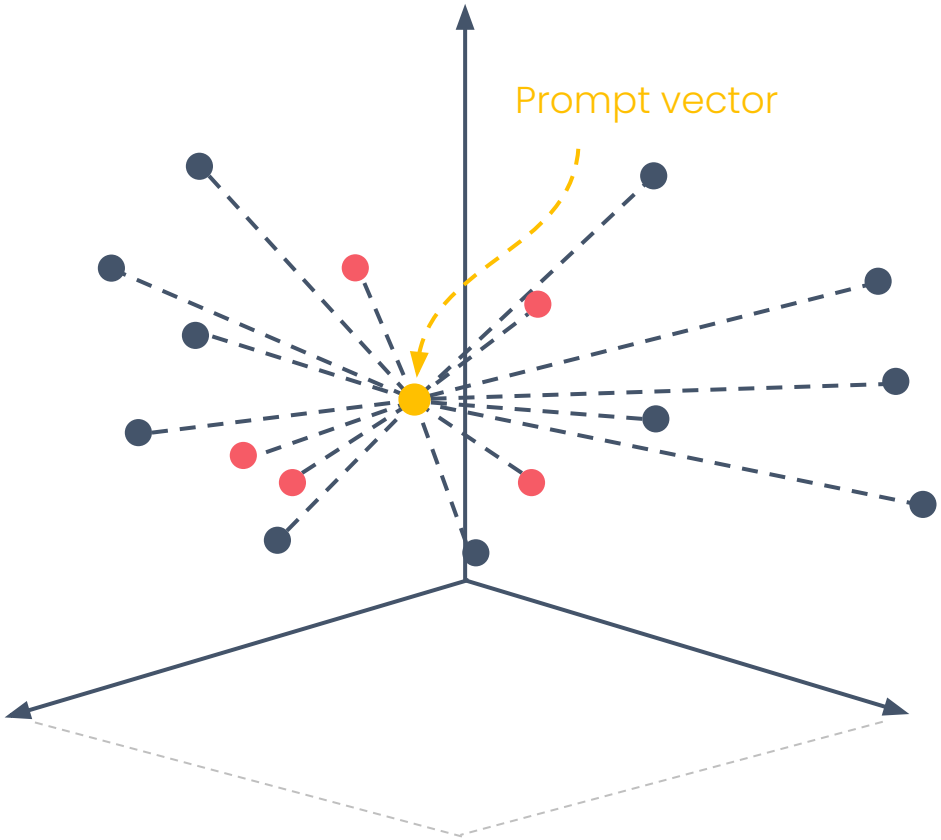
Basic Vector Retrieval - KNN



KNN: K Nearest Neighbors

- **Vectorize** all documents and prompt
- **Compute** distances to all document vectors
- **Sort** by distance

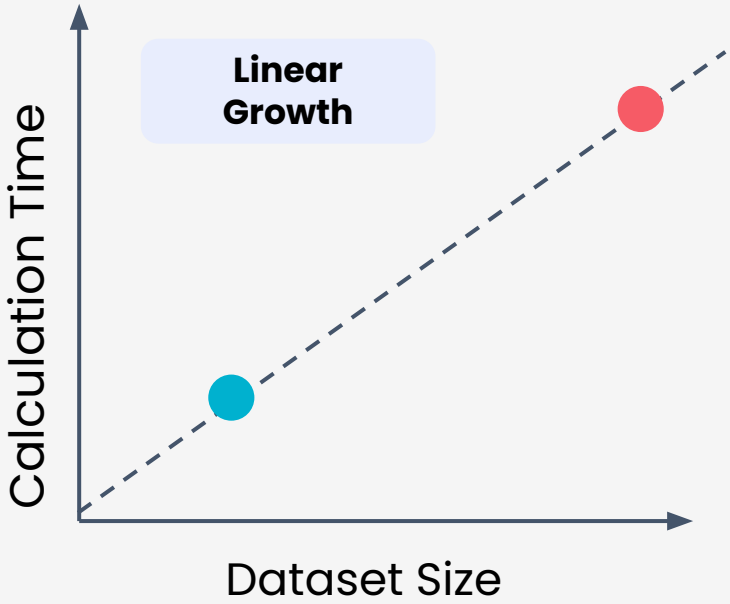
Basic Vector Retrieval - KNN



KNN: K Nearest Neighbors

- **Vectorize** all documents and prompt
- **Compute** distances to all document vectors
- **Sort** by distance
- **Return** closest elements

Scaling Challenges

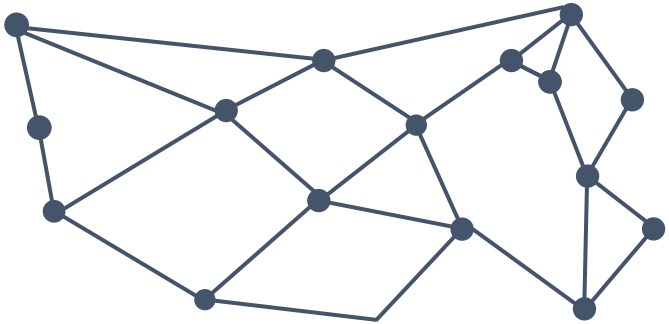


- 1,000 docs = 1,000 distance calculations per search
- 1B docs = 1B distance calculations every time

If you want to build high-performing retrievers, you'll need a better approach!

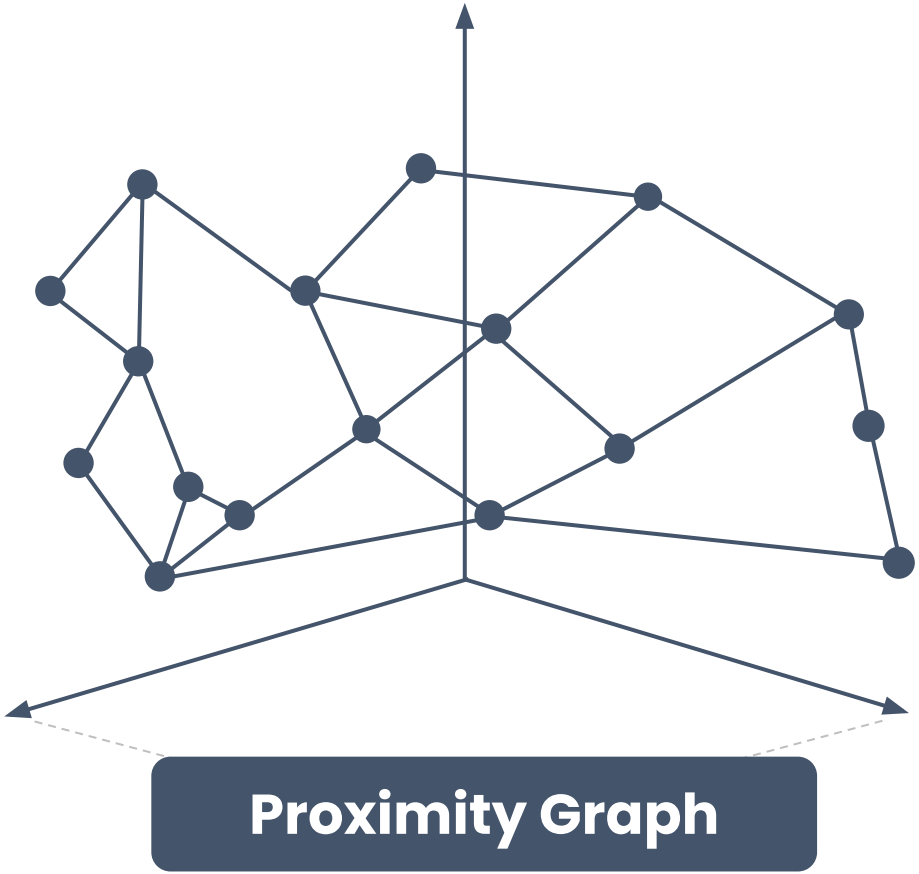
Approximate Nearest Neighbors

- ANN is significantly faster than KNN
- Rely on additional data structures
- Not guaranteed to find the absolute closest documents

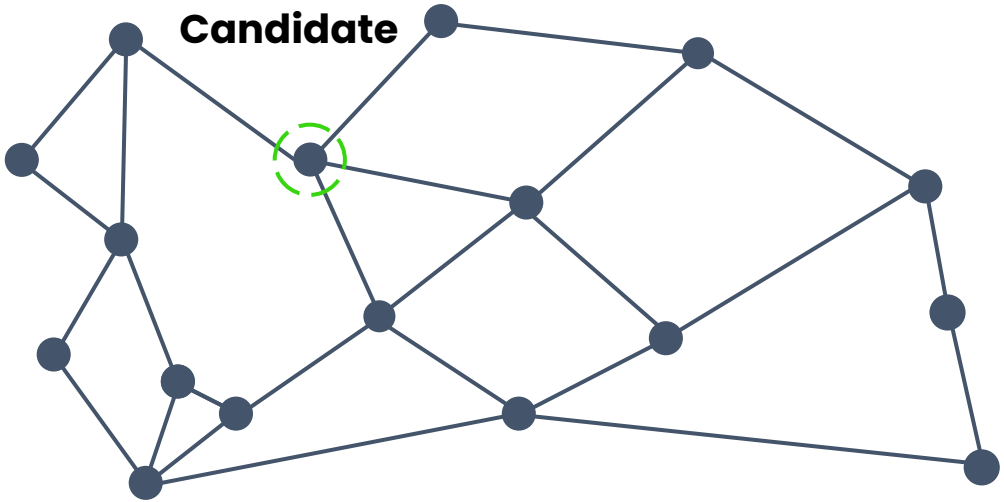


Navigable Small World

- Compute distances between all document vectors
- Add one node to the graph for each document
- Connect each node to its nearest neighbors
- Can traverse the graph moving along edges between neighboring documents

