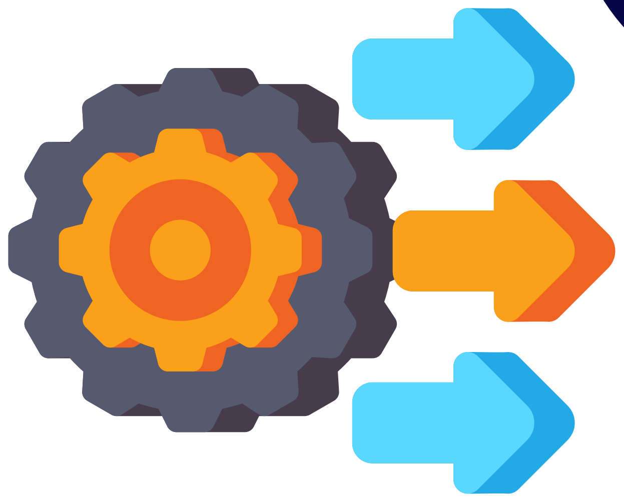
Learn and Grow

Generative AI LangChain

ASHIMA MALIK

ashimamalik58@gmail.com

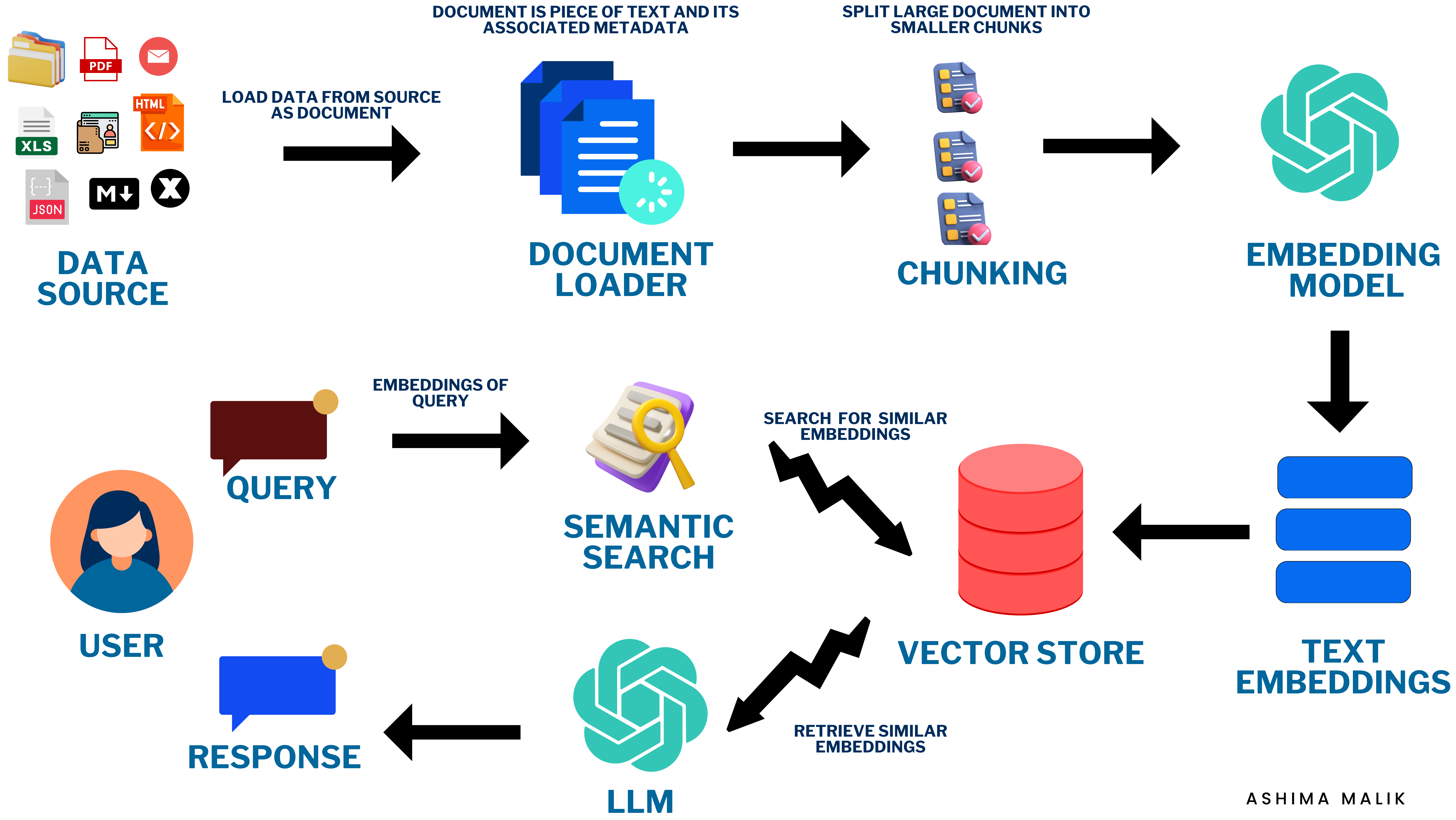
Retrieval



Limited by training data, many LLMs struggle with user-specific information. Retrieval-Augmented Generation (RAG) tackles this challenge by retrieving relevant external data and incorporating it into the generation process alongside the LLM's internal knowledge.

LLMs benefit from information beyond their training data.

RAG helps bridge this gap by retrieving relevant external data for the LLM to utilize during generation.


DATA SOURCE
PDF
HTML
XLS
JSON
LOAD DATA FROM SOURCE AS DOCUMENT
DOCUMENT IS PIECE OF TEXT AND ITS ASSOCIATED METADATA
DOCUMENT LOADER
SPLIT LARGE DOCUMENT INTO SMALLER CHUNKS
CHUNKING
EMBEDDING MODEL
TEXT EMBEDDINGS
VECTOR STORE
SEARCH FOR SIMILAR EMBEDDINGS
SEMANTIC SEARCH
EMBEDDINGS OF QUERY
QUERY
USER
RETRIEVE SIMILAR EMBEDDINGS
LLM
RESPONSE
ASHIMA MALIK

DOCUMENT LOADERS

ASHIMA MALIK

Document Loader (Import Statement)	Function
TextLoader (from langchain_community.document_loaders import TextLoader)	Loads text content from a file
CSVLoader (from langchain_community.document_loaders.csv_loader import CSVLoader)	Loads data from a CSV file into Document objects, with each row becoming a Document
JSONLoader (from langchain_community.document_loaders import JSONLoader)	Reads JSON data from a file where each line represents a JSON object, converting each object into a Document.
HTMLLoader (from langchain_community.document_loaders import UnstructuredHTMLLoader)	Fetches the text content and basic metadata from a website URL, creating a Document.
MarkdownLoader (from langchain_community.document_loaders import UnstructuredMarkdownLoader)	Loads the text content from a Markdown file, converting it into a Document.
PDFLoader (from langchain.document_loaders import PyPDFLoader)	Extracts text content from a PDF file, creating a Document (may require additional libraries).
DirectoryLoader (from langchain.document_loaders import Directory_Loader)	Loads all files within a specified directory as Documents, using appropriate loaders based on file extensions (.txt, .csv, etc.).

TEXT SPLITTERS

ASHIMA MALIK

Character Text Splitter

Character Splitting, in the context of text processing, refers to the process of dividing a string of text into individual characters. It's a fundamental operation used in various text manipulation tasks, data analysis, and machine learning applications.

Basic Function

Takes a string of text as input. Breaks down the string into its constituent characters. Each character becomes a separate element in a data structure, typically a list or an array.

Advantages

Simple and efficient for basic text processing tasks. Provides a granular level of control over text manipulation.

