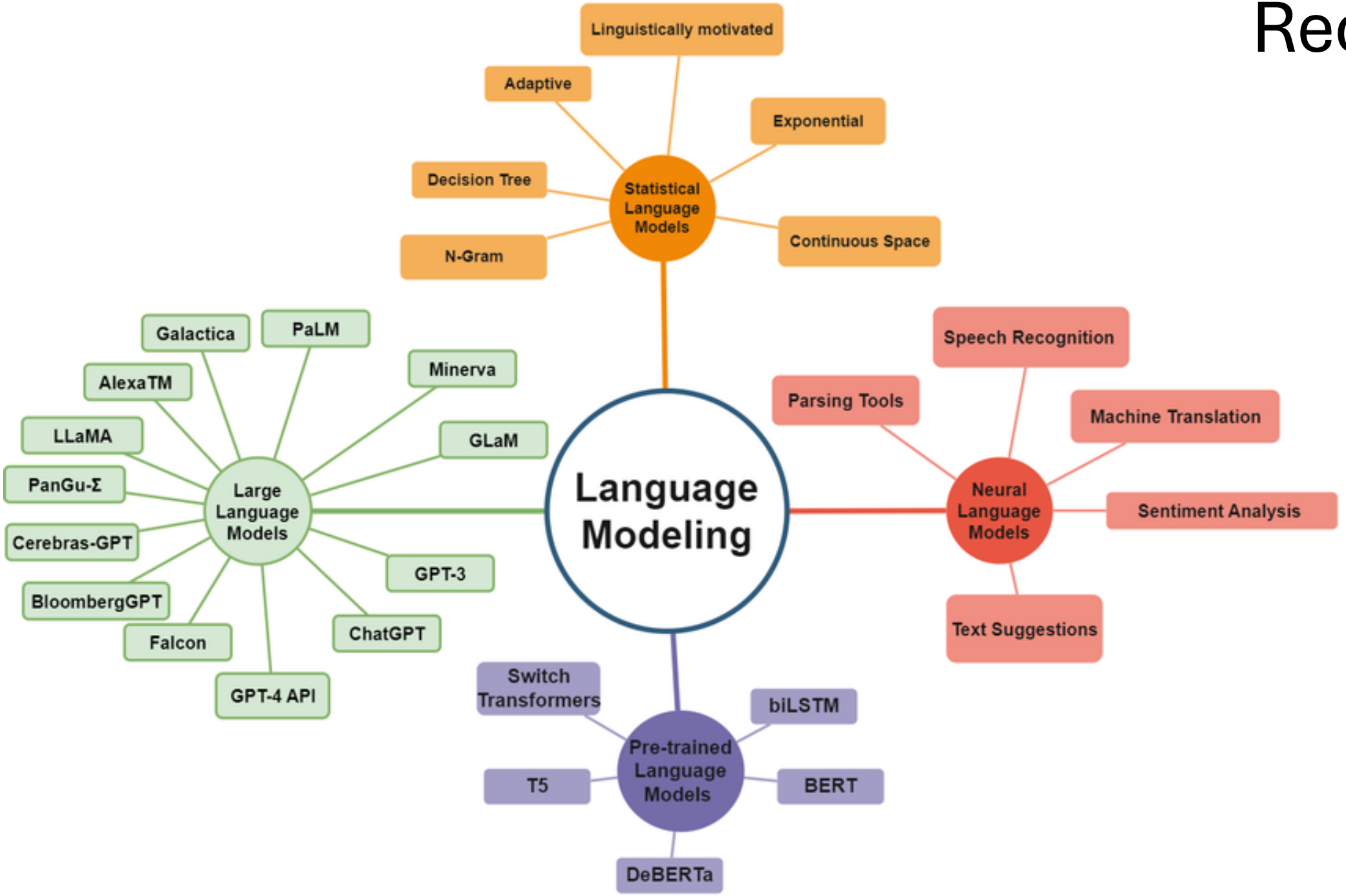
IR with LLMs and RAGs

- Contextual LLMs – Recap
- Non-Textual LLMs
- IR with LLMs
- IR with LLMs – Problems
- Solutions - Fine Tuning
- Solutions - RAGs
- Solutions – Prompting
- Agentic Rags

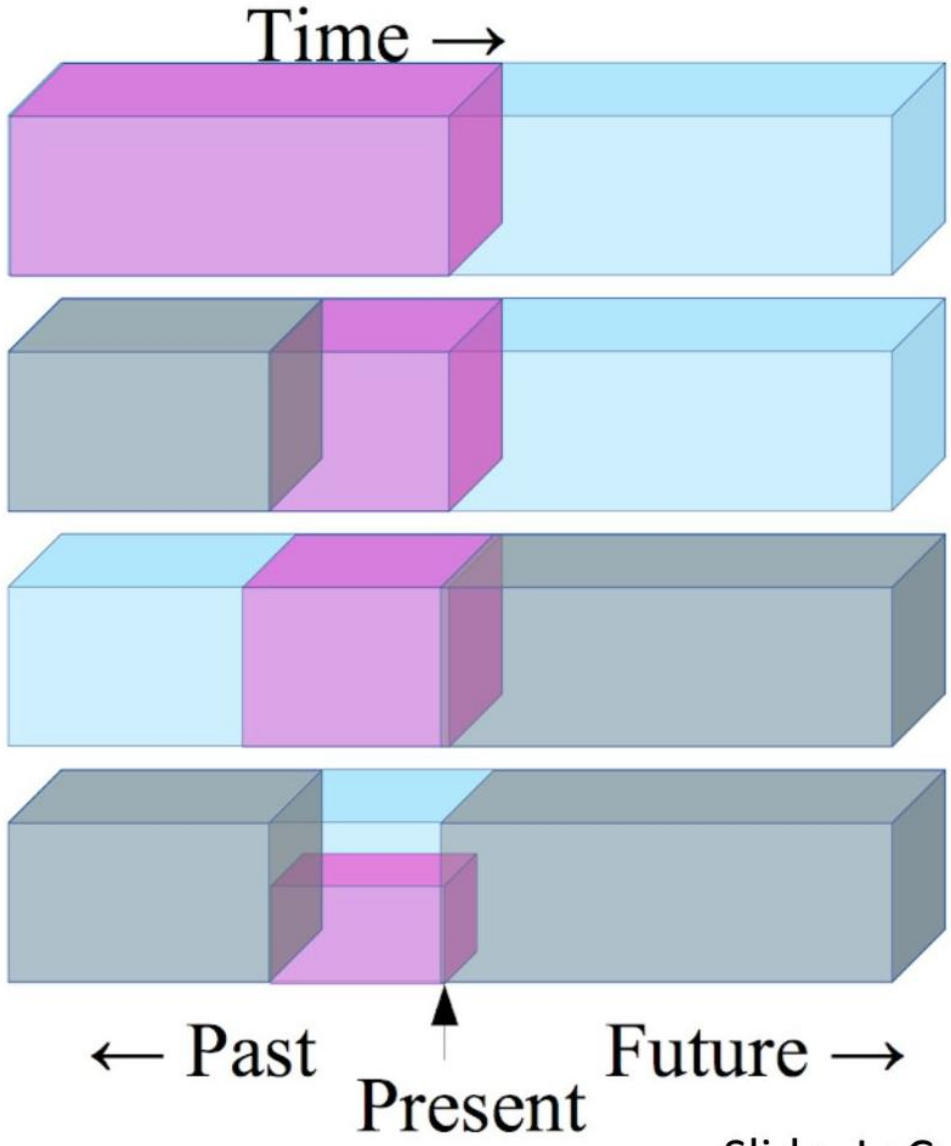
Recap


Language Modeling
Statistical Language Models
Linguistically motivated
Adaptive
Exponential
Decision Tree
Continuous Space
N-Gram
Large Language Models
Galactica
PaLM
Minerva
AlexaTM
GLaM
LLaMA
PanGu-Σ
Cerebras-GPT
GPT-3
BloombergGPT
ChatGPT
Falcon
GPT-4 API
Neural Language Models
Speech Recognition
Parsing Tools
Machine Translation
Sentiment Analysis
Text Suggestions
Pre-trained Language Models
Switch Transformers
biLSTM
T5
BERT
DeBERTa

Recap

Predicting neighbouring context

- **Predict any part of the input from any other part.**
- **Predict the future from the past.**
- **Predict the future from the recent past.**
- **Predict the past from the present.**
- **Predict the top from the bottom.**
- **Predict the occluded from the visible**
- **Pretend there is a part of the input you don't know and predict that.**



Slide: LeCun

Recap

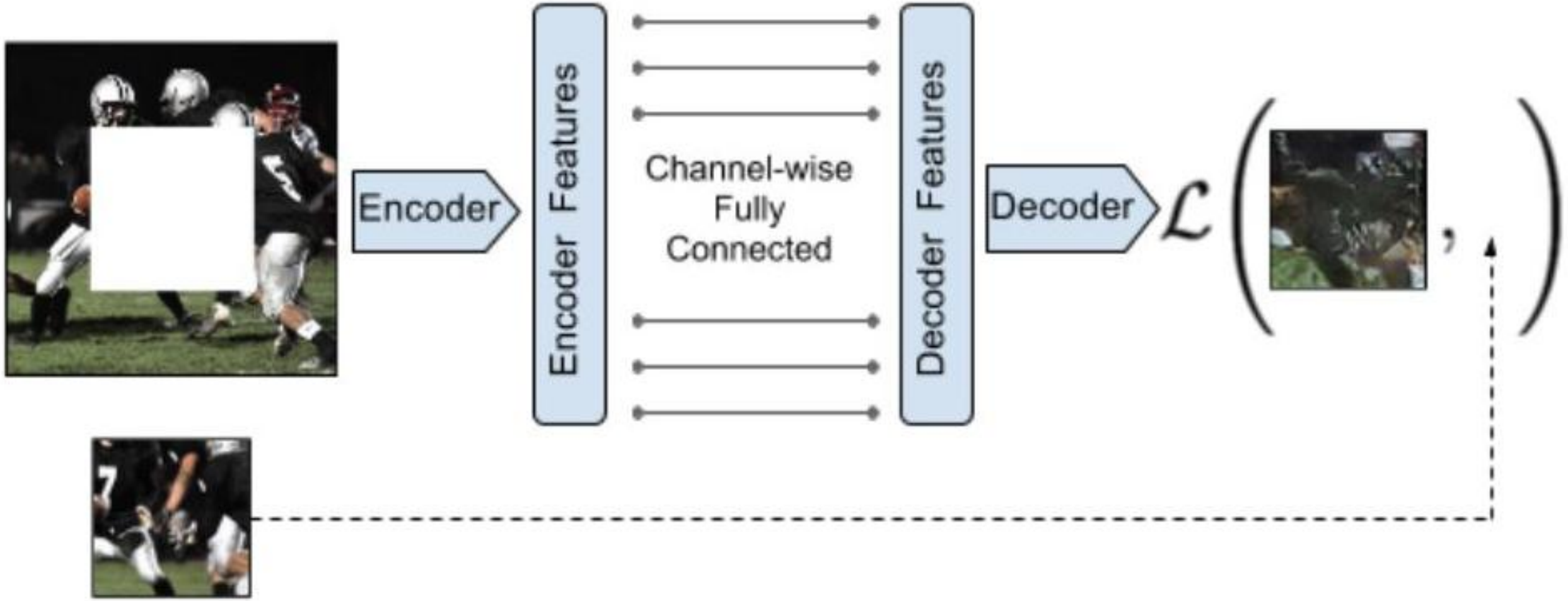
- **"Pure" Reinforcement Learning (cherry)**
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**

- **Supervised Learning (icing)**
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**

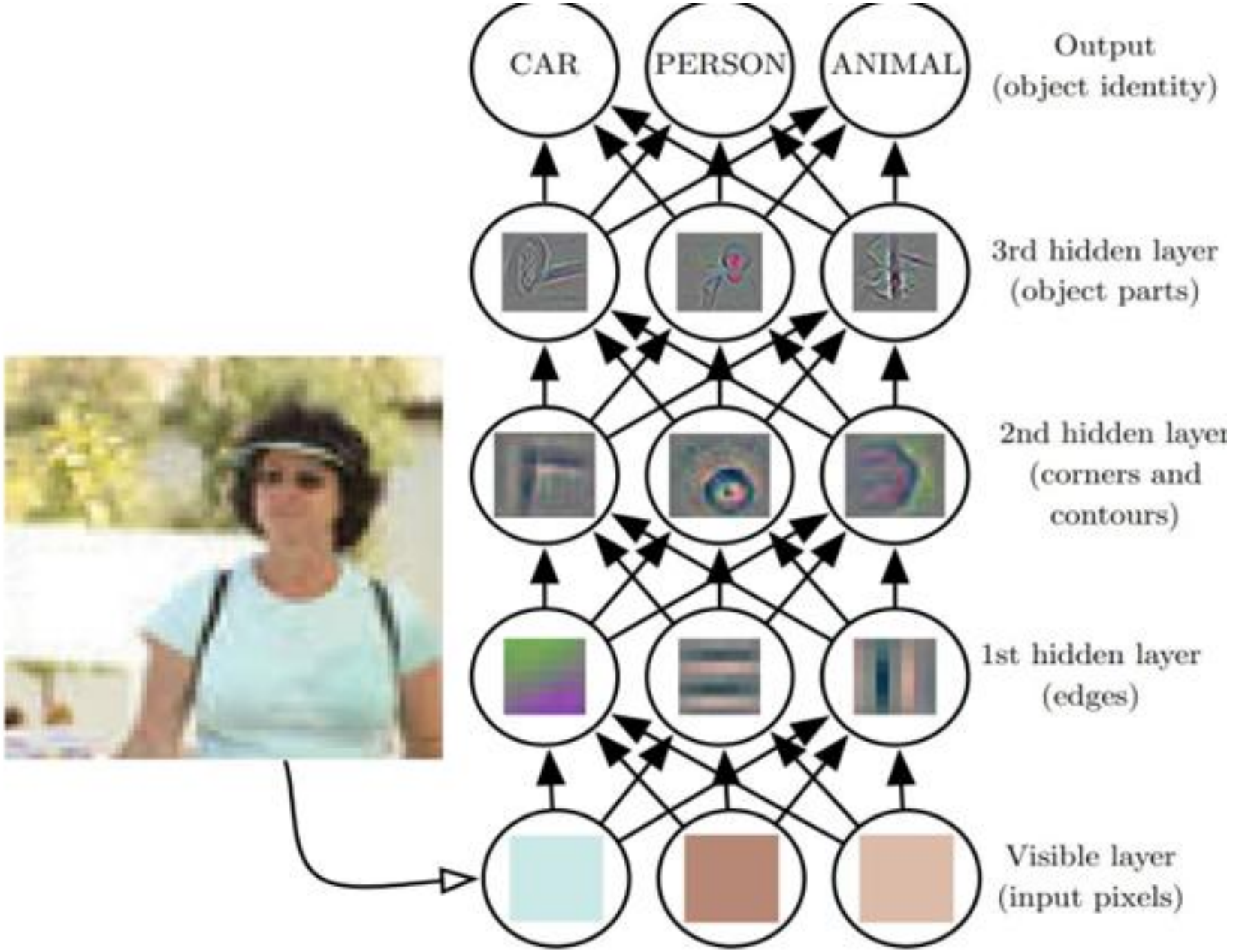
- **Self-Supervised Learning (cake génoise)**
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

Predict missing pieces

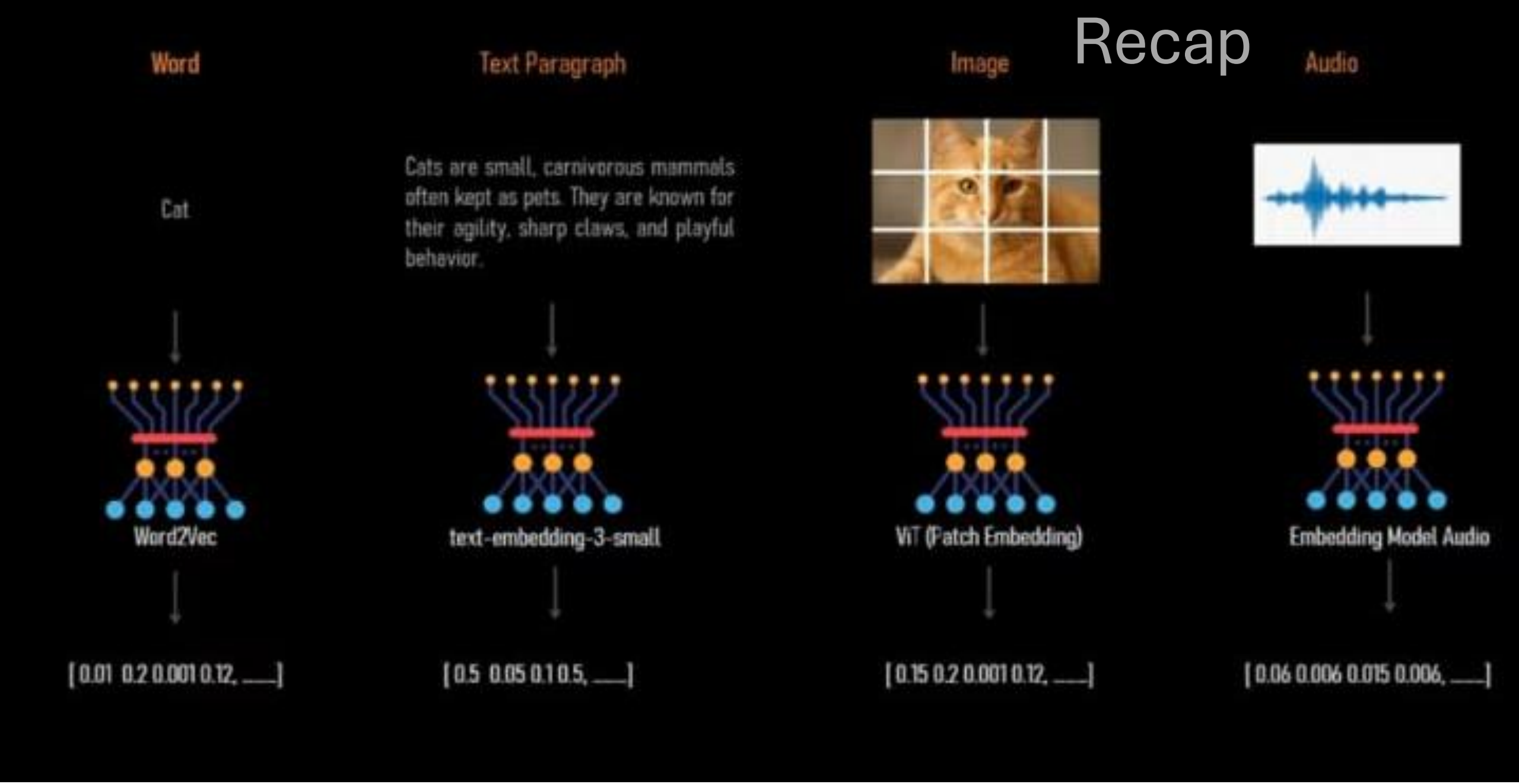
Context Encoders



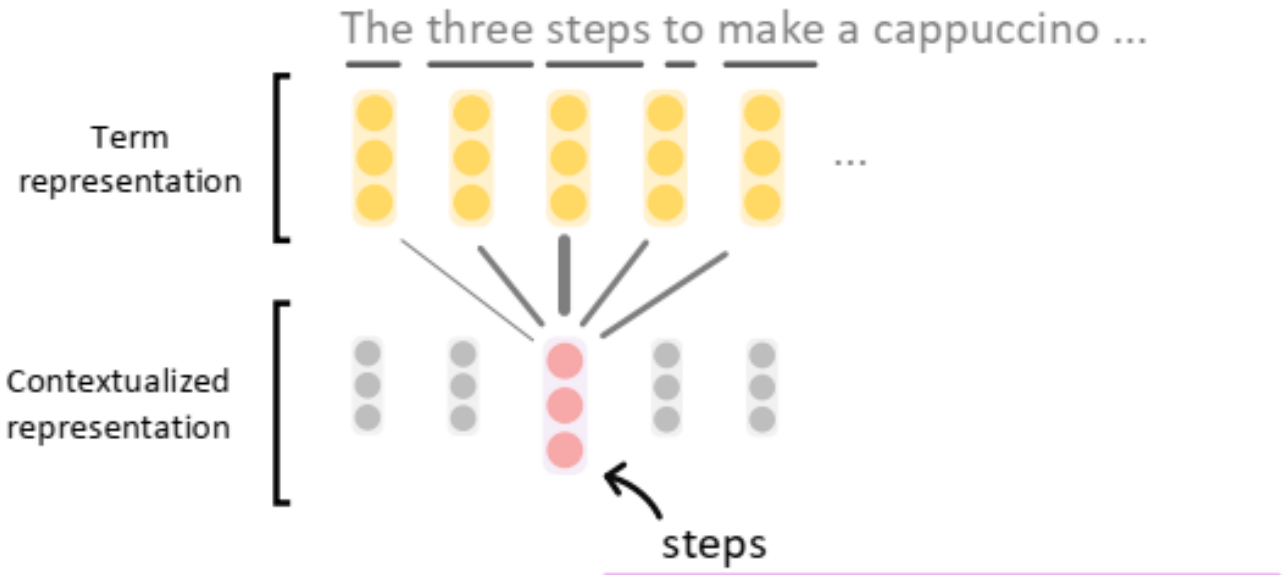
Recap


CAR
PERSON
ANIMAL
Output
(object identity)
3rd hidden layer
(object parts)
2nd hidden layer
(corners and
contours)
1st hidden layer
(edges)
Visible layer
(input pixels)

Recap

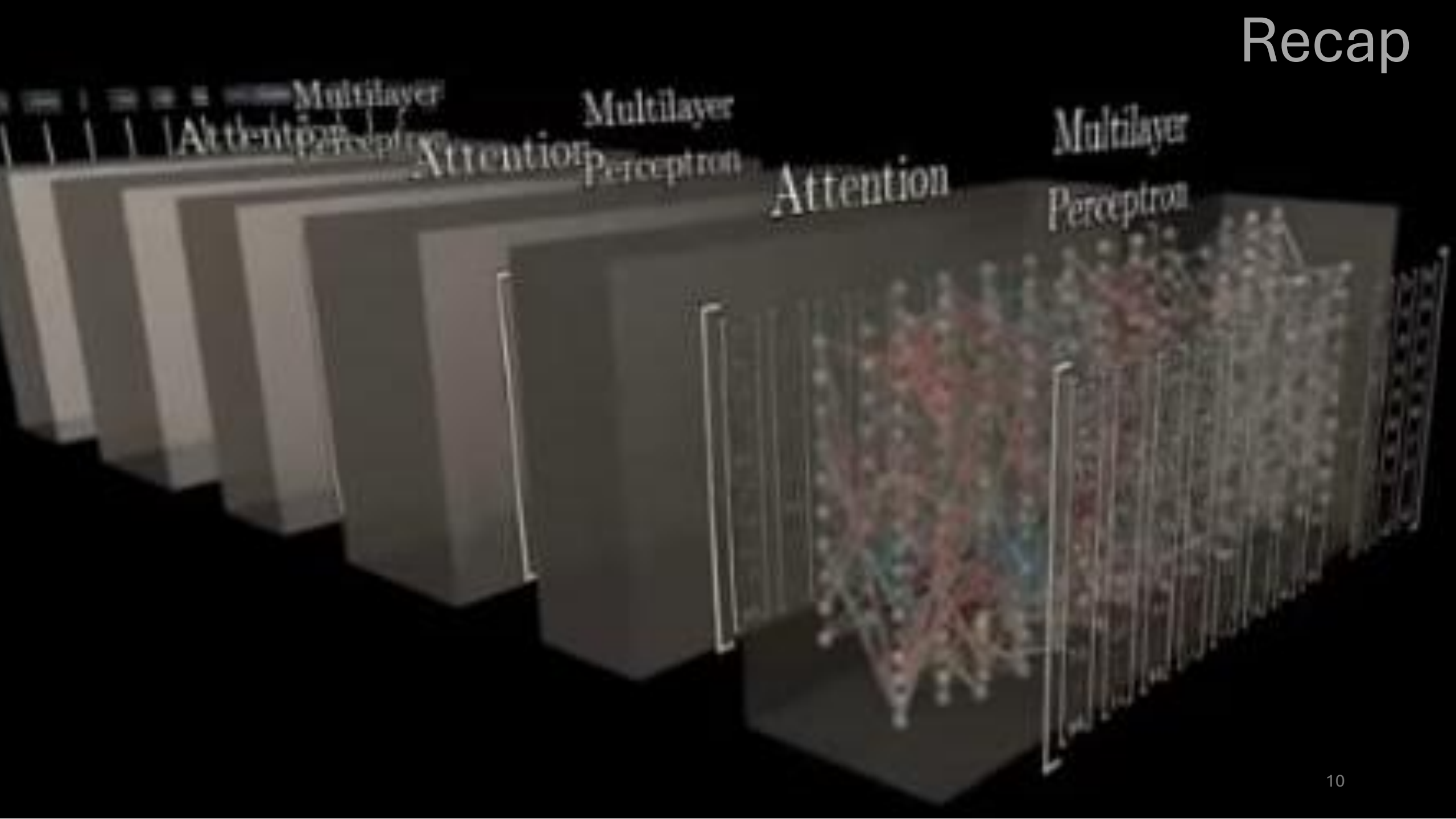

Word
Cat
Word2Vec
[0.01 0.2 0.001 0.12, ____]
Text Paragraph
Cats are small, carnivorous mammals often kept as pets. They are known for their agility, sharp claws, and playful behavior.
text-embedding-3-small
[0.5 0.05 0.1 0.5, ____]
Image
ViT (Patch Embedding)
[0.15 0.2 0.001 0.12, ____]
Audio
Embedding Model Audio
[0.06 0.006 0.015 0.006, ____]

Contextualization via Self-Attention



- Learn meaning based on surrounding context for every word occurrence
- This *contextualization* combines representations
- Context here is local to the sequence (not necessary a fixed window)
- Is computationally intensive $O(n^2)$
  - Every token attends to every other token

Recap


Multilayer
Perceptron
Attention
Multilayer
Perceptron
Attention
Multilayer
Perceptron

Transformer

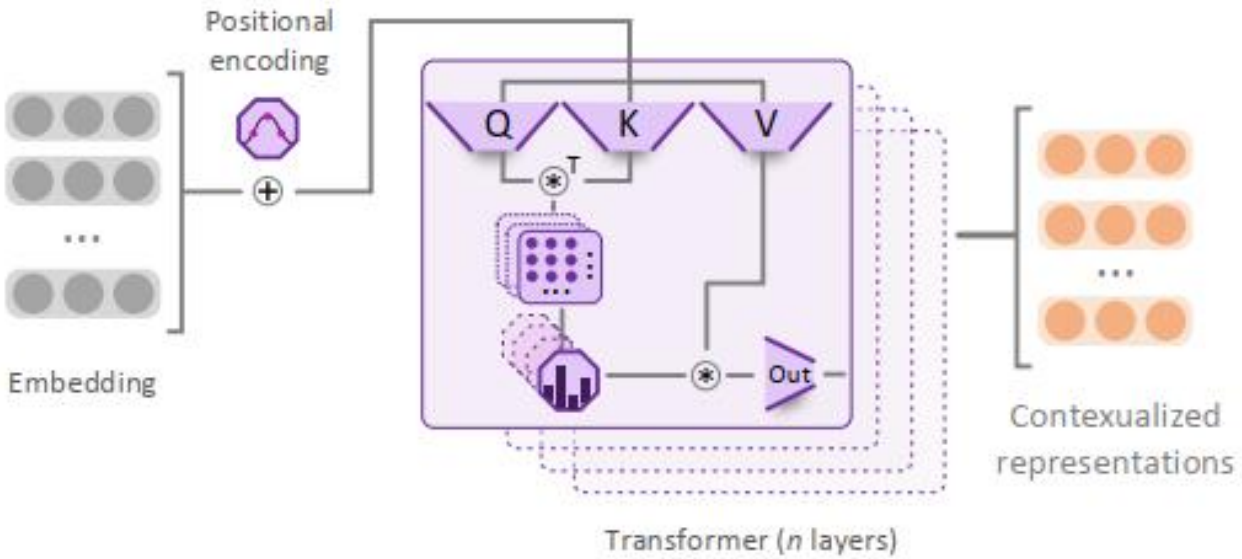
Recap

- Transformers contextualize with multi-head self-attention
  - Every token attends to every other token $O(n^2)$ complexity
- Commonly Transformers stack many layers
- Can be utilized as encoder-only or encoder-decoder combination
- Do not require any recurrence
  - The attention breaks down to a series of matrix multiplications over the sequence
- Initially proposed in translation
  - Now the backbone of virtually every NLP advancement in the last years

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, et al.
Attention is all you need. In NeurIPS. 2017.

Transformer – Architecture

Recap



- We embed (subword) tokens
- We add a positional encoding
- In each Transformer-Layer:
  - Project each vector with 3 linear layers to **Q**uery, **K**ey, **V**alue
  - Transform projections to another multi-head dimension
  - Matrix-multiply Query & Key
  - Get Q-K attention via softmax
  - Multiply attention with Values and project back to output

Nice detailed walkthrough code + paper:
https://nlp.seas.harvard.edu/2018/04/03/attention.html

Self-Attention Definition

- The Transformer Self-Attention is defined as:

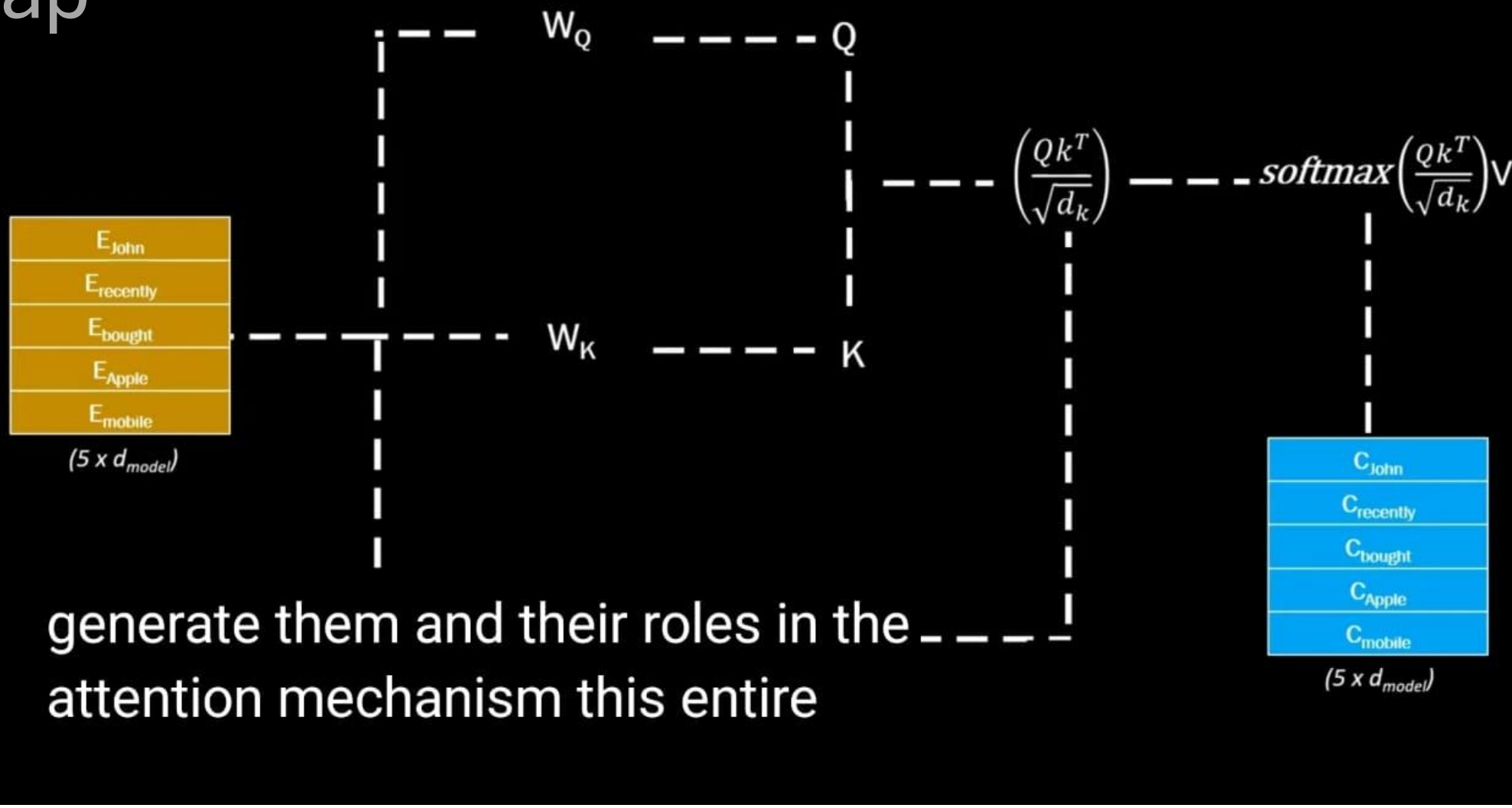
$$\text{SelfAttention}(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) * V$$

- Q, K, V are projections of the **same input** sequence
- This definition hides quite a bit of complexity, visible in the code

Recap

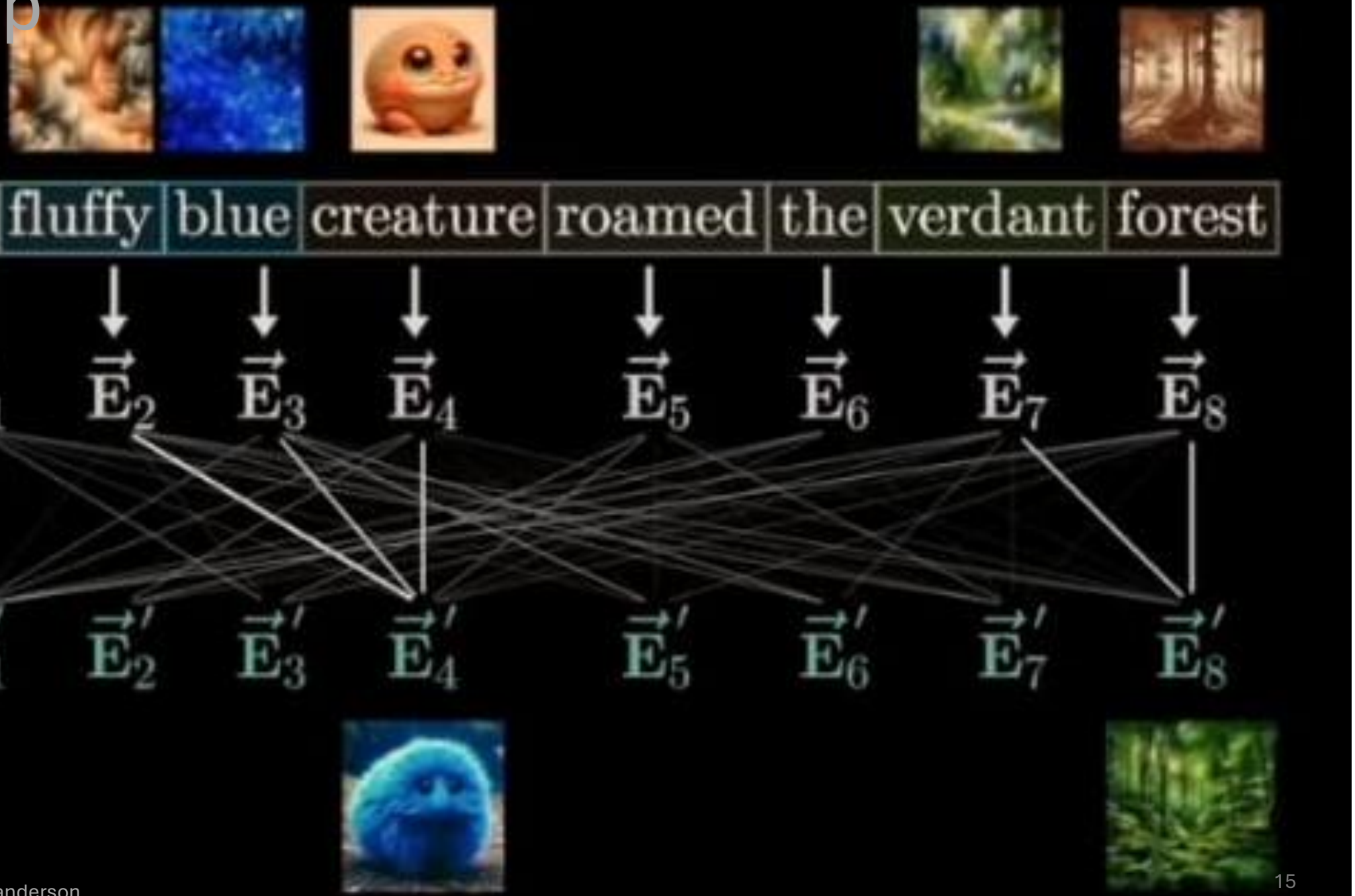
Q	Attention "Query"
K	Attention "Key"
V	Attention "Value"
d_k	Dimension of key embeddings

Recap



Credit: Grant Sanderson