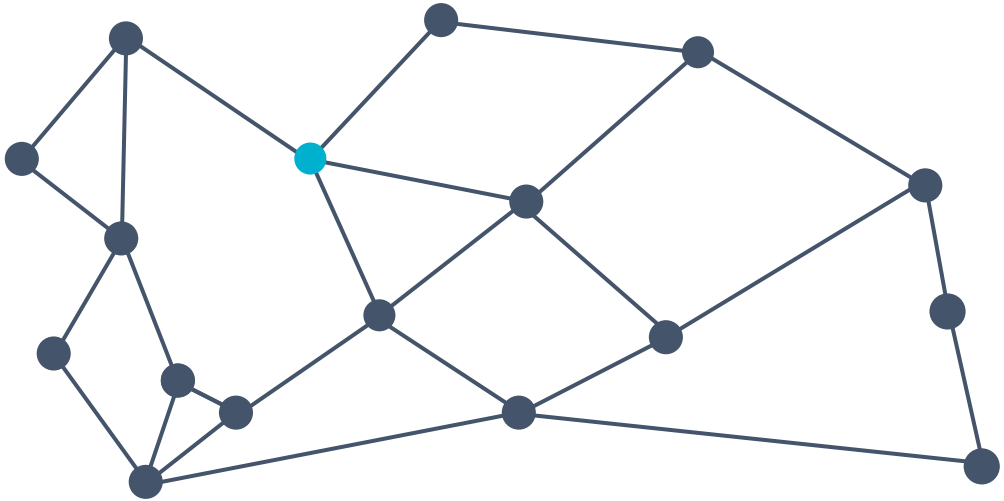


Query Entry Point



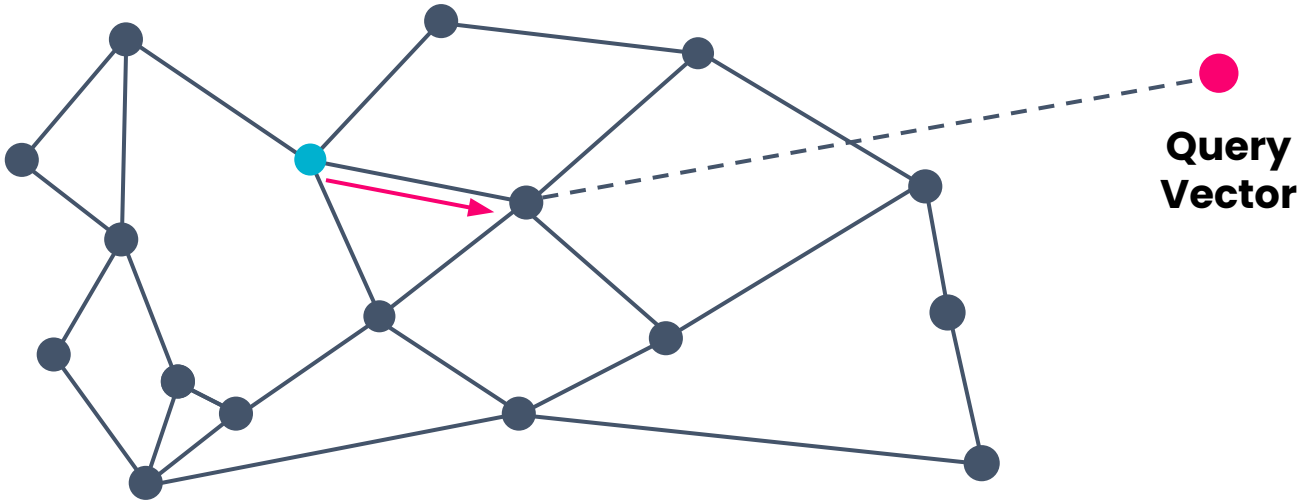
Query Vector

Search Algorithm

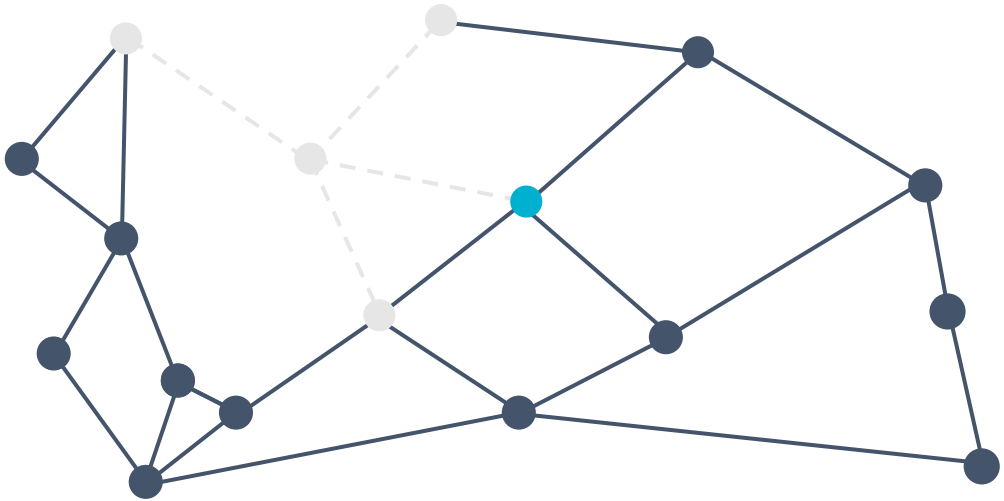


Query Vector

Search Algorithm



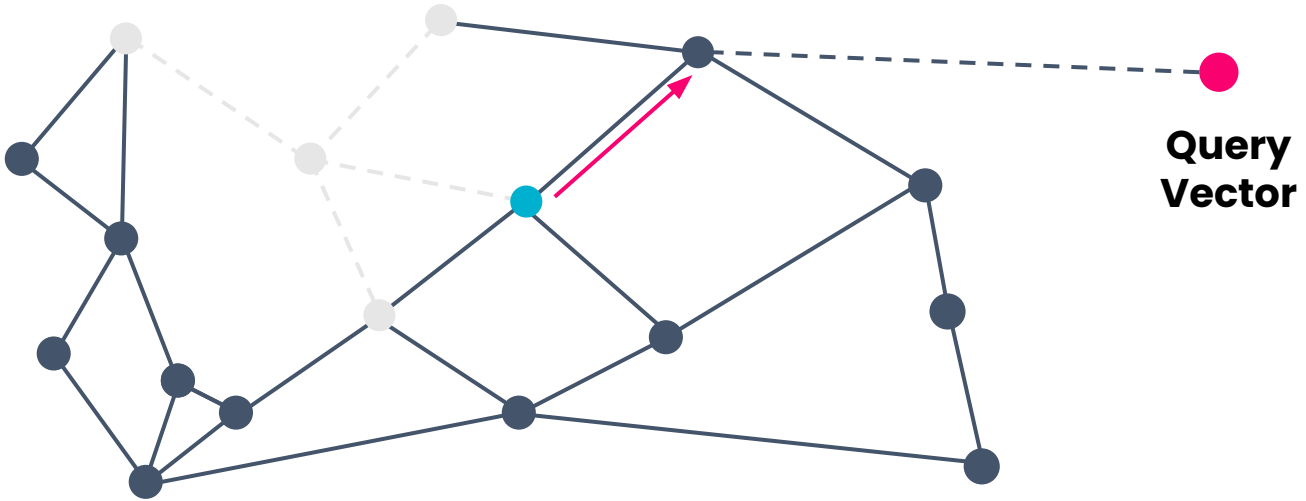
Search Algorithm



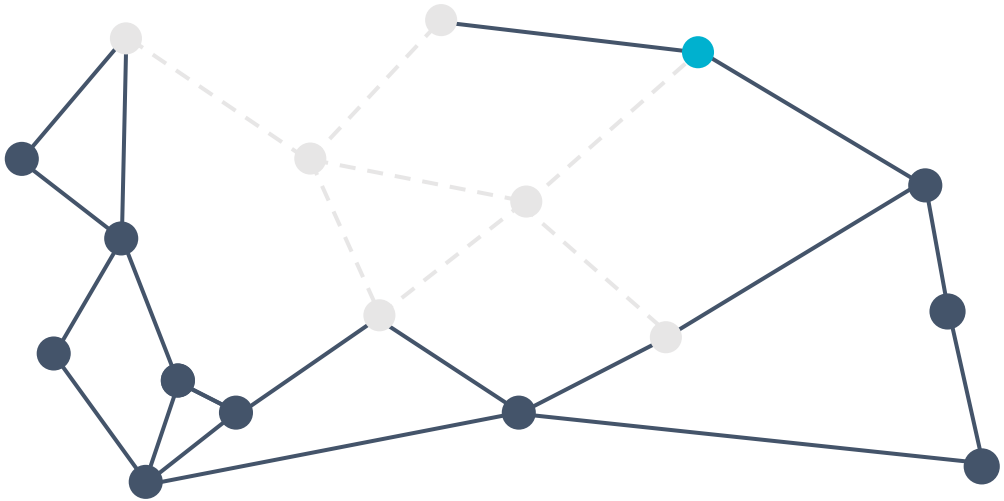
Query Vector

Zain Hasan

Search Algorithm

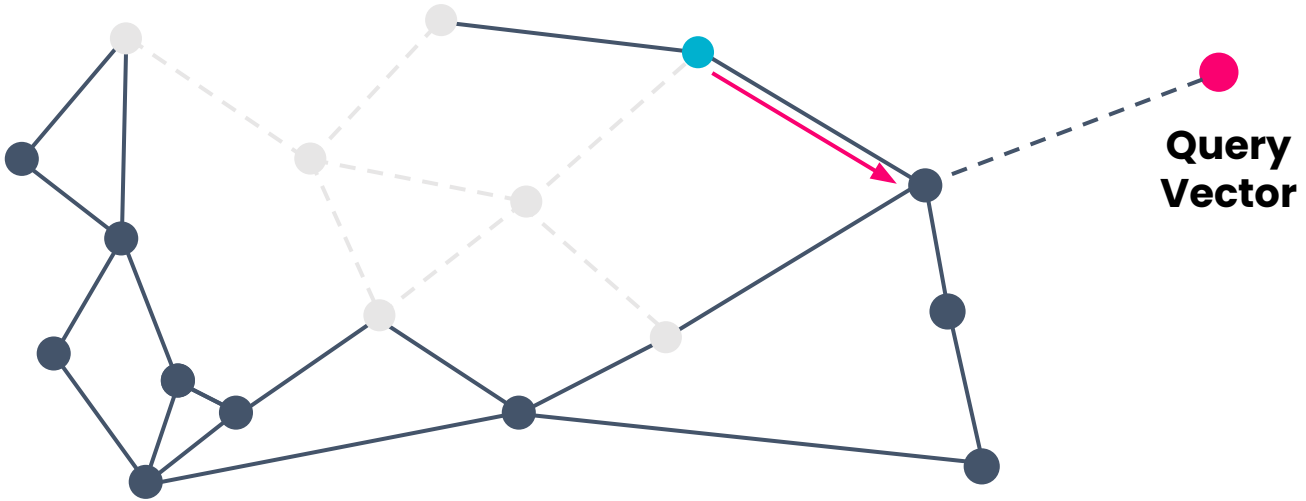


Search Algorithm

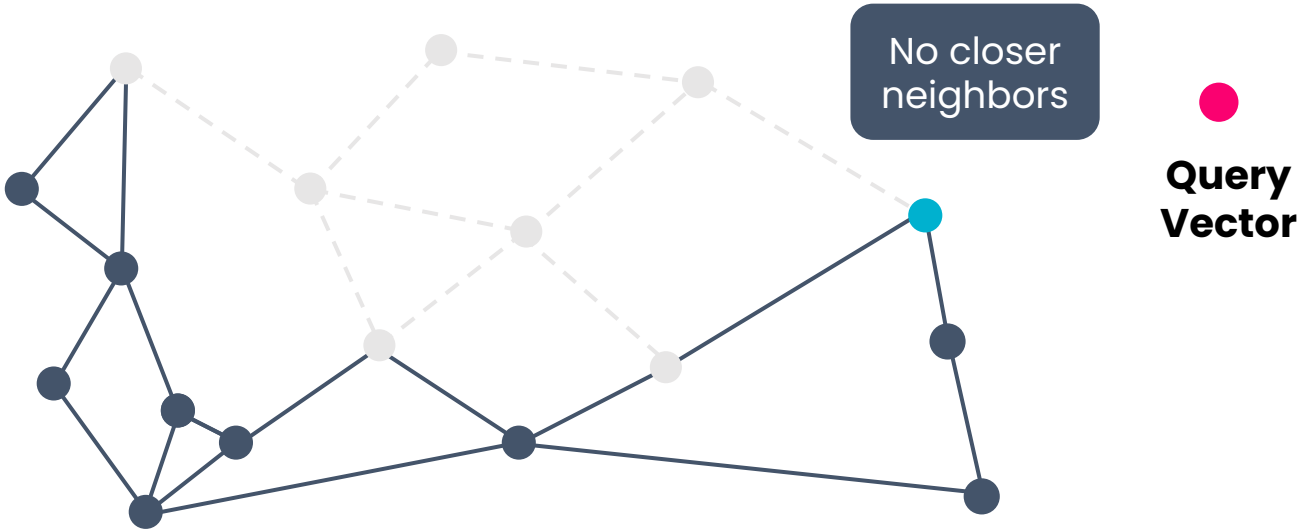


Query Vector

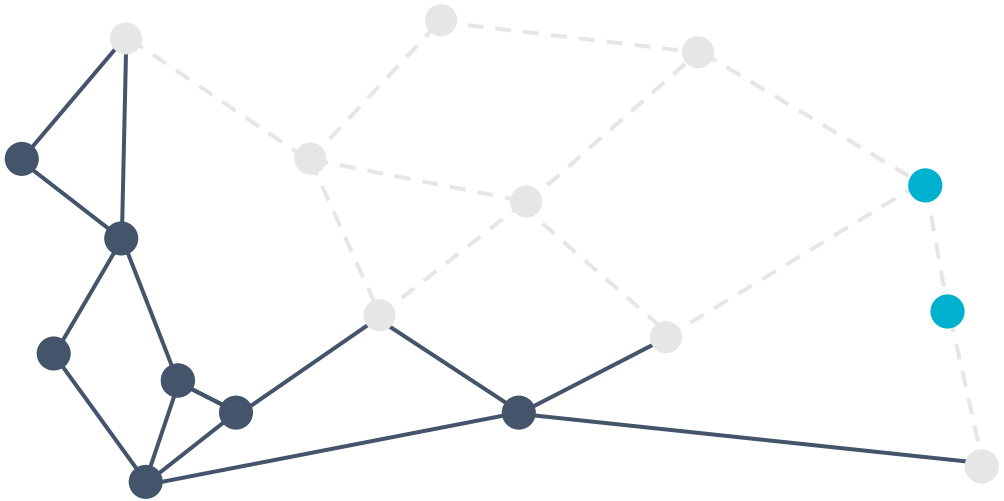
Search Algorithm



Search Algorithm



Search Algorithm



Query Vector

Can find multiple nearest neighbors

- May not find closest possible vectors, algorithm doesn't pick optimal overall path, just best path in each moment
- In practice performs well and much faster than KNN

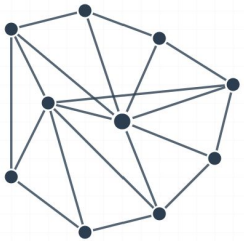
Hierarchical Improvement

- Hierarchical Navigable Small World (HNSW) enhances Navigable Small World by speeding up early parts of the search

- Relies on a hierarchical proximity graph

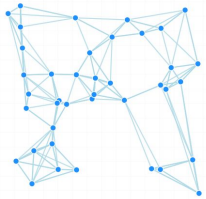
3 **Layer 3**

Randomly drop to just **10 vectors** and create a proximity graph for fast navigation at the highest level



2 **Layer 2**

Randomly drop to **100 vectors** and build a new proximity graph for intermediate navigation

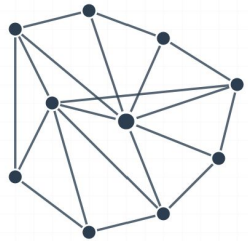


1 **Layer 1**

Contains all **1,000 vectors** with complete proximity graph for precise final search

3 Layer 3

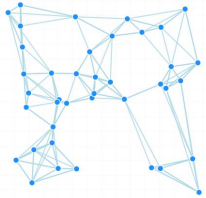
Choose random candidate vector and search top layer to get as close as you can in this layer



Make big jumps early

2 Layer 2

Start at best candidate from layer 3, and complete normal search through the layer 2 proximity graph



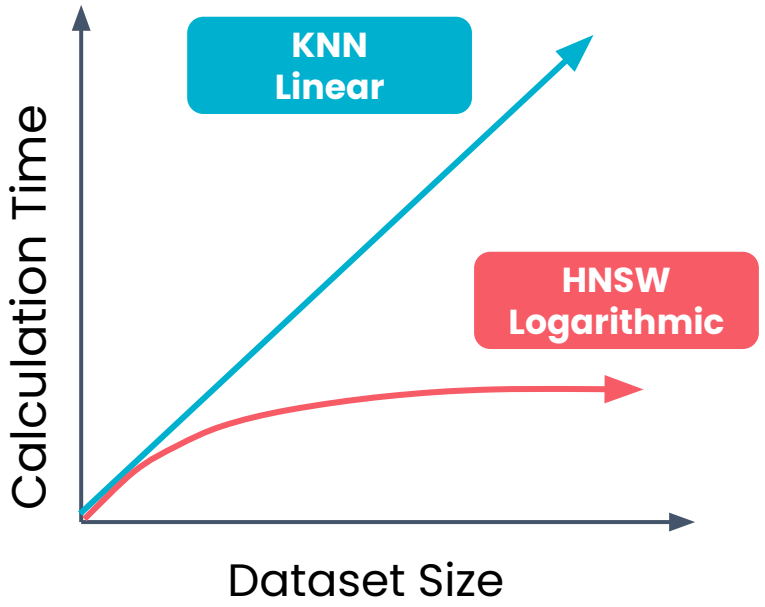
Close to prompt vector once in Layer 1

1 Layer 1

Start at best candidate in layer 2, and complete normal search through the layer 1 proximity graph

Hierarchical Navigable Small World

- Significantly faster than KNN
- Exponentially fewer vectors in each layer (for example 1000 → 100 → 10) makes it approximately logarithmic
- Allows scaling up to billions of vectors



Approximate Nearest Neighbors - Takeaways

- ANN is significantly faster than KNN at scale
- Find close documents, but can’t guarantee best matches