Disadvantages

Might not be the most efficient approach for splitting text into words or sentences, where higher-level splitting techniques like tokenization are preferred.

Recursive Character Text Splitting

"\n\n" - Double new line, or most commonly paragraph breaks
"\n" - New lines
" " - Spaces
"" - Characters

Basic Function

- Initial Split
- Recursion in Action
- Stopping Criterion

Advantages

- Preserves Meaning
- Flexibility
- Useful for Generative AI

Disadvantages

- Complexity
- Fine-Tuning

Semantic Chunking

LangChain utilizes techniques like sentence embedding to determine semantic similarity between different parts of the text. Sentence embedding involves converting sentences into numerical vectors that capture their meaning.

Basic Function

- Sentence Embedding
- Similarity Measurement
- Identifying Breaks
- Refined Chunking

Advantages

- Enhanced LLM Performance
- Flexibility

Disadvantages

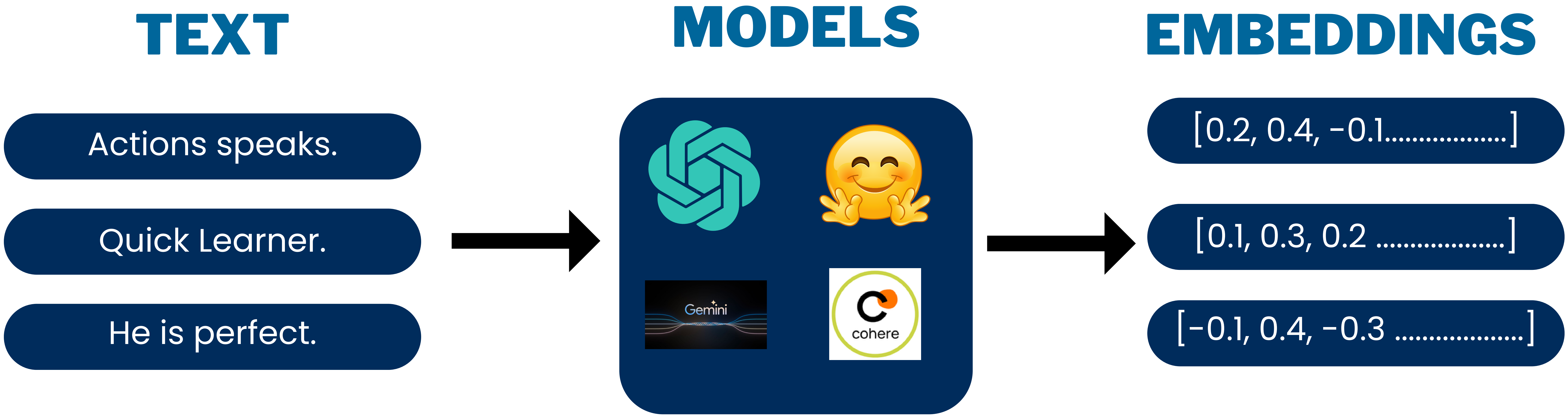
- Complexity
- Fine-Tuning

Breakpoints- Percentile, Standard Deviation, Interquartile

This chunker decides when to split sentences by checking for differences between them. If the difference is big enough, it splits them. We can figure out how big the difference needs to be in a few different ways.

TEXT EMBEDDING MODELS

ASHIMA MALIK


TEXT
Actions speaks.
Quick Learner.
He is perfect.
MODELS
Gemini
cohere
EMBEDDINGS
[0.2, 0.4, -0.1..................]
[0.1, 0.3, 0.2 ....]
[-0.1, 0.4, -0.3 ....]

VECTOR STORES & RETRIEVERS

ASHIMA MALIK

WORD EMBEDDINGS

Step1: Collect data
It was the best of times,
it was the worst of times,
it was the age of wisdom,
it was the age of foolishness!

Step2: Design the vocabulary
Now we can make a list of all of the words in our model vocabulary.
The unique words here (ignoring case and punctuation) are:

“it”
“was”
“the”
“best”
“of”
“times”
“worst”
“age”
“wisdom”
“foolishness”

This is a vocabulary of 10 words from a corpus containing 24 words.

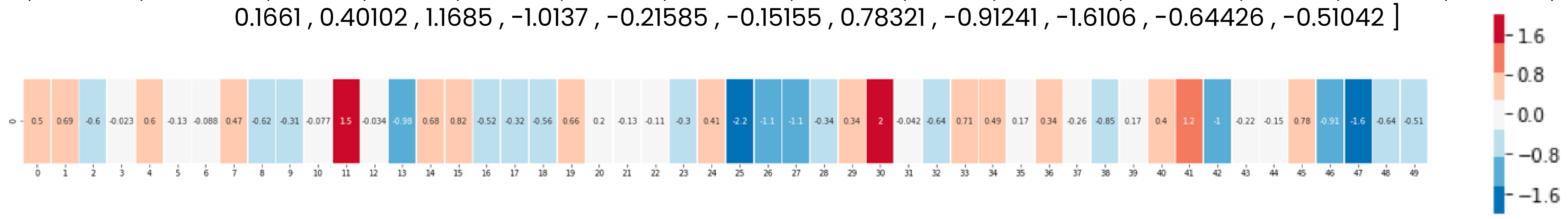
Step3: Create document vectors

	it	was	the	best	of	time	worst	age	wisdom	foolishness
"it was the worst of times" =	1	1	1	0	1	1	1	0	0	0
"it was the age of wisdom" =	1	1	1	0	1	0	0	1	1	0
"it was the age of foolishness" =	1	1	1	0	1	0	0	1	0	1

WORD EMBEDDINGS

king

[0.50451 , 0.68607 , -0.59517 , -0.022801, 0.60046 , -0.13498 , -0.08813 , 0.47377 , -0.61798 , -0.31012 , -0.076666, 1.493 , -0.034189, -0.98173 , 0.68229 , 0.81722 , -0.51874 , -0.31503 , -0.55809 , 0.66421 , 0.1961 , -0.13495 , -0.11476 , -0.30344 , 0.41177 , -2.223 , -1.0756 , -1.0783 , -0.34354 , 0.33505 , 1.9927 , -0.04234 , -0.64319 , 0.71125 , 0.49159 , 0.16754 , 0.34344 , -0.25663 , -0.8523 , 0.1661 , 0.40102 , 1.1685 , -1.0137 , -0.21585 , -0.15155 , 0.78321 , -0.91241 , -1.6106 , -0.64426 , -0.51042]



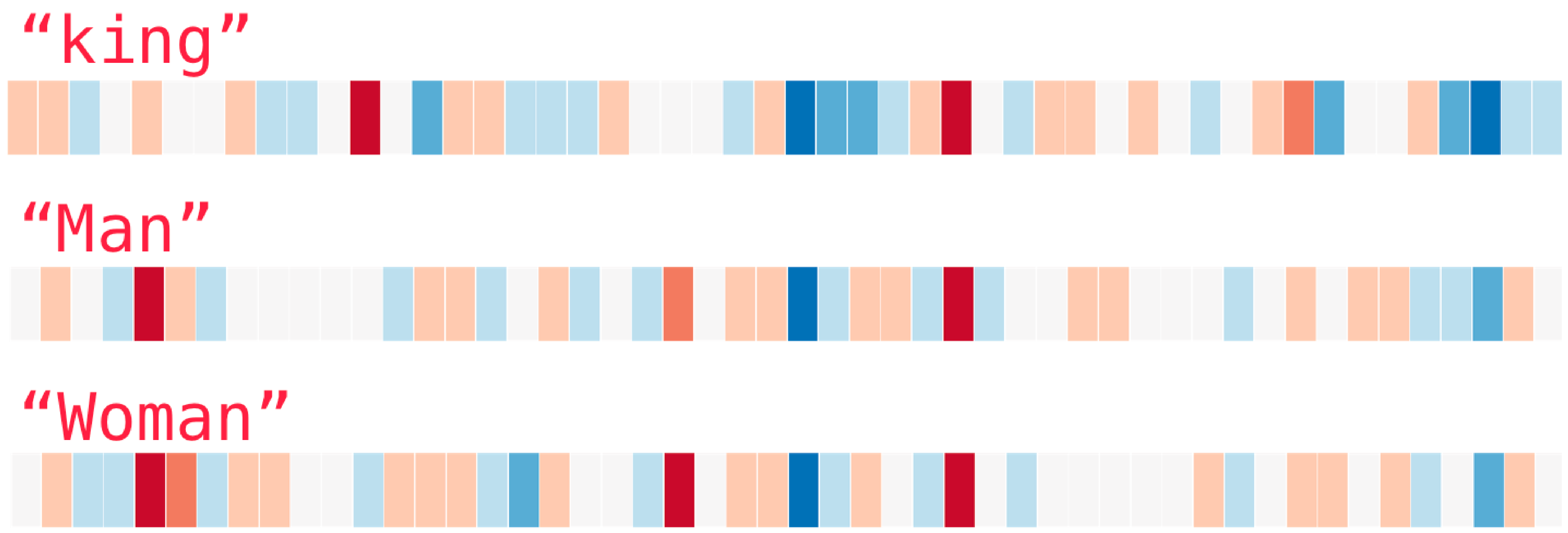


Image Source: http://jalammar.github.io/illustrated-word2vec/

ASHIMA MALIK

WORD EMBEDDINGS

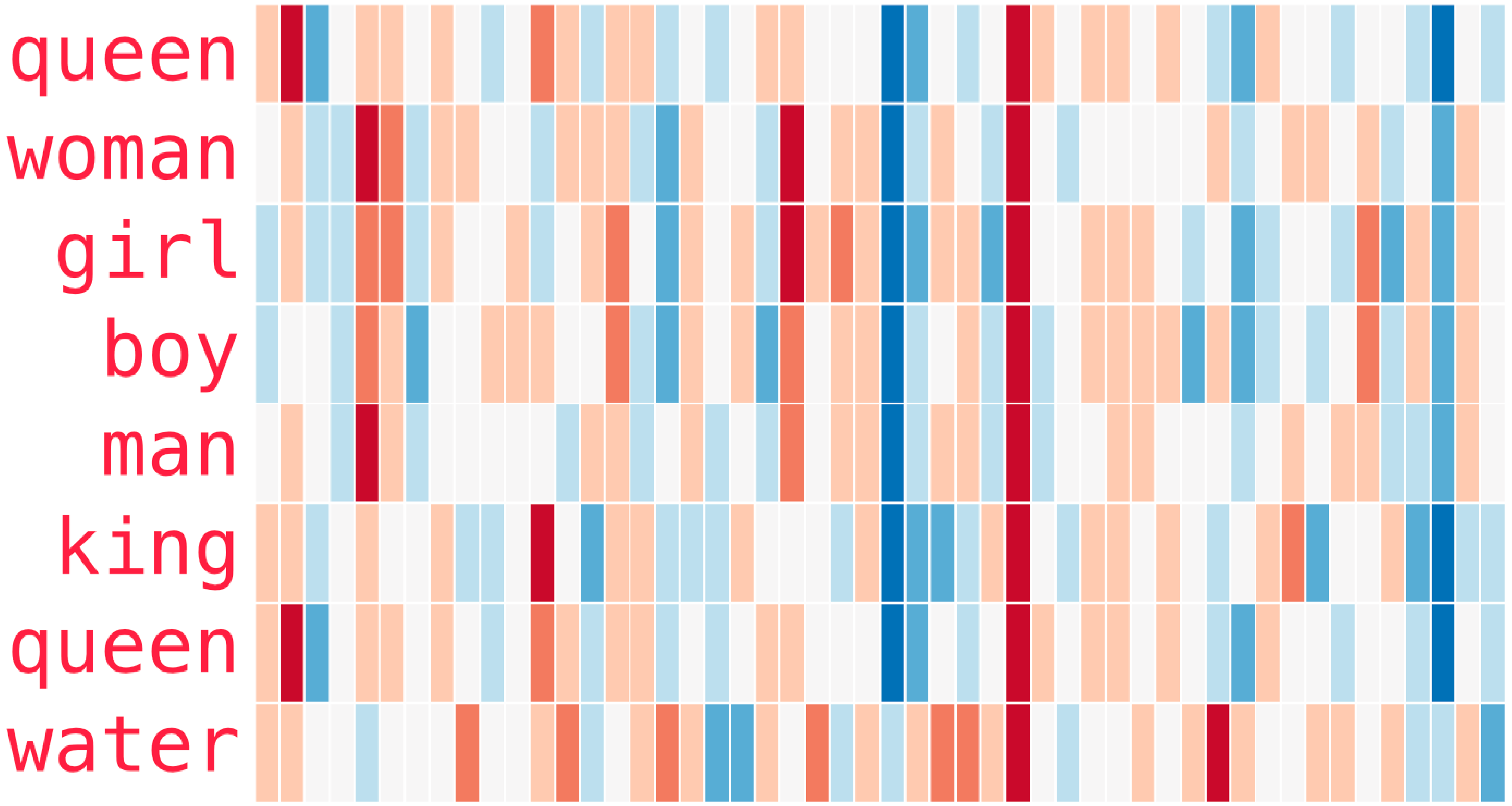
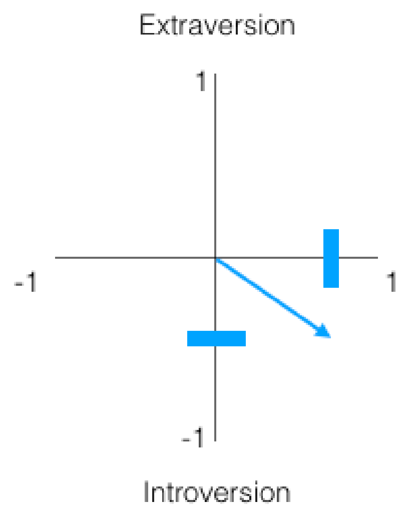
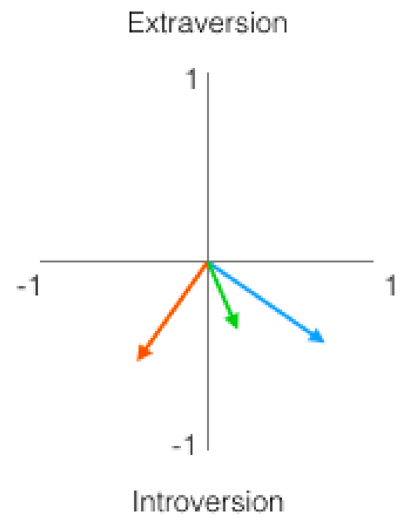


Image Source: http://jalammar.github.io/illustrated-word2vec/

WORD EMBEDDINGS



	Trait #1	Trait #2			
Jay	-0.4	0.8			



	Trait #1	Trait #2			
Jay	-0.4	0.8			
Person #1	-0.3	0.2			
Person #2	-0.5	-0.4			

```
                    Jay         Person #1
cosine_similarity( -0.4  0.8 , -0.3  0.2 ) = 0.87 ✓

                    Jay         Person #2
cosine_similarity( -0.4  0.8 , -0.5  -0.4 ) = −0.20
```

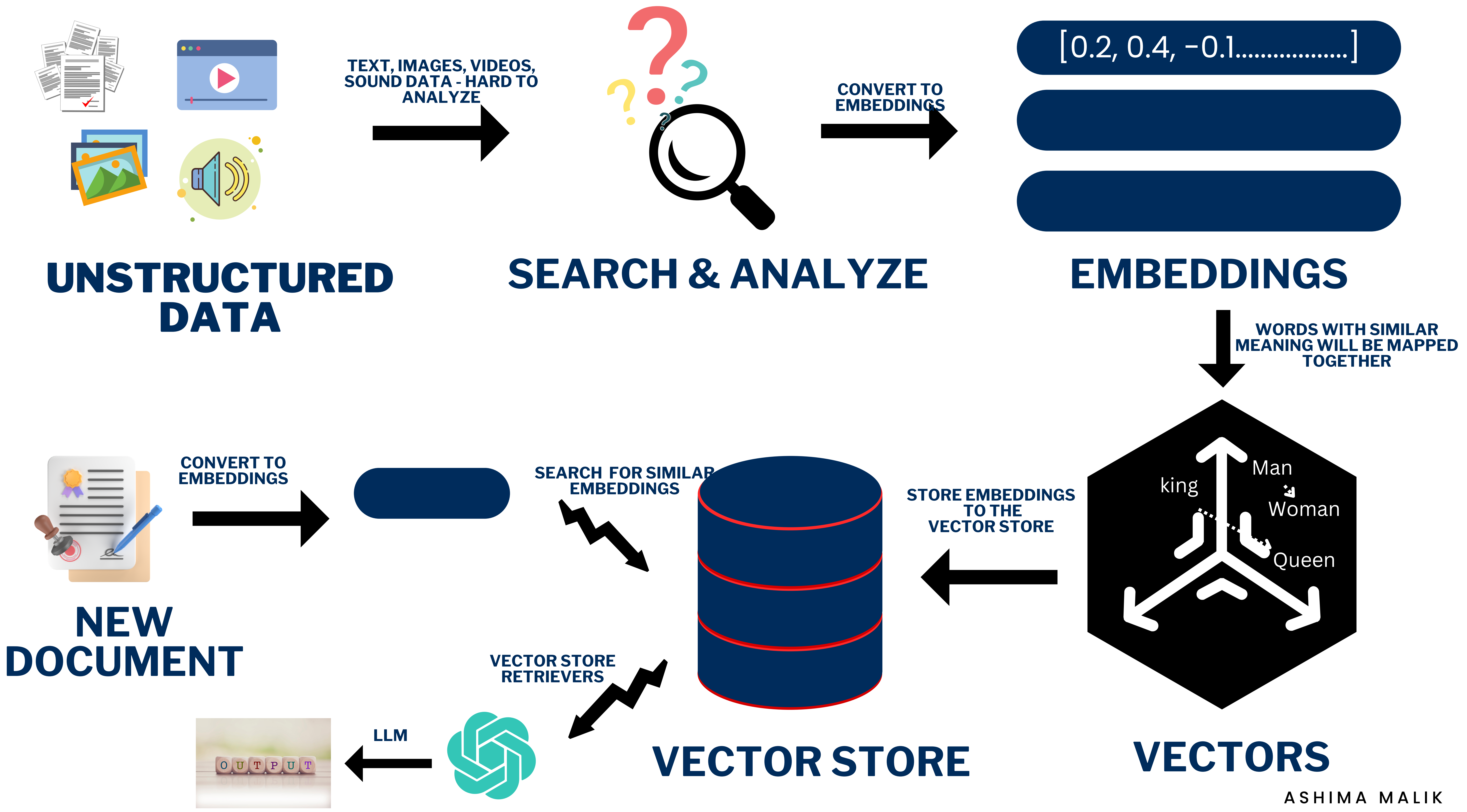
	Trait #1	Trait #2	Trait #3	Trait #4	Trait #5
Jay	-0.4	0.8	0.5	-0.2	0.3
Person #1	-0.3	0.2	0.3	-0.4	0.9
Person #2	-0.5	-0.4	-0.2	0.7	-0.1

```
                    Jay                           Person #1
cosine_similarity( -0.4  0.8  0.5  -0.2  0.3 , -0.3  0.2  0.3  -0.4  0.9 ) = 0.66 ✓

                    Jay                           Person #2
cosine_similarity( -0.4  0.8  0.5  -0.2  0.3 , -0.5  -0.4  -0.2  0.7  -0.1 ) = −0.37
```

Image Source: http://jalammar.github.io/illustrated-word2vec/

ASHIMA MALIK


TEXT, IMAGES, VIDEOS, SOUND DATA - HARD TO ANALYZE
CONVERT TO EMBEDDINGS
[0.2, 0.4, -0.1.................]
UNSTRUCTURED DATA
SEARCH & ANALYZE
EMBEDDINGS
WORDS WITH SIMILAR MEANING WILL BE MAPPED TOGETHER
Man
king
Woman
Queen
STORE EMBEDDINGS TO THE VECTOR STORE
CONVERT TO EMBEDDINGS
SEARCH FOR SIMILAR EMBEDDINGS
NEW DOCUMENT
VECTOR STORE RETRIEVERS
LLM
OUTPUT
VECTOR STORE
VECTORS
ASHIMA MALIK

Name	Index Type	Uses an LLM	When to Use	Description
Vectorstore	Vectorstore	No	If you are just getting started and looking for something quick and easy.	This is the simplest method and the one that is easiest to get started with. It involves creating embeddings for each piece of text.
ParentDocument	Vectorstore + Document Store	No	If your pages have lots of smaller pieces of distinct information that are best indexed by themselves, but best retrieved all together.	This involves indexing multiple chunks for each document. Then you find the chunks that are most similar in embedding space, but you retrieve the whole parent document and return that (rather than individual chunks).
Multi Vector	Vectorstore + Document Store	Sometimes	If you are able to extract information from documents that you think is more relevant to index than the text itself.	This involves creating multiple vectors for each document. Each vector could be created in a myriad of ways - examples include summaries of the text and hypothetical questions.
Self Query	Vectorstore	Yes	If users are asking questions that are better answered by fetching documents based on metadata rather than similarity with the text.	This uses an LLM to transform user input into two things: (1) a string to look up semantically, (2) a metadata filter to go along with it. This is useful because oftentimes questions are about the METADATA of documents (not the content itself).
Contextual Compression	Any	Sometimes	If you are finding that your retrieved documents contain too much irrelevant information and are distracting the LLM.	This puts a post-processing step on top of another retriever and extracts only the most relevant information from retrieved documents. This can be done with embeddings or an LLM.
Time-Weighted Vectorstore	Vectorstore	No	If you have timestamps associated with your documents, and you want to retrieve the most recent ones	This fetches documents based on a combination of semantic similarity (as in normal vector retrieval) and recency (looking at timestamps of indexed documents)
Multi-Query Retriever	Any	Yes	If users are asking questions that are complex and require multiple pieces of distinct information to respond	This uses an LLM to generate multiple queries from the original one. This is useful when the original query needs pieces of information about multiple topics to be properly answered. By generating multiple queries, we can then fetch documents for each of them.
Ensemble	Any	No	If you have multiple retrieval methods and want to try combining them.	This fetches documents from multiple retrievers and then combines them.
Long-Context Reorder	Any	No	If you are working with a long-context model and noticing that it's not paying attention to information in the middle of retrieved documents.	This fetches documents from an underlying retriever, and then reorders them so that the most similar are near the beginning and end.

ASHIMA MALIK

Indexing

The indexing API enables you to import and maintain documents from various sources within a vector store efficiently.

RecordManager keeps track of document writes into the vector store.

Its primary functions include preventing duplicate content from being written to the vector store, avoiding rewriting content that remains unchanged, and preventing the recomputation of embeddings for content that has not been altered.