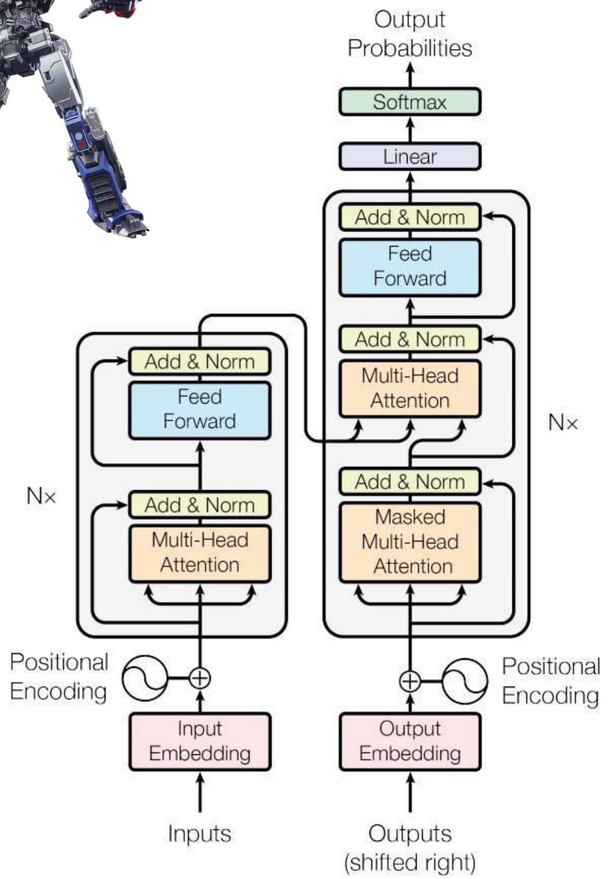
Large Language Model

Yu-Chung Wang

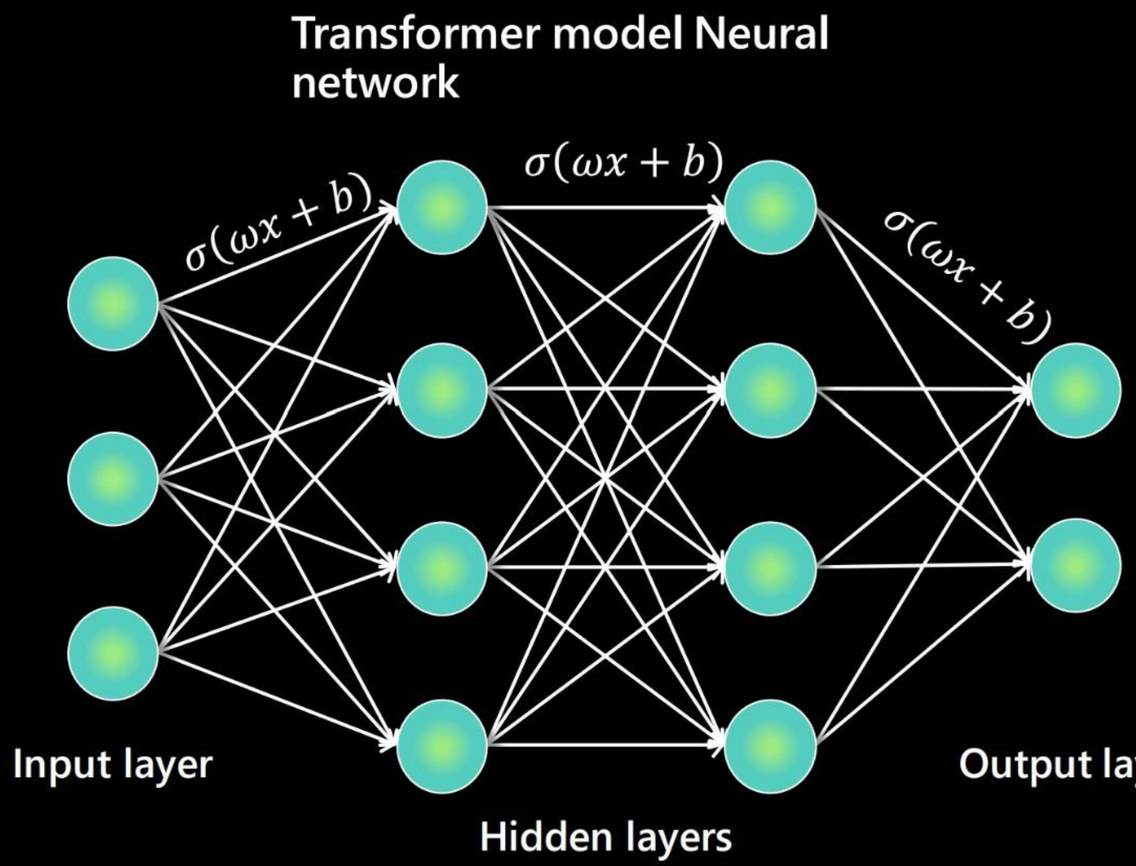
Focus on Language Models

You said Large Language Model ?