- Depends on proximity graph, computationally expensive to build but can be pre-computed

DeepLearning.AI

Vector databases

Information Retrieval in Production

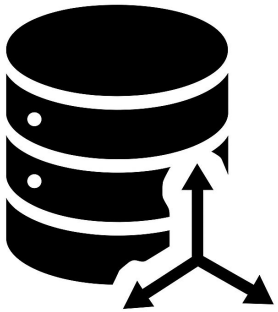
Vector Database

Designed for Vector Search
Designed to store high-dimensional vectors and perform vector search using ANN algorithms

Outperform Relational Databases
Relational databases perform vector search similarly to an inefficient KNN search

Optimized for ANN Search
Designed to build HNSW indexes and compute vector distances. They scale well and operate quickly

Weaviate
Popular open-source vector database you'll use in this course



Vector Database

Weaviate

Typical Vector Database Operations

- Database setup
- Load documents
- Create sparse vectors for keyword search
- Create dense vectors for semantic search
- Create HNSW index to power ANN algorithm
- Run searches!

Connect to Database and Create a Collection

```
from weaviate.classes.config import Configure, Property, DataType

client.collections.create(
    "Article",
    vectorizer_config=Configure.Vectorizer.text2vec_openai(),
    properties=[  # properties configuration is optional
        Property(name="title", data_type=DataType.TEXT),
        Property(name="body", data_type=DataType.TEXT),
    ]
)
```

Creates collection

Specifies vectorizer

Specifies Properties

Adding objects to a collection

```
with collection.batch.fixed_size(batch_size=200) as batch:
    for data_row in data_rows:
        batch.add_object(
            properties=data_row,
        )
        if batch.number_errors > 10:
            print("Batch import stopped due to excessive errors.")
            break

failed_objects = collection.batch.failed_objects
if failed_objects:
    print(f"Number of failed imports: {len(failed_objects)}")
    print(f"First failed object: {failed_objects[0]}")
```

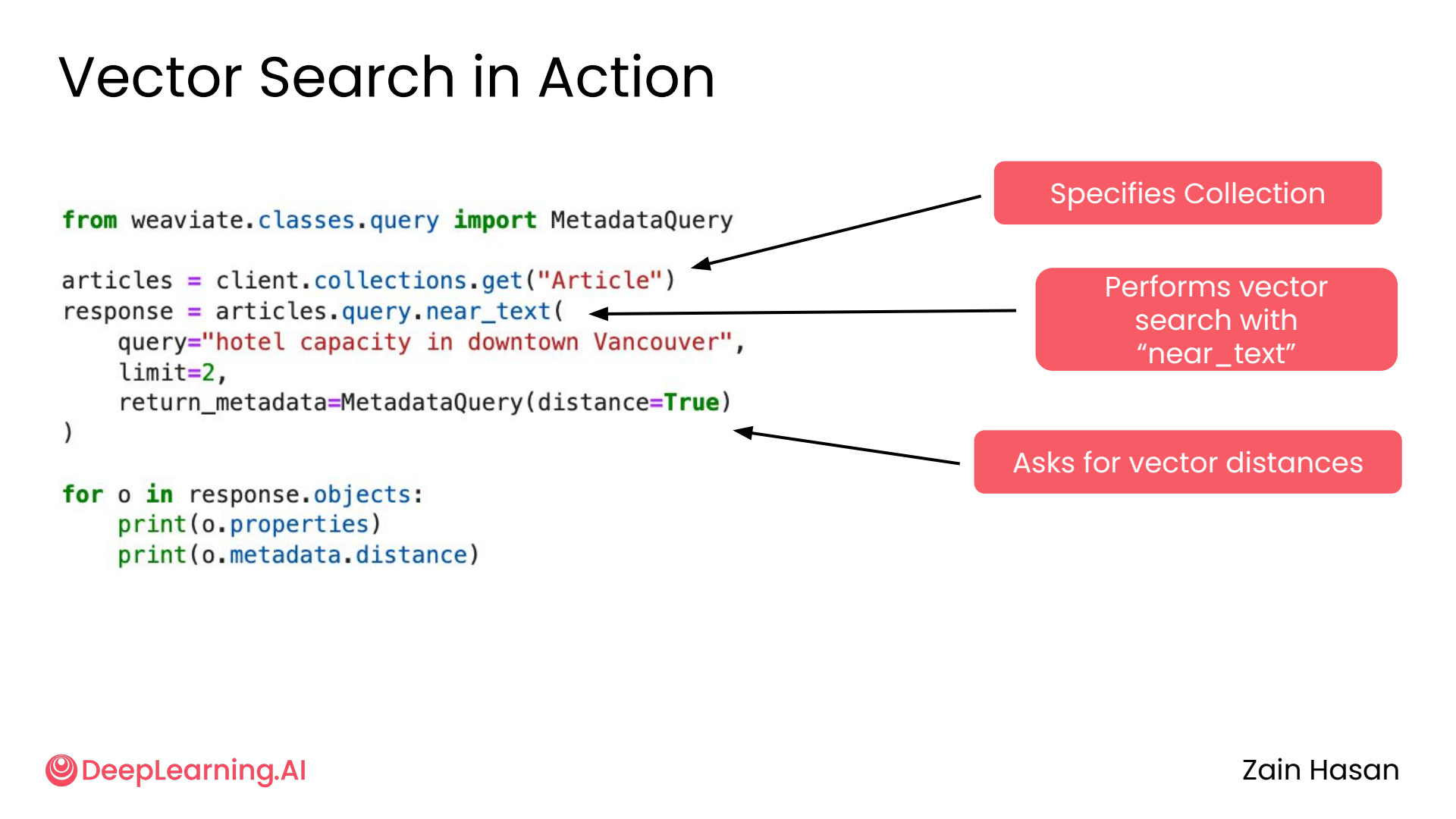
Batch adding data

Detecting errors

Handling failures

Vector Search in Action

```python
from weaviate.classes.query import MetadataQuery

articles = client.collections.get("Article")
response = articles.query.near_text(
    query="hotel capacity in downtown Vancouver",
    limit=2,
    return_metadata=MetadataQuery(distance=True)
)

for o in response.objects:
    print(o.properties)
    print(o.metadata.distance)
```



Keyword Search in Action

```
response = articles.query.bm25(
    query="Vancouver hotel capacity",
    limit=3
)

for o in response.objects:
    print(o.properties)
```

Hybrid Search in Action

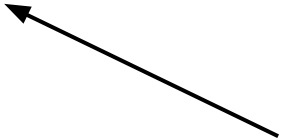
```
response = articles.query.hybrid(
    query="Vancouver hotel capacity",
    alpha=0.25,
    limit=3,
)

for o in response.objects:
    print(o.properties)
```

Hybrid search

Weighs the hybrid search to 25% vector, 75% keyword

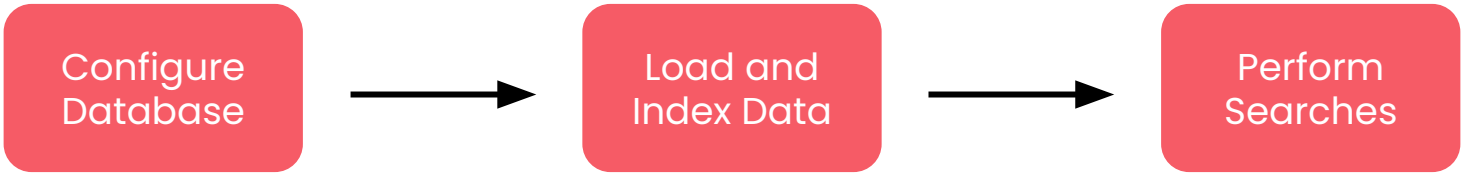
Filtered Search in Action

```
response = articles.query.hybrid(
    query = 'History of urban development in Vancouver',
    filters = Filter.by_property('title').contains_any(['Vancouver']),
    alpha = 0.3,
    limit = 4
)

for o in response.objects:
    print(o.properties)
```



Adding a metadata filter

Complete Workflow



DeepLearning.AI

Chunking

Information Retrieval in Production

Why Chunk Documents?

Token Limits

Improved Relevancy

LLM only sent relevant context

Indexing without chunking

- Knowledge base contains 1,000 books
- Each book is vectorized by an embedding model
- **Result:** 1,000 vectors

Entire Book

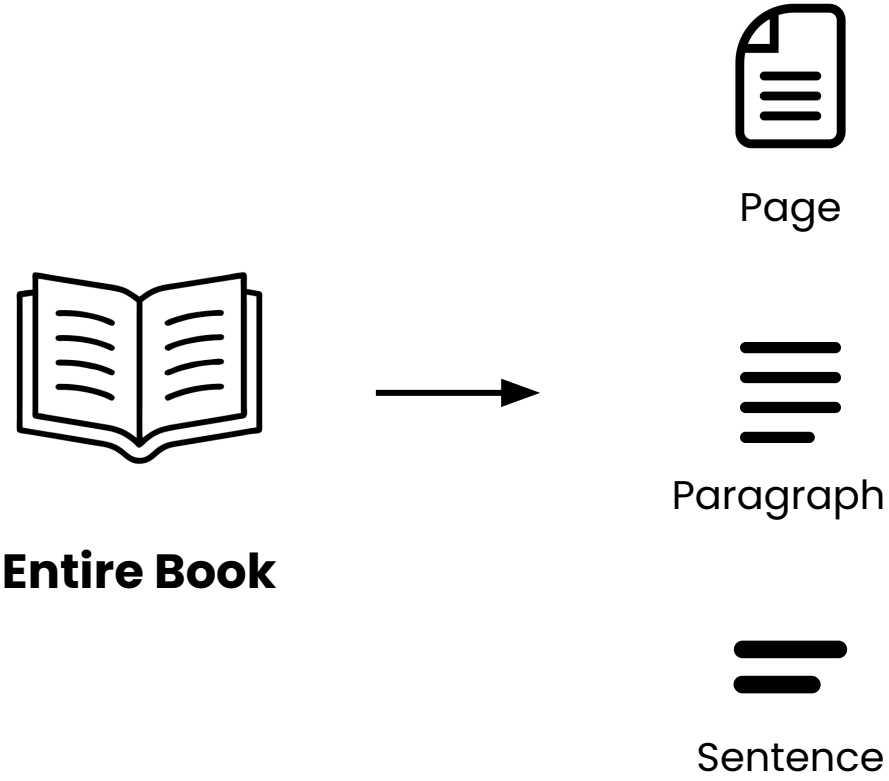
`[0.1,0.4,0.3,0.2,0.9]`

Entire book compressed into a single vector

The problems with this approach

- Compresses entire book meaning into single vector
- Can't sharply represent specific topics, chapters or pages
- Creates "averaged" representation across all content
- Results in poor search relevance
- Retrieves entire books, quickly filling LLM context window

Chunking your content



Balancing Chunk Size

Too Large

Chapter level chunks

- Too many topics in one vector
- Fill LLM context window

Too Small

Word level chunks

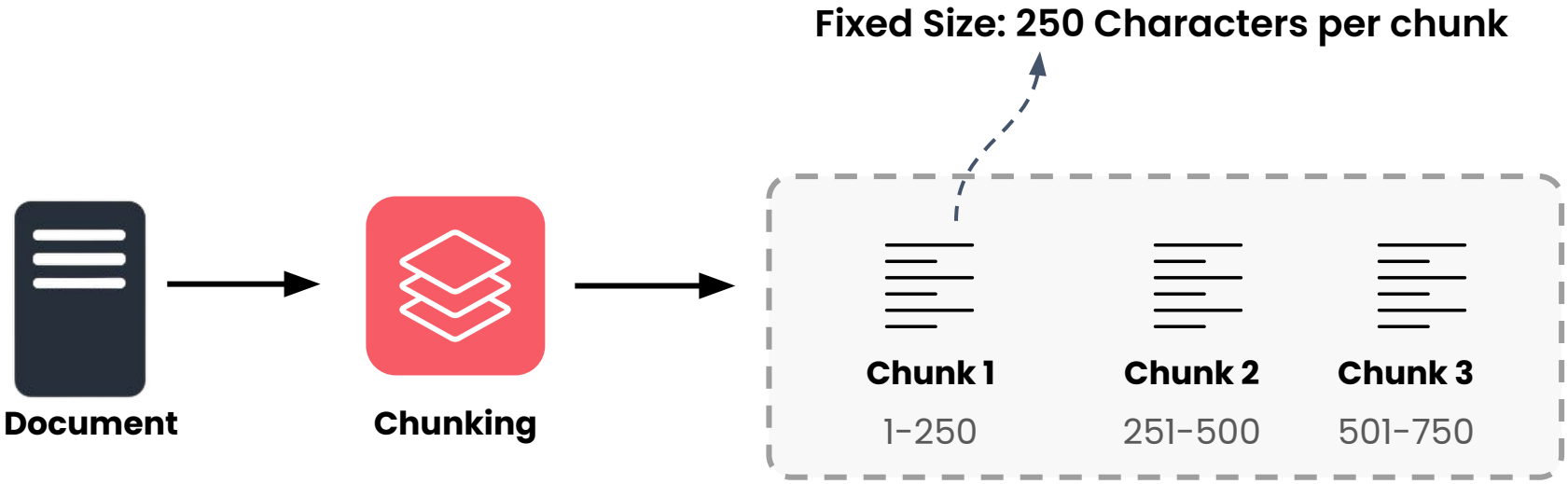
- Loses surrounding context
- Reduces search relevance

Just Right

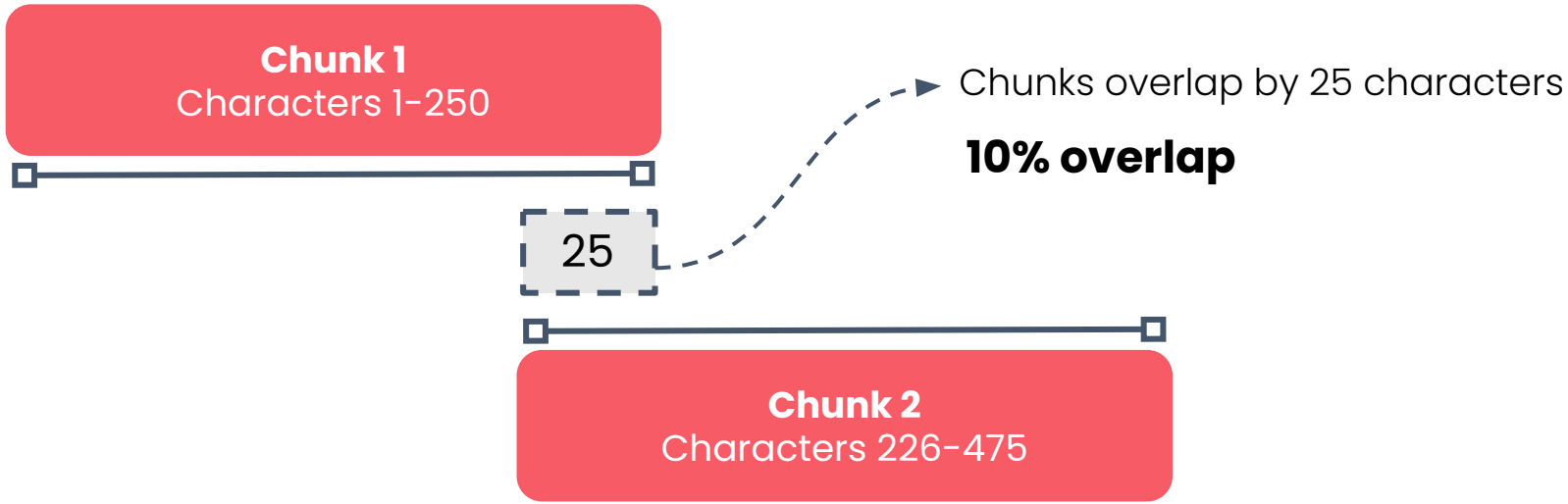
Optimal Chunks

- Balance between capturing too much and too little

Fixed Size Chunking



Overlapping Chunking



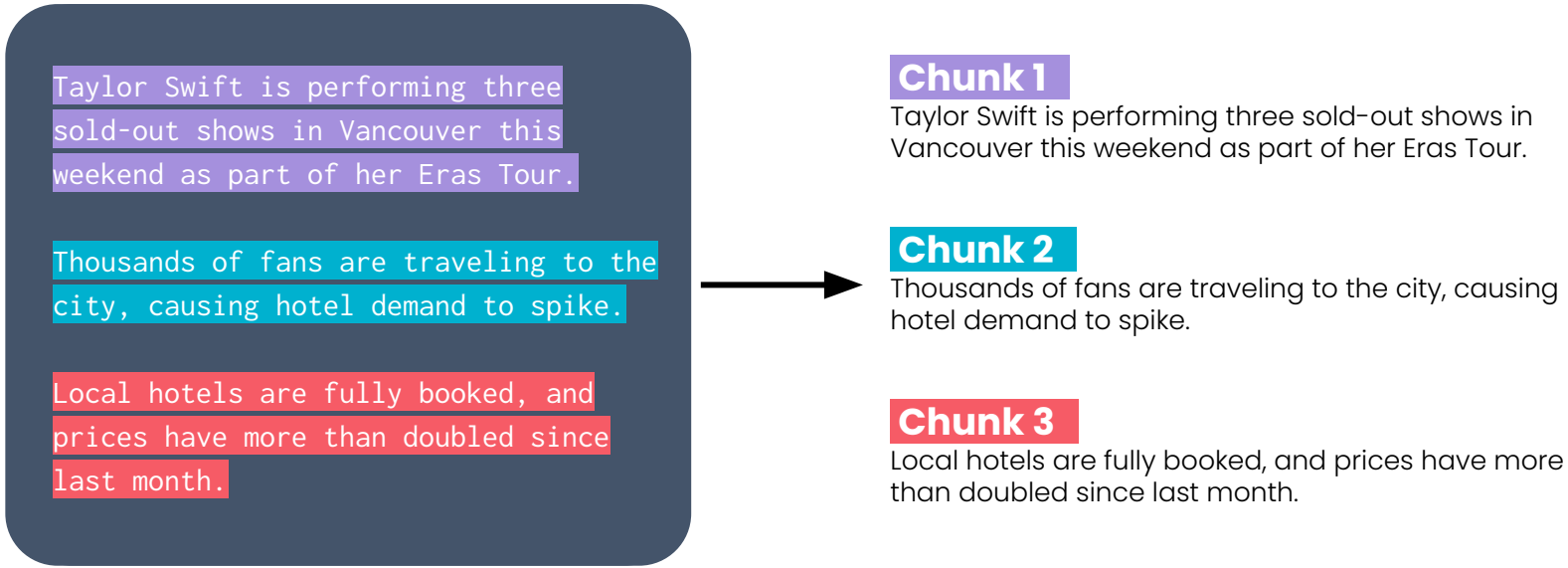
- Minimizes words cut off from context
- Words in the middle of a chunk have context on each side, words on the ends are in two chunks
- Increases relevancy, uses more space

Chunk 3
Characters 451-700

Recursive Character Splitting

Splitting text into chunks at a specified character, for example newlines

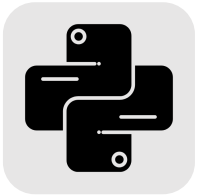
Variable chunk size, but better accounts for document structure



Splitting on Different Characters

HTML Documents

Split paragraph or header characters



Python

Chunk by function definitions

```
<h1>Document Title</h1>
<p>First paragraph</p>
<p>Second paragraph</p>
def function_one():
    """Docstring"""
    return result
class MyClass:
    def __init__(self):
        self.value =
```

Implementing Chunking Strategies

- **Implement Yourself vs. Libraries**
  Fixed size splitting with overlaps is straightforward to implement yourself or with external libraries

- **Metadata Preservation**
  Chunks inherit source document metadata plus location information

DeepLearning.AI

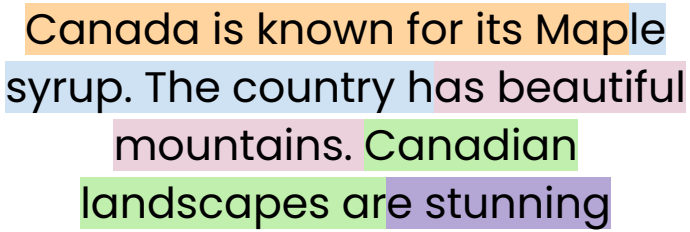
Advanced chunking techniques

Information Retrieval in Production

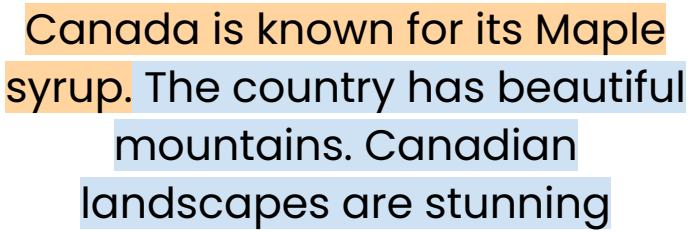
“That night she dreamed, as she did often, that she was finally an Olympic champion”

```
That night she dreamed, as she
did often, that she was finally
an Olympic champion
```

Semantic Chunking



Fixed Size Chunking



Semantic Chunking

Groups sentences together based on **similar meanings** rather than arbitrary character limits

Move Through Document
Process the document sentence by sentence

Vectorize & Compare
Convert chunk and next sentence to vectors, calculate cosine distance

Check Threshold
If distance is below threshold, add sentence to chunk

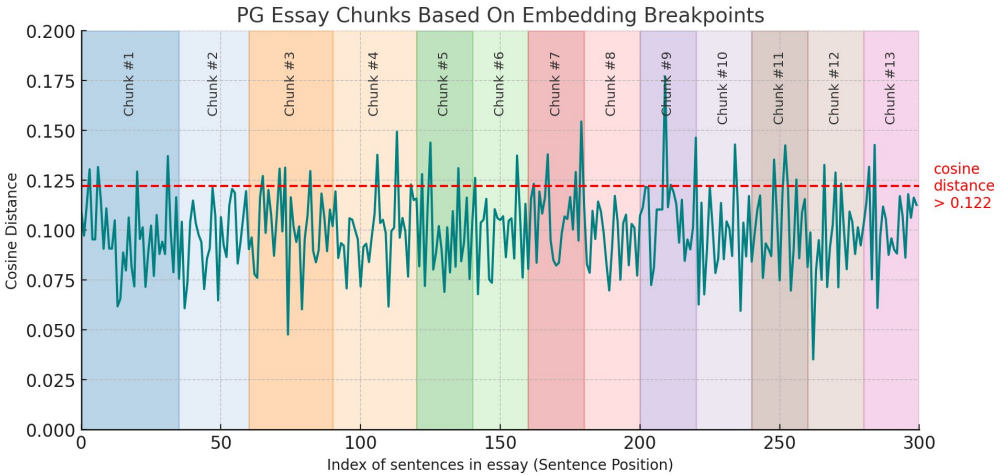
Split when Different
When distance crosses threshold, start new chunk

Canada is known for its Maple syrup. The country has beautiful mountains. Canadian landscapes are stunning

Semantic Chunking

Semantic Chunking

I was puzzled by the 1401. I couldn't figure out what to do with it.
And in retrospect there's not much I could have done with it.
The only form of input to programs was data stored on punched cards...



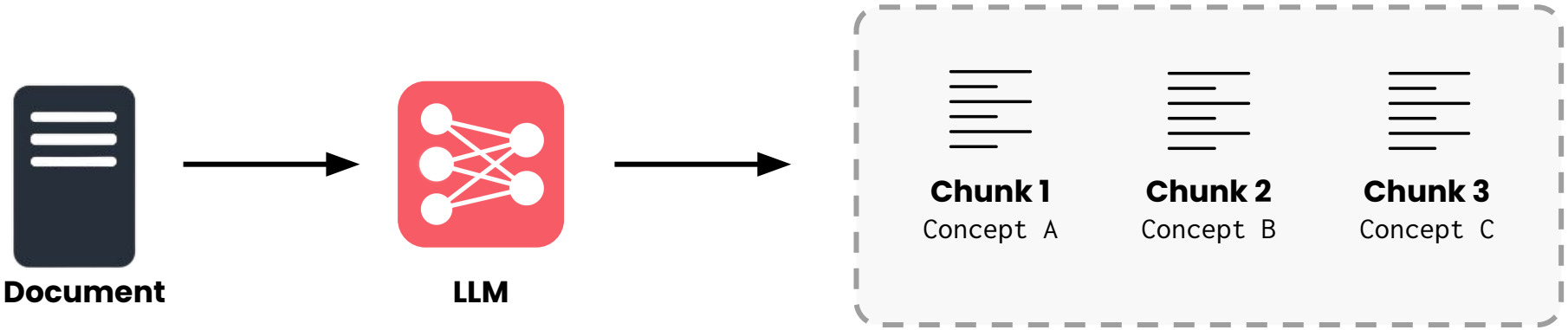
Kamradt, G. (2023). *5 Levels of Text Splitting*. GitHub.

Pros and Cons

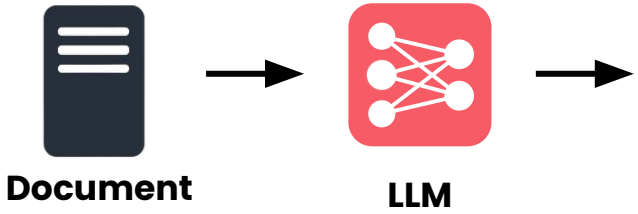
Pros	Cons
• Follows author's train of thought • Smarter chunk boundaries • Higher recall and precision	• Chunking can be Computationally expensive • Requires repeated vector calculations

Language Based Chunking

- Prompt LLM to create chunks from a document
- Include instructions on types of chunks, like keeping concepts together, adding breaks when new topic starts
- Performs well, increasingly more economically viable



Context-Aware Chunking



"Reflections on Launching Our First App"

After six months of development, we finally launched our first app...

The biggest lessons were around user feedback...

Thanks to Daniel Townsend, Sarah Johnson, Mike Chen,...

Chunks

`context: app launch title.`

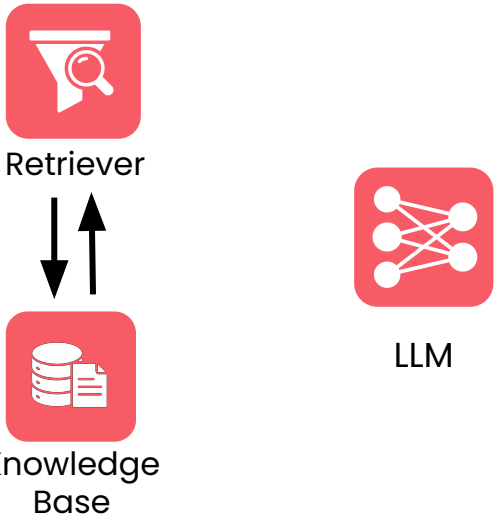
`context: launch announcement.`

`context: summary of key development lessons.`

`context: list of supporters and contributors.`

Additional Context

Context-Aware Chunking



Added context helps both search relevance and as the LLM generates responses

"Reflections on Launching Our First App"

After six months of development, we finally launched our first app...

The biggest lessons were around user feedback...

Thanks to Daniel Townsend, Sarah Johnson, Mike Chen,...

- **Costly pre-processing:** LLM adds context to each chunk.
- search, no impact on speed.

`context: app launch title.`

`context: launch announcement.`

`context: summary of key development lessons.`

`context: list of supporters and contributors.`

Additional Context

Choosing a Chunking Approach

- **Fixed Width and Recursive Character Splitting:** good defaults
- **Semantic and LLM Chunking:** can yield higher performance, but more complex. Experiment to see if it's worth it
- **Context Aware Chunking:** improves any chunking technique at some cost. A good "first improvement" to explore

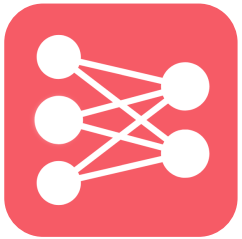
DeepLearning.AI

Query parsing

Information Retrieval in Production

Prompt

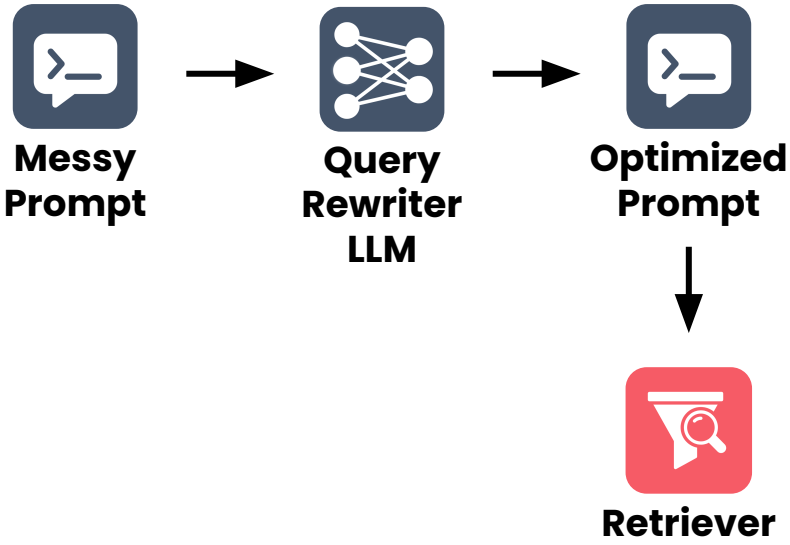
Hey, can you tell me about that **climate thing** we talked about?"



LLM

Query Rewriting

Use an LLM to rewrite the query **before** it's submitted to the retriever.




```
The following prompt was submitted by a
user in order to query a database of
medical documents linking symptoms to
diagnoses. Rewrite the prompt to
optimize it for searching the database
by doing the following:

 - Clarify ambiguous phrases
 - Use medical terminology where
applicable
 - Add synonyms that increase odds of
finding matching documents
 - Remove unnecessary or distracting
information.

{user prompt}
```

Patient's Original Prompt

I was out walking my dog, a beautiful black lab named Poppy, when she raced away from me and yanked on her leash hard while I was holding it.

Three days later my shoulder is still numb and my fingers are all pins and needles. What's going on?

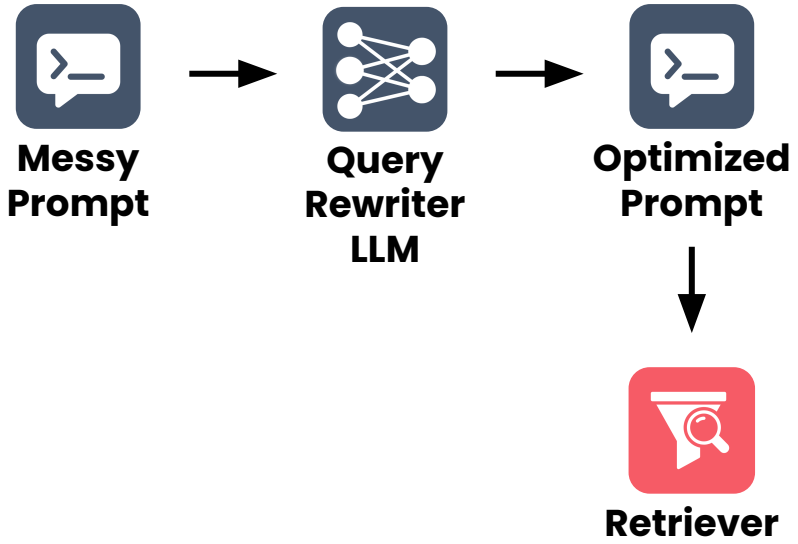
LLM Query Rewriter

Optimized Prompt

```
Experienced a sudden, forceful pull on the shoulder, resulting in persistent
shoulder numbness and finger numbness for three days. What are the potential
causes or diagnoses, such as neuropathy or nerve impingement?
```

Query Rewriting

- Iterate on the prompt for your query rewriter
- Benefits are substantial, easily justify costs



Named Entity Recognition

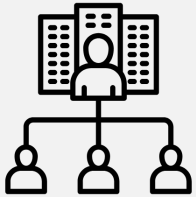
Identifies and categorizes specific types of information within queries, enabling more targeted search and filtering strategies.

Places

Dates

People

Characters



Organizations

Original Prompt

"I read The Great Gatsby by F. Scott Fitzgerald last summer while visiting New York. Are there any similar books set in the 1920s that I might enjoy?"

Entities

Person, books, location, dates, actors, characters

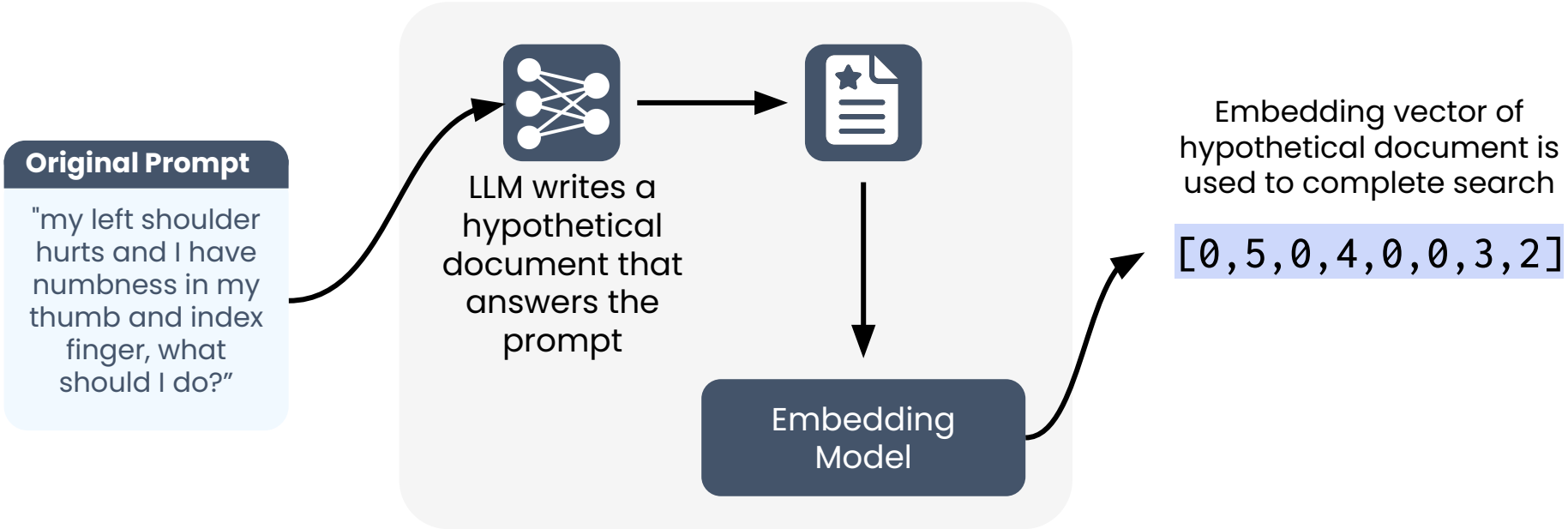
GLINER Named Entity Recognition

Labeled Prompt

"I read The Great Gatsby BOOK by F. Scott Fitzgerald PERSON last summer while visiting New York. LOCATION Are there any similar books set in the 1920s DATE that I might enjoy?"

Hypothetical Document Embeddings (HyDE)

Uses generated “hypothetical documents” that would be ideal search results to help with the search process



HyDE

Hypothetical Document Embeddings (HyDE)

- Normally a retriever is matching prompts to documents
- HyDE means the retriever is matching documents to documents, one is the “perfect” hypothetical one generated from the prompt
- Can provide performance improvements but adds latency and some cost

DeepLearning.AI

Cross-encoders and ColBERT

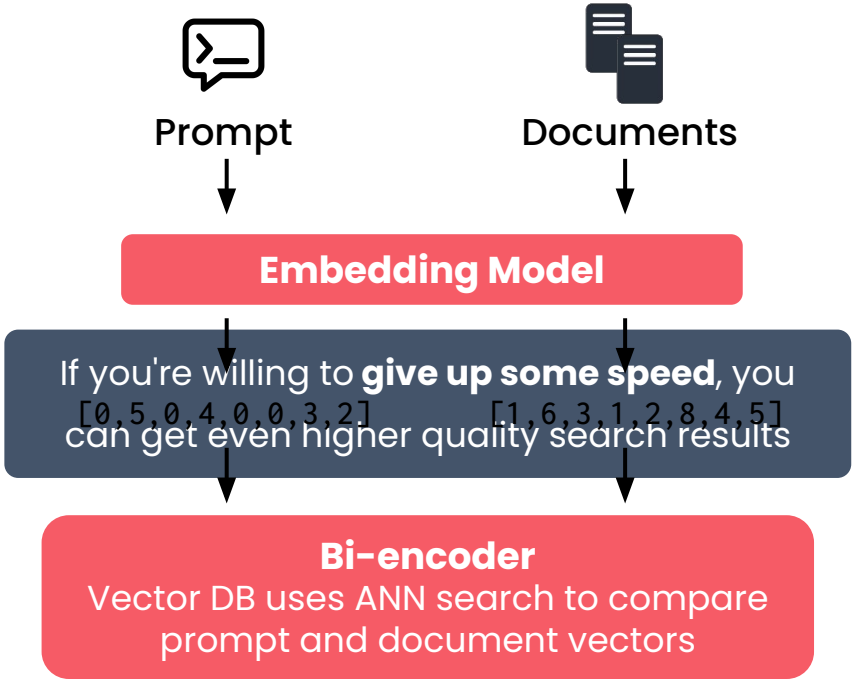
Information Retrieval in Production

Bi-Encoder

Separate Semantic Vectors
Documents and prompts are embedded separately by an embedding mode

ANN Search
ANN is used by a vector database to rapidly identify documents whose vectors are close to the prompt vector.

Document Vectors are Pre-Computed
Documents can be embedded ahead of time and only the prompt itself needs to be embedded after it's received

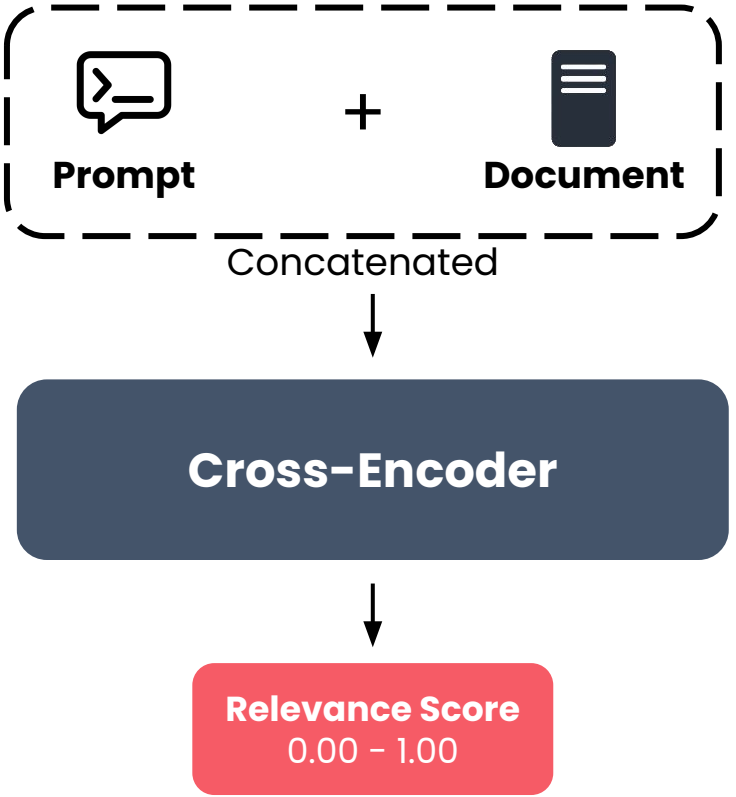


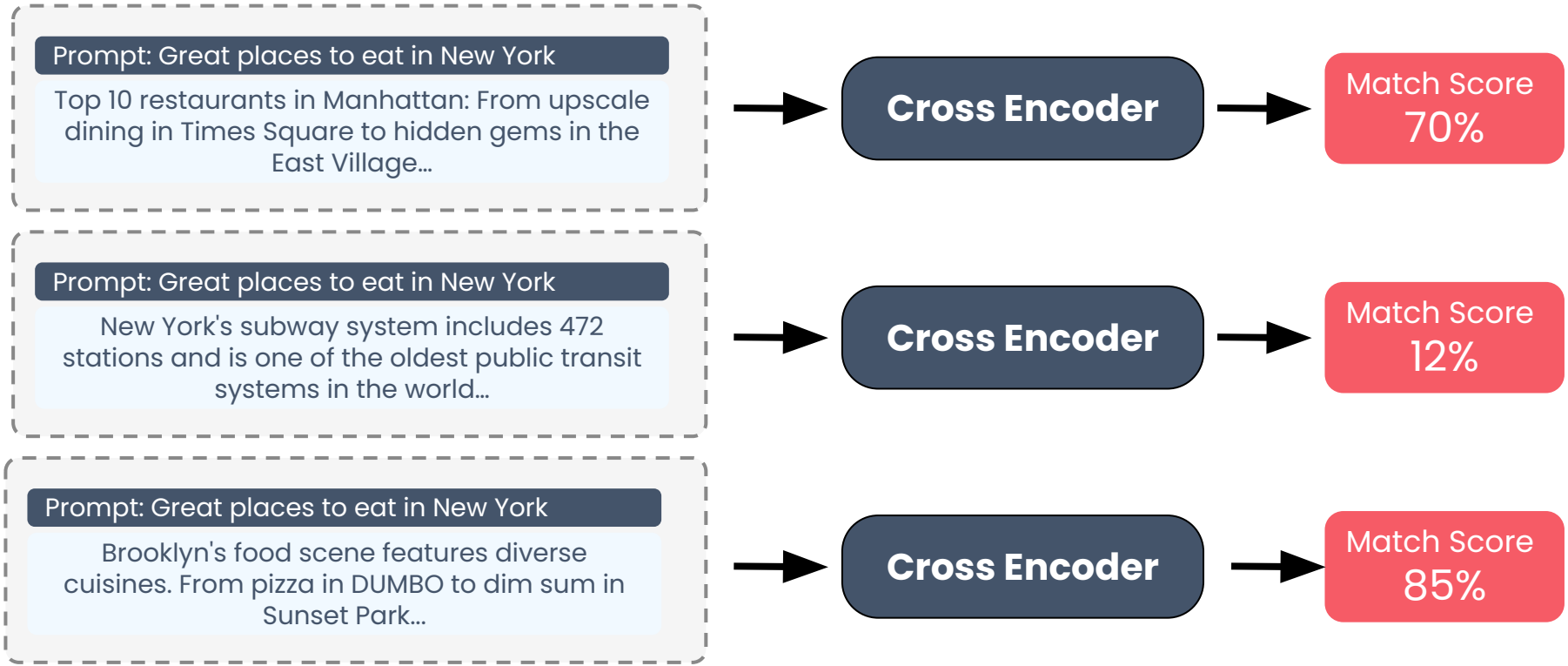
Cross-Encoder

Concatenate Document and Prompt
One by one documents are concatenated with the prompt

Feed to Cross Encoder
Cross encoder develops deep contextual understanding of interactions between prompt and document

Generate Relevance Score
Cross encoder directly outputs relevance score between 0 and 1




Prompt: Great places to eat in New York
Top 10 restaurants in Manhattan: From upscale dining in Times Square to hidden gems in the East Village...
Cross Encoder
Match Score
70%
Prompt: Great places to eat in New York
New York's subway system includes 472 stations and is one of the oldest public transit systems in the world...
Cross Encoder
Match Score
12%
Prompt: Great places to eat in New York
Brooklyn's food scene features diverse cuisines. From pizza in DUMBO to dim sum in Sunset Park...
Cross Encoder
Match Score
85%

Cross-Encoder Pros and Cons

Pros

- Almost always provide better search results than a bi-encoder
- Great for improving the results of other search techniques

Cons

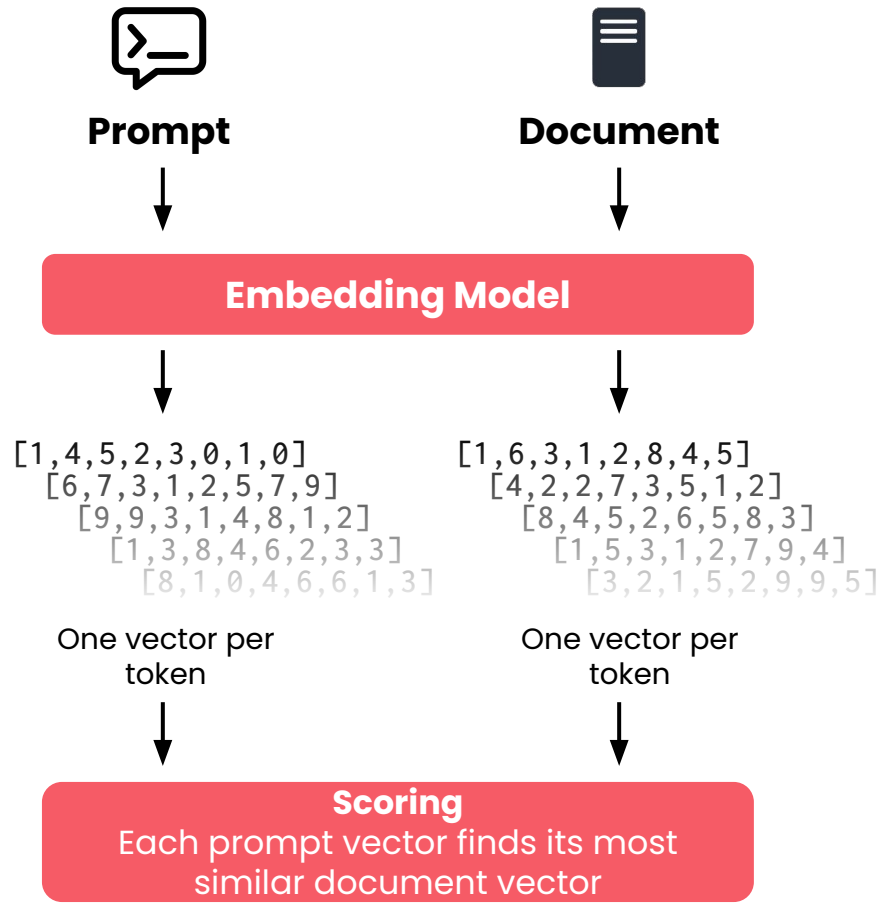
- Scale terribly with millions or billions of documents
- Can't pre-process since they run on prompt-document pairs
- Too inefficient to use as a default search technique

ColBERT (Contextualized Late Interaction Over BERT)

Split the Difference between Bi and Cross Encoders
Generate document vectors ahead of time like bi-encoders but also capture deep text interactions like in cross-encoders

Each Token Gets a Vector
Rather than generating one vector for each prompt and document, each token is vectorized

ColBERT Scoring
Each prompt vector tries to find its most similar document vector



Zain Hasan

Document Tokens

Prompt Tokens

	The	cuisine	of	New	York	City	comprises	many	. . .
Great	0.1	0.0	0.0	0.0	0.1	0.1	0.0	0.3	. . .
places	0.0	0.2	0.0	0.2	0.2	0.3	0.1	0.0	. . .
to	-0.1	-0.5	0.1	-0.3	0.1	-0.2	0.2	0.1	. . .
eat	0.0	0.7	-0.2	-0.1	-0.2	0.1	-0.9	-0.3	. . .
in	0.0	-0.2	0.3	-0.2	0.3	-0.2	0.2	0.2	. . .
New	0.2	0.2	0.2	0.9	0.8	0.7	0.2	-0.3	. . .
York	0.1	0.2	0.1	0.8	0.9	0.8	0.2	-0.1	. . .

Similarity scores between document and prompt tokens

Document Tokens

Prompt Tokens

	The	cuisine	of	New	York	City	comprises	many	. . .
Great	0.1	0.0	0.0	0.0	0.1	0.1	0.0	0.3	. . .
places	0.0	0.2	0.0	0.2	0.2	0.3	0.1	0.0	. . .
to	-0.1	-0.5	0.1	-0.3	0.1	-0.2	0.2	0.1	. . .
eat	0.0	0.7	-0.2	-0.1	-0.2	0.1	-0.9	-0.3	. . .
in	0.0	-0.2	0.3	-0.2	0.3	-0.2	0.2	0.2	. . .
New	0.2	0.2	0.2	0.9	0.8	0.7	0.2	-0.3	. . .
York	0.1	0.2	0.1	0.8	0.9	0.8	0.2	-0.1	. . .

Similarity scores between document and prompt tokens

Document Tokens

Prompt Tokens

	The	cuisine	of	New	York	City	comprises	many	. . .
Great	0.1	0.0	0.0	0.0	0.1	0.1	0.0	0.3	. . .
places	0.0	0.2	0.0	0.2	0.2	0.3	0.1	0.0	. . .
to	-0.1	-0.5	0.1	-0.3	0.1	-0.2	0.2	0.1	. . .
eat	0.0	0.7	-0.2	-0.1	-0.2	0.1	-0.9	-0.3	. . .
in	0.0	-0.2	0.3	-0.2	0.3	-0.2	0.2	0.2	. . .
New	0.2	0.2	0.2	0.9	0.8	0.7	0.2	-0.3	. . .
York	0.1	0.2	0.1	0.8	0.9	0.8	0.2	-0.1	. . .

$$0.3 + 0.3 + 0.2 + 0.7 + 0.3 + 0.9 + 0.9 = 3.6$$

MaxSim score for this document

ColBERT Pros and Cons

Pros

- Scalability of bi-encoder, much of the rich interactions of a cross-encoder
- Reasonably fast, can still be used in real-time or close-to-real-time scenarios

Cons

- Requires significant vector storage as each token, rather than each document, needs a dense vector

Key Takeaways

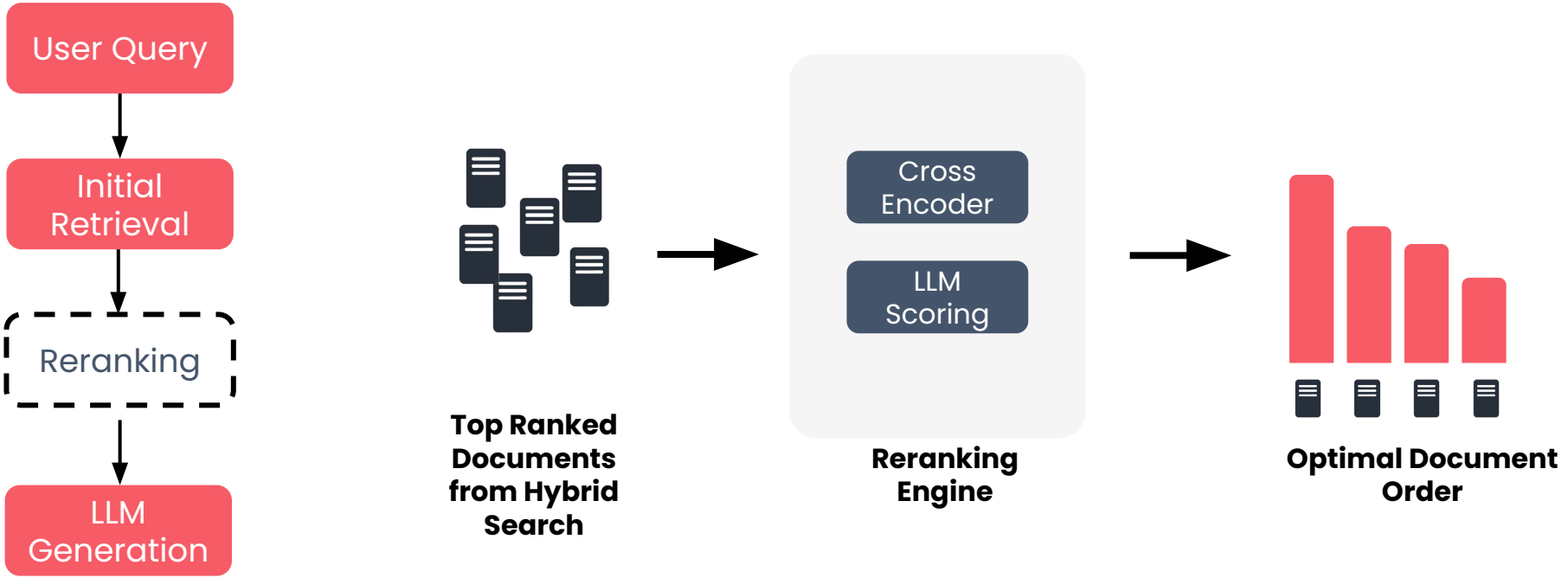
- **Bi-encoders:** reasonably good quality, great speed, minimal storage, default semantic search
- **Cross-encoders:** best quality, extremely slow, minimal storage
- **ColBERT:** nearly the quality of a cross-encoder, decent speed, significant vector storage.
- ColBERT and similar approaches increasingly supported by vector DBs

DeepLearning.AI

Reranking

Information Retrieval in Production

Purpose of Reranking



Original Query

What is the capital of Canada?

Semantically similar but **not** all directly **relevant**

Initial Results

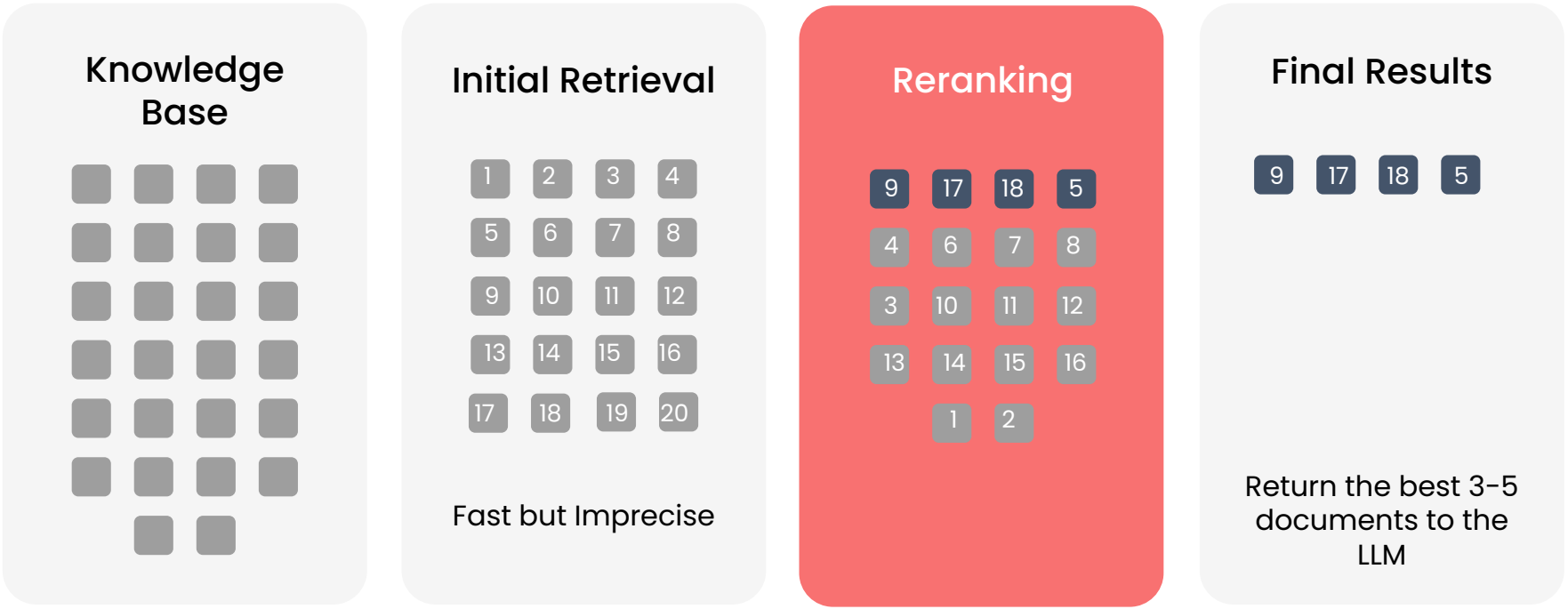
Toronto is in Canada

The capital of France is in Paris

Canada is the maple syrup capital of the world

Ottawa is the capital of Canada

Overview


Knowledge Base
Initial Retrieval
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
17 18 19 20
Fast but Imprecise
Reranking
9 17 18 5
4 6 7 8
3 10 11 12
13 14 15 16
1 2
Final Results
9 17 18 5
Return the best 3-5 documents to the LLM

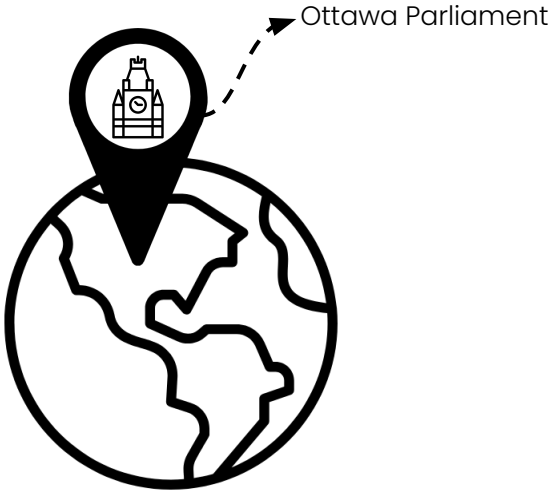
Original Query

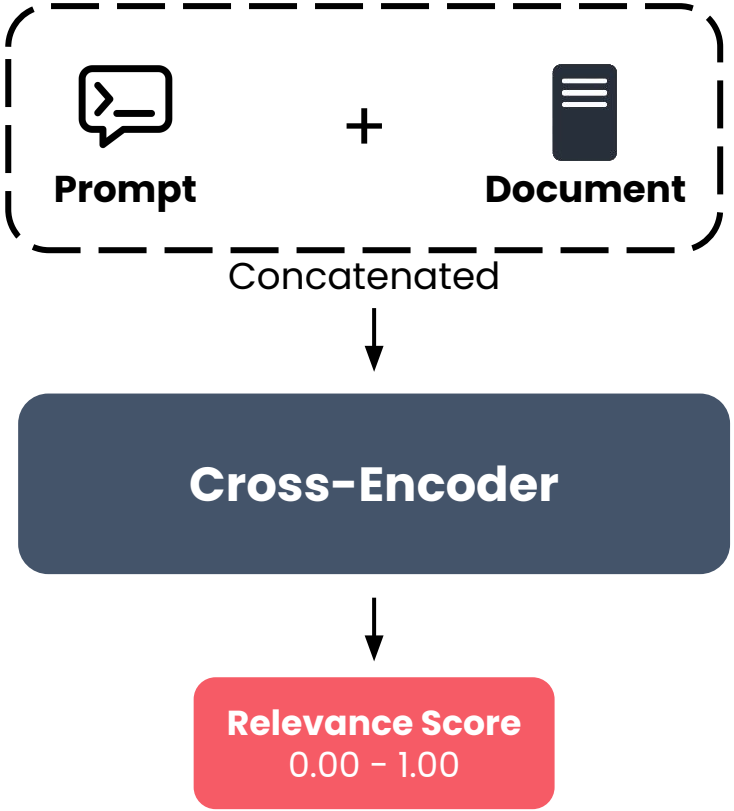
What is the capital of Canada?

Reranked Results

```
Ottawa is the capital of Canada

Toronto is in Canada

The Capital of France is in Paris

Canada is the maple syrup capital of
the world
```

You'll still return just 5–10 documents but reranking ensures they're far **more relevant**.

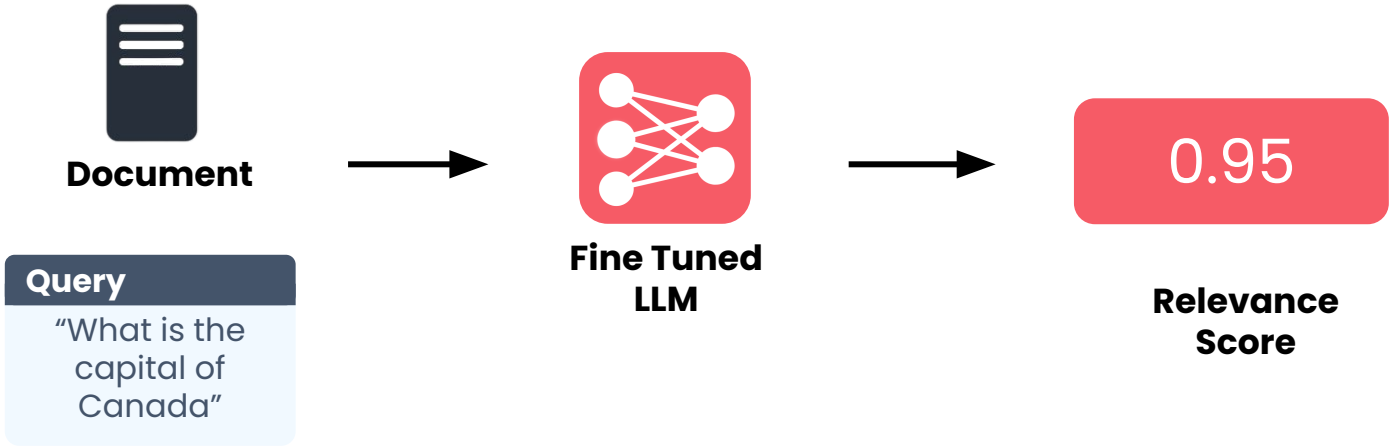




Cross-Encoder re-rankers

- Cross-encoders give better results than bi-encoders but are slower
- Using cross-encoders only after initial bi-encoder filtering makes the quality-time tradeoff feasible.
- Adds minor latency, but typically yields significantly better results.

LLM Based Scoring



LLM based scoring is powerful but costly. Like cross encoders, it is too slow for large scale retrieval and is best used for reranking after initial filtering.

DeepLearning.AI

Module 3 Conclusion

Information Retrieval in Production

Conclusion

Approximate Nearest Neighbors

ANN algorithms perform vector search significantly faster than brute force k-nearest neighbors

Vector Databases

Optimized to store high-dimensional vector data and perform approximate nearest neighbor searches

RAG Techniques

- Chunking
- Query Parsing
- Re-Ranking