- Generative deep learning models for understanding and generating text, images and other types
- A special kind : Transformers
  - “*Attention is All you Need”*, Vaswani et al. 2017 (https://arxiv.org/abs/1706.03762)
- Transformers analyse chunks of data, called “tokens” and learn to predict the next token in a sequence
- Prediction is a probability
- Model that can generalize : one single model to address several use cases



How large are they?



Function: weight * input plus bias

BERT Large - 2018
345M

GPT2 - 2019
1.5B

GPT3 - 2020
175B

Turing Megatron NLG 2021
530B

GPT4 – 2023
1.4T (estimated)

Build the model - Training

What it’s like ?

- Foundational models
- Datasets

  LLM are trained using techniques that requires huge text-based datasets, e.g.

  “The Pile” : +880 Gb (Wikipedia, Youtube st, Github, …)

  “RedPajama”: +5Tb (wikipedia, StackExchange, ArXiv, …)

  Choosing and curating datasets for training is the secret sauce !

- Computing Power

  Transformer-based model have limitations: quadratic-complexity of attention mechanism

  Computationally intensive for long sequences

Use the model - Inference

Common patterns

- **Context**

  The size of input data given to the model : size is limited !

- **Prompt**

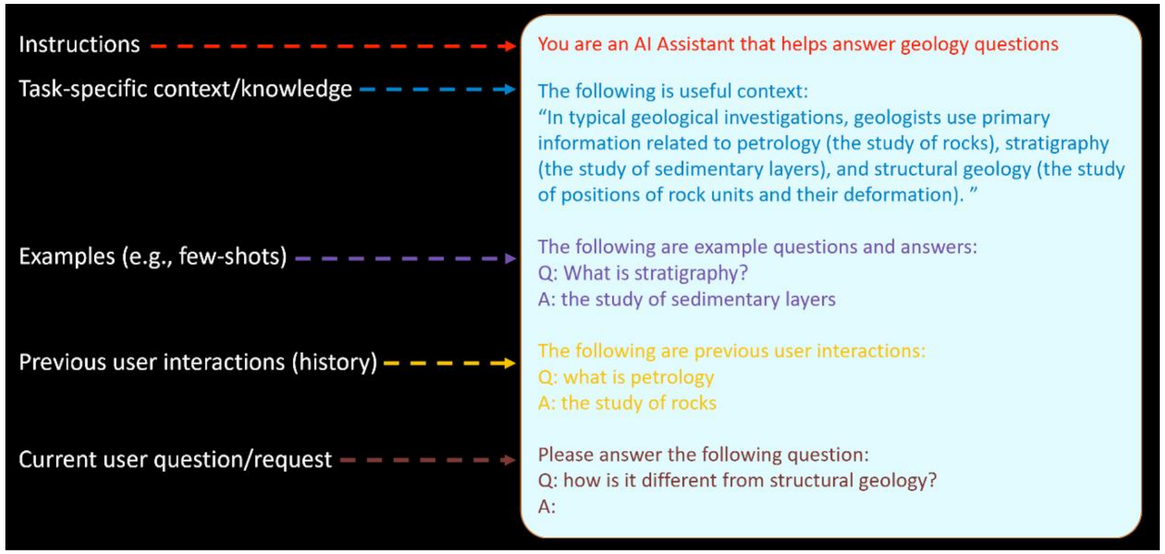
  The question / the task, enriched with ‘pre-prompt’

- **Zero-shot / Few-shot, …**

  To give or not samples of answers expected

- **Temperature**

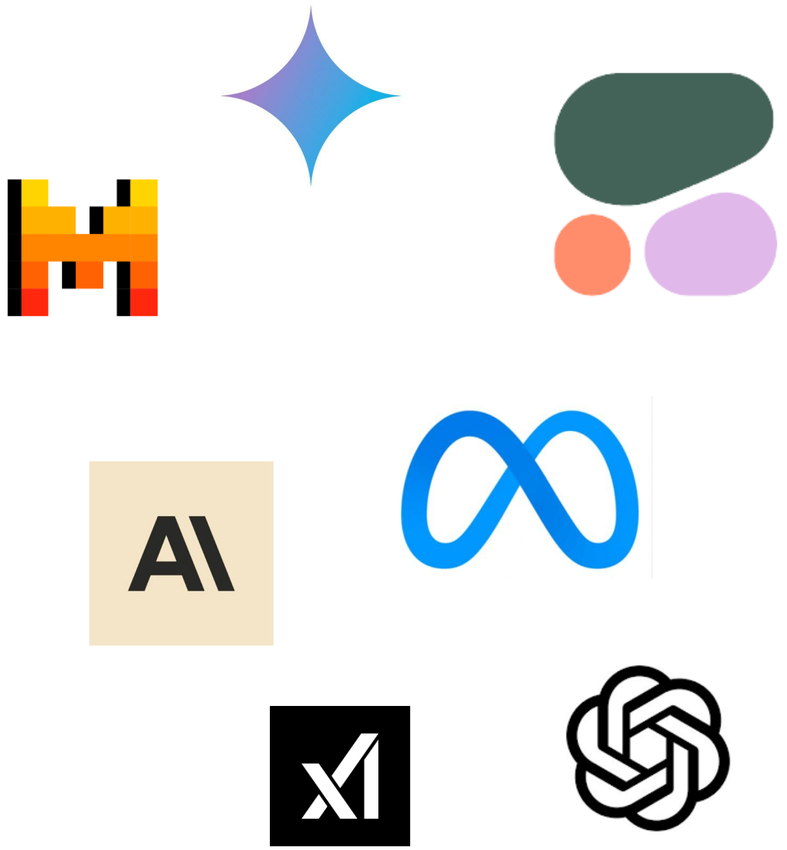
  How much the model is imaginative



Which Model ?

Criteria to take in account for a use case

- Open Source vs Commercial
- Best of breed
- Versioning & lifecycle
- Cost efficiency vs Overkill -> Size
- Accuracy

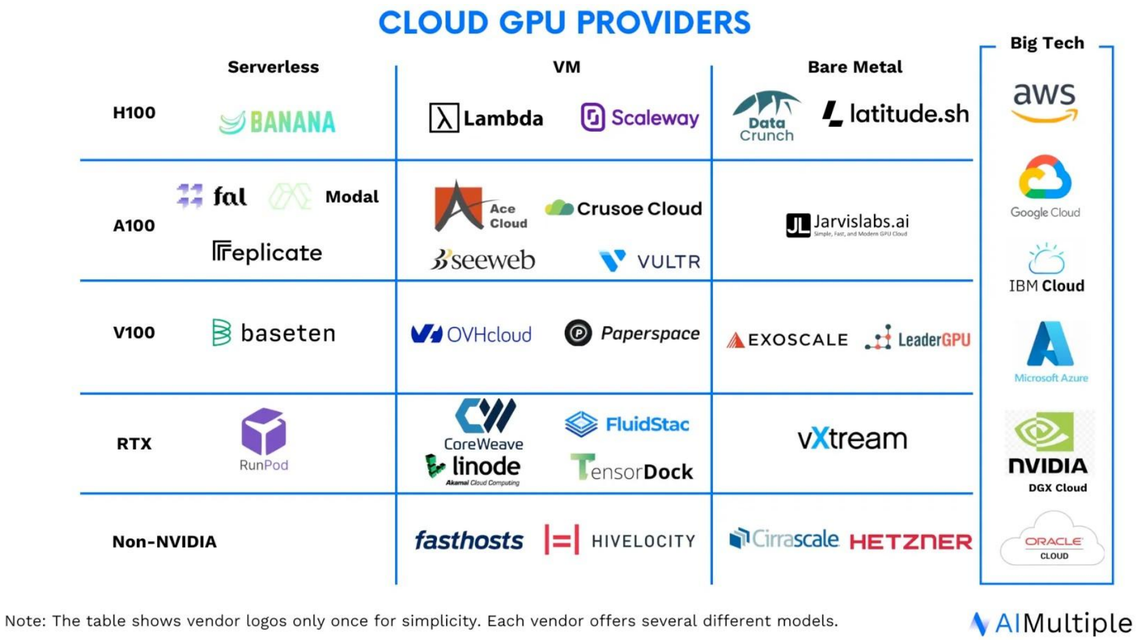


Infrastructure

At the heart of the machine

- On Premises
  - Compute: GPUs choice / VRAM size / Model quantization
    - NVIDIA T4 = 16Gb / 1100$
    - NVIDIA A100 = 80Gb / 8000$
  - Scalability : concurrent users, context size
  - Online vs batch

- On Cloud
  - Which one ? Cost, diversity and availability
  - Pricing model: 1M token comes very fast ! 1 word ~ 4 tokens
  - Sovereignty, data privacy



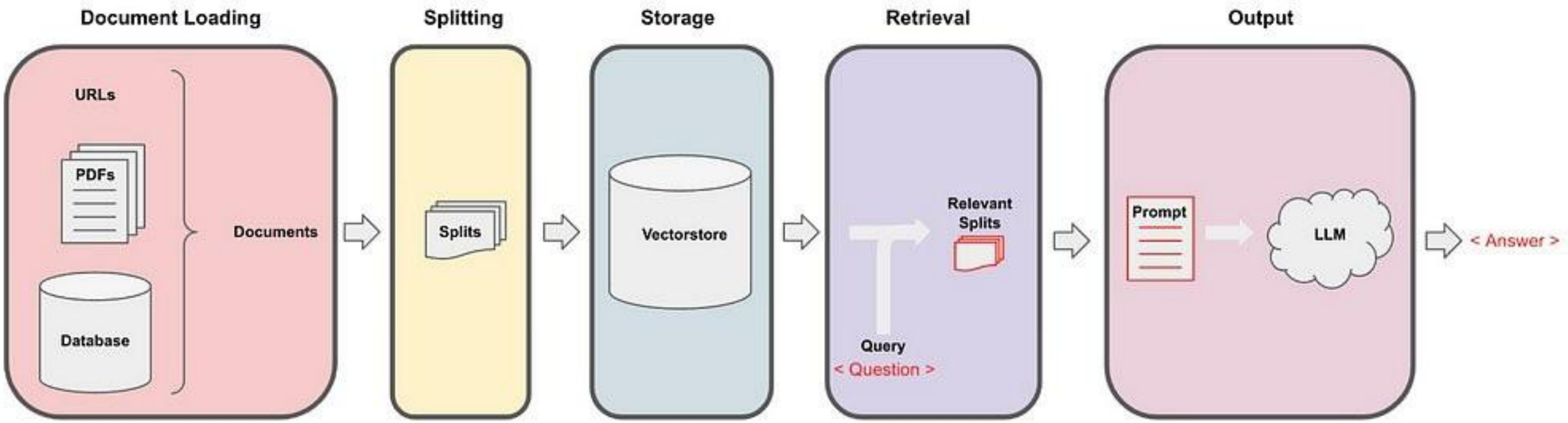
Real-world usage

Very common use case = “Retrival Augmented Generation”

Aka your search engine 2.0

RAG

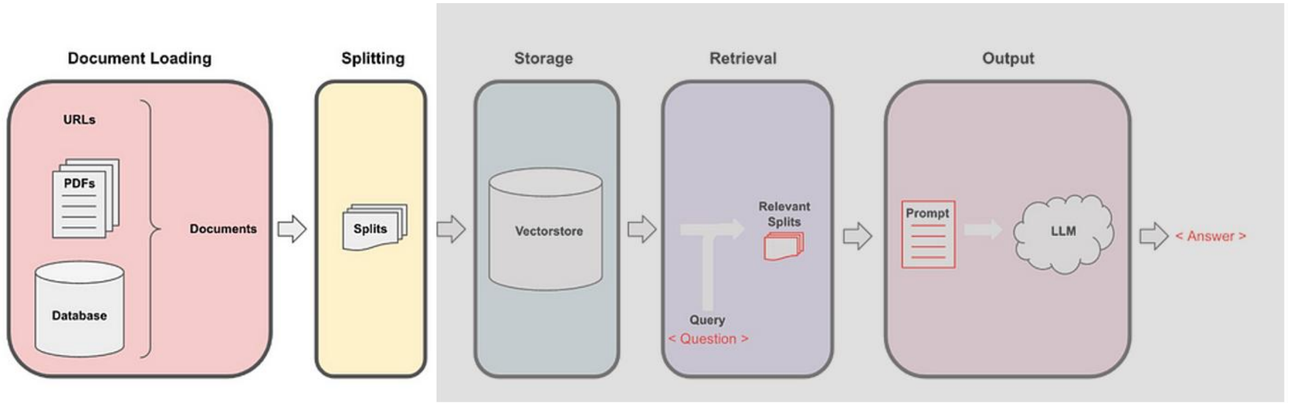
Search & Summarize In 4 Steps



RAG

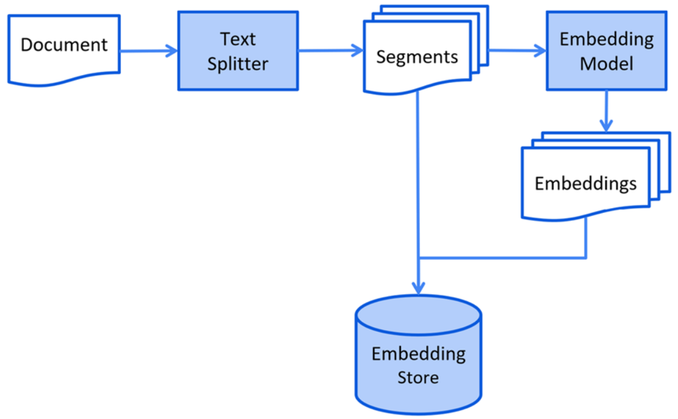
Step 1 - Document loading

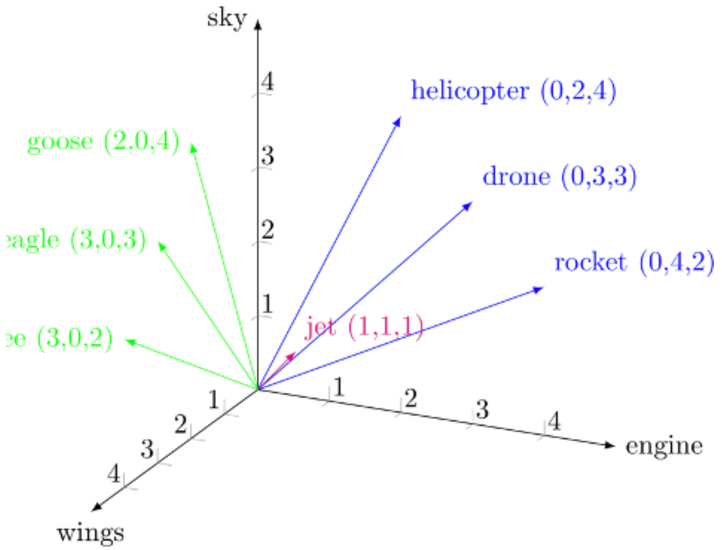
- Documents are loaded from data connectors
- They are split into chunks



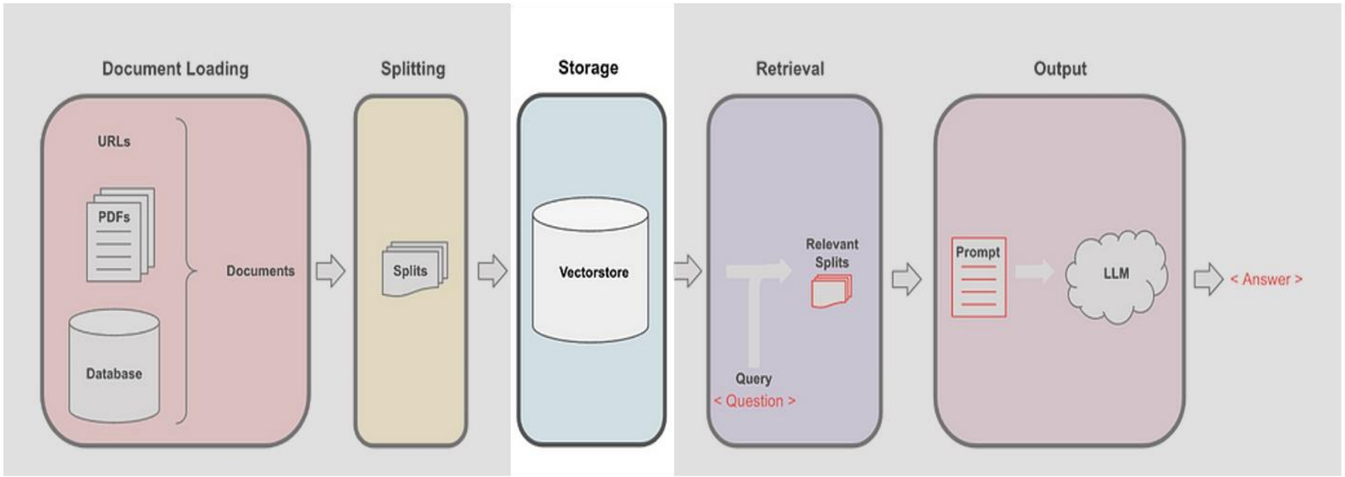
RAG

Step 2 - Embeddings



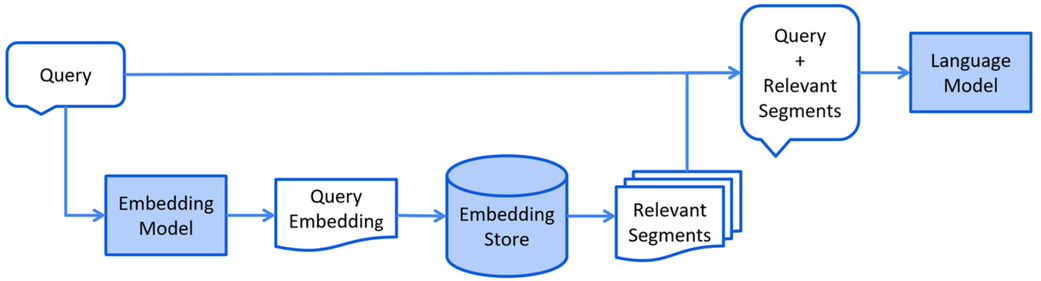


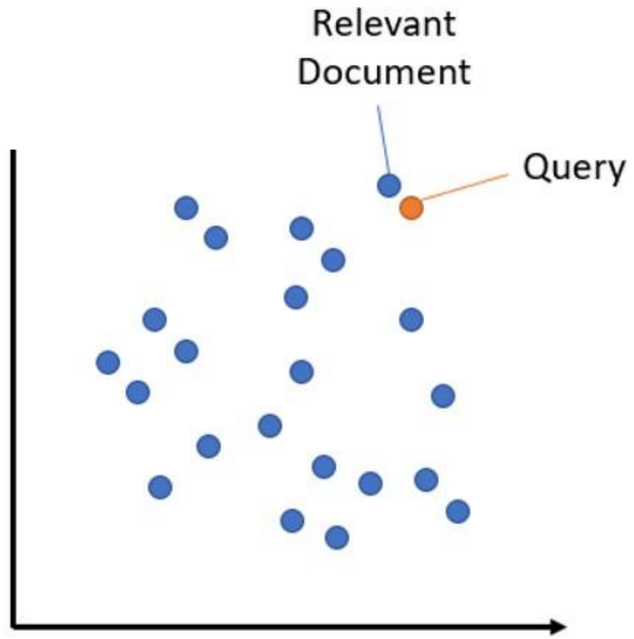
- Chunks are 'transformed' in vectors (numbers)
  - ✓ It's the process of word embedding, using a pre-trained model
  - ✓ hundreds (even thousands !) of dimensions are required to represent the space of all words
- Vectors are stored in a dedicated database (a vector database)



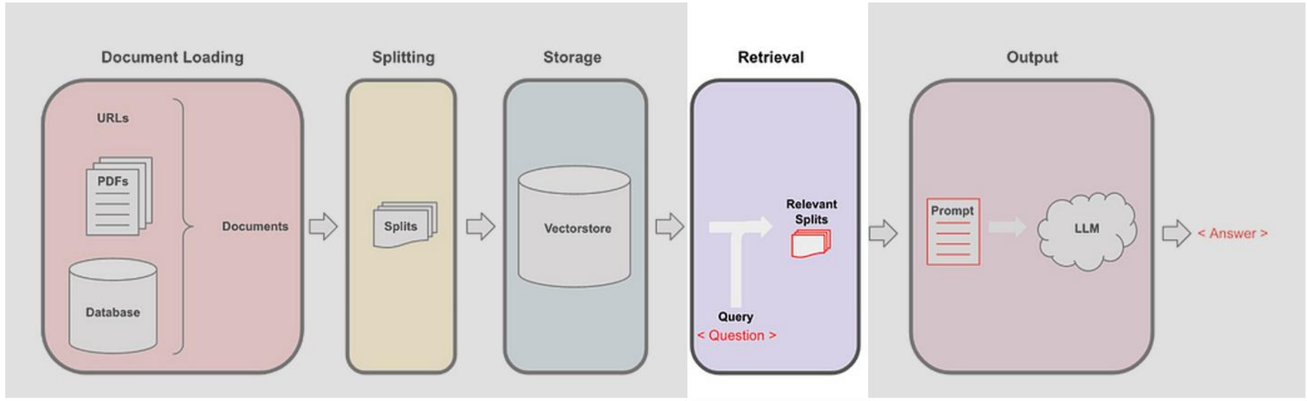
RAG

Step 3 - Retrieval





- Previous steps were preparatory work, now comes the live part
- Question is vectorized as well, used as an input for similarity search
- Most relevant chunks are retrieved, i.e. vectors coordinates are close together

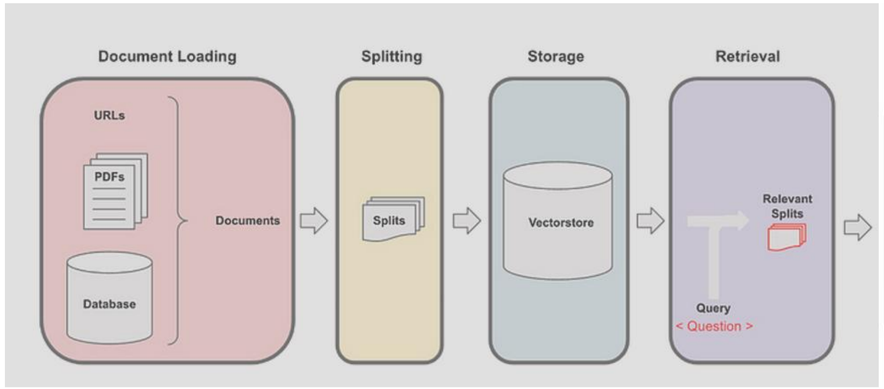


RAG

Step 4 - Generation

- Retrieved chunks are used to feed the LLM prompt context
- Question is added to the prompt
- LLM reads the prompt and generates a natural language answer
- During this inference time, the model requires a lot of GPU power !

```
prompt =
"""\

Context information is below.

--------------------- {context_str} ---------------------
Given the context information and not prior knowledge, answer the query
asking about citations over different topics. Please provide your answer in
the form of a structured JSON format containing a list of authors as the
citations. Some examples are given below.

{few_shot_examples}

Query: {query_str}

Answer: \

"""
```



RAG engineering

Lots of moving part to reach performance !

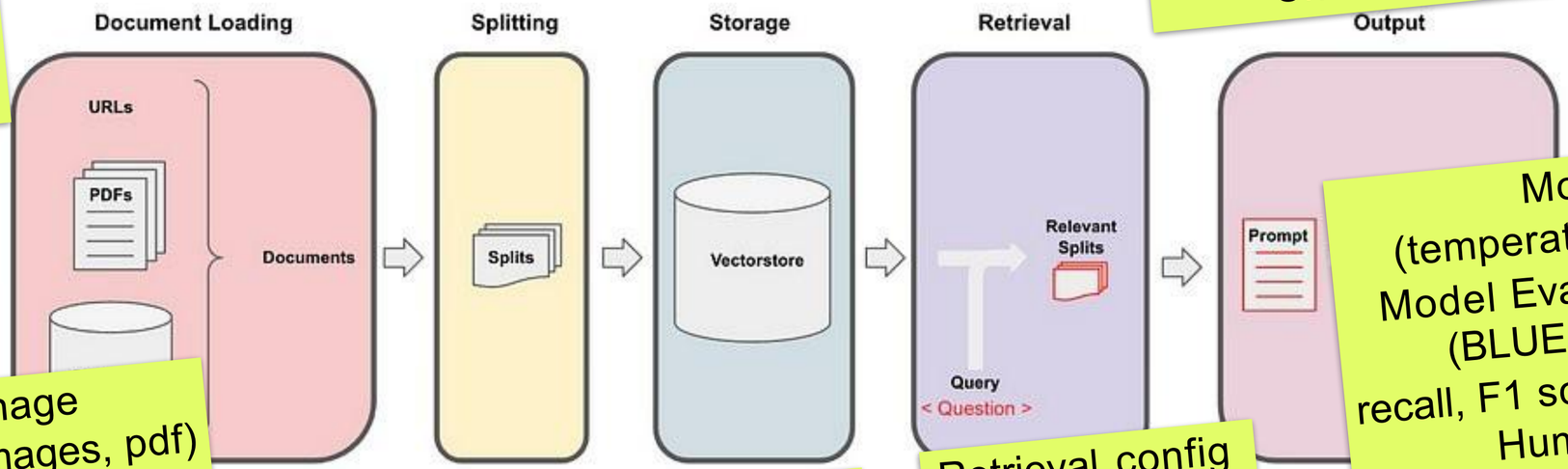
Flow / Batch
Data Policy
Deduplication

Chunking strategy

Vector DB Choice
Cloud / Local
Vectors dimensions
& reduction

RAG techniques
(Corrective, Self-reflective
Rag-Fusion, HyDE)
Chat memory

UI-Integration
LLMOPS / MLOPS
Cost Efficiency



Data cleanage
Attachments (images, pdf)
PII / Anonymization
Data policy / criticity

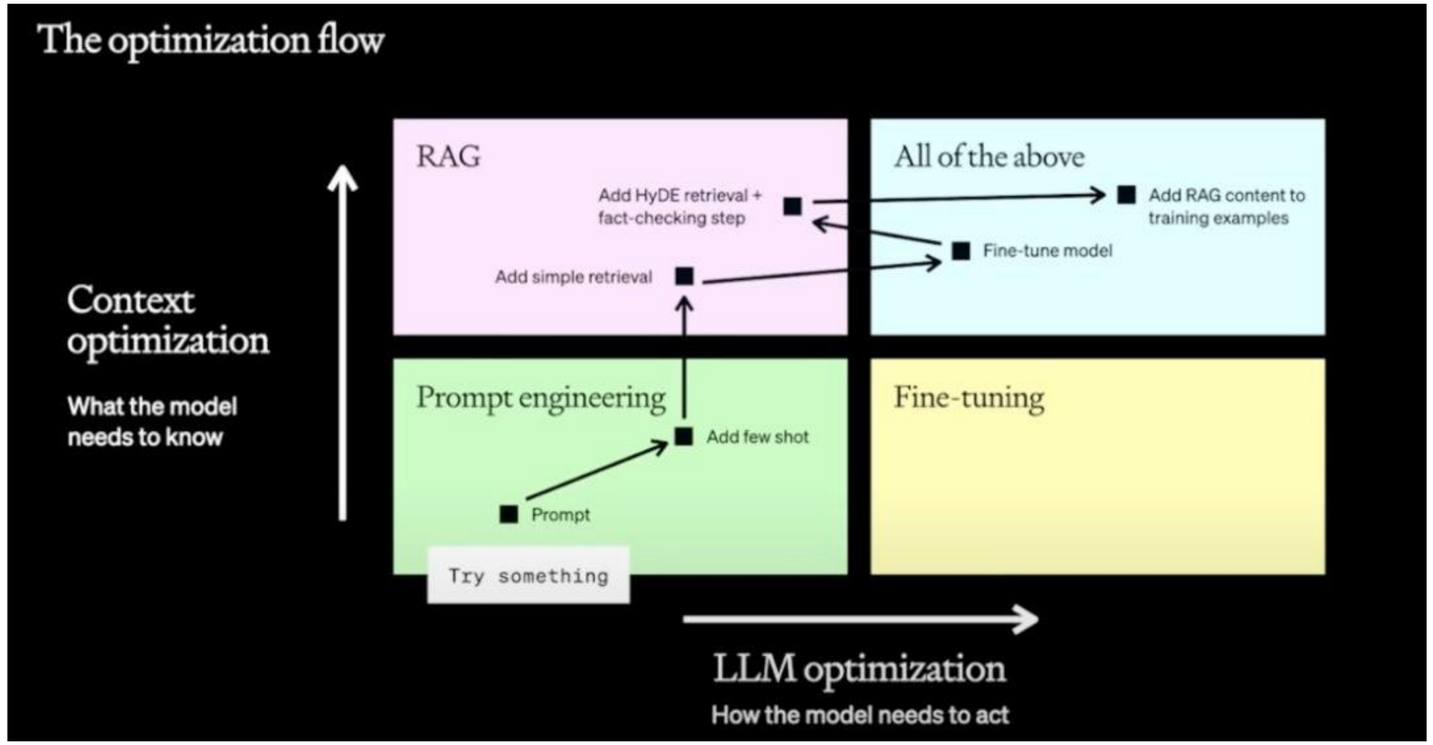
Embedding Model
Size
Language
Tokenizer

Retrieval config
(top_k, similarity)
Re-ranking
MMR score

Model config
(temperature, top_k, top_p)
Model Evaluation / derivation
(BLUE/RED, precision,
recall, F1 score, Ragas, truelens,
Human Feedback)
Prompt eng.
Guard rails
(Hallucinations, NSFW, …)
model compare / VertexSxS
Performance (TTFT, TPS, …)
PII / Anon (again)

Fine Tuning ?

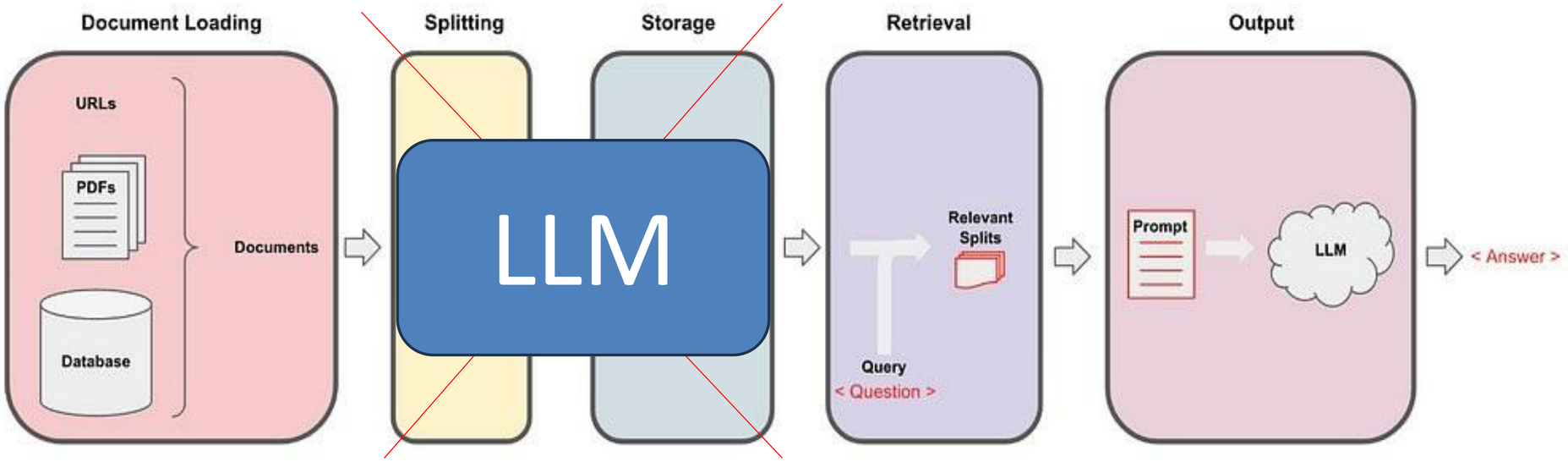
OpenAI’s strategy



How to improve and innovate?

Use the knowledge received in this course

For example:



Demo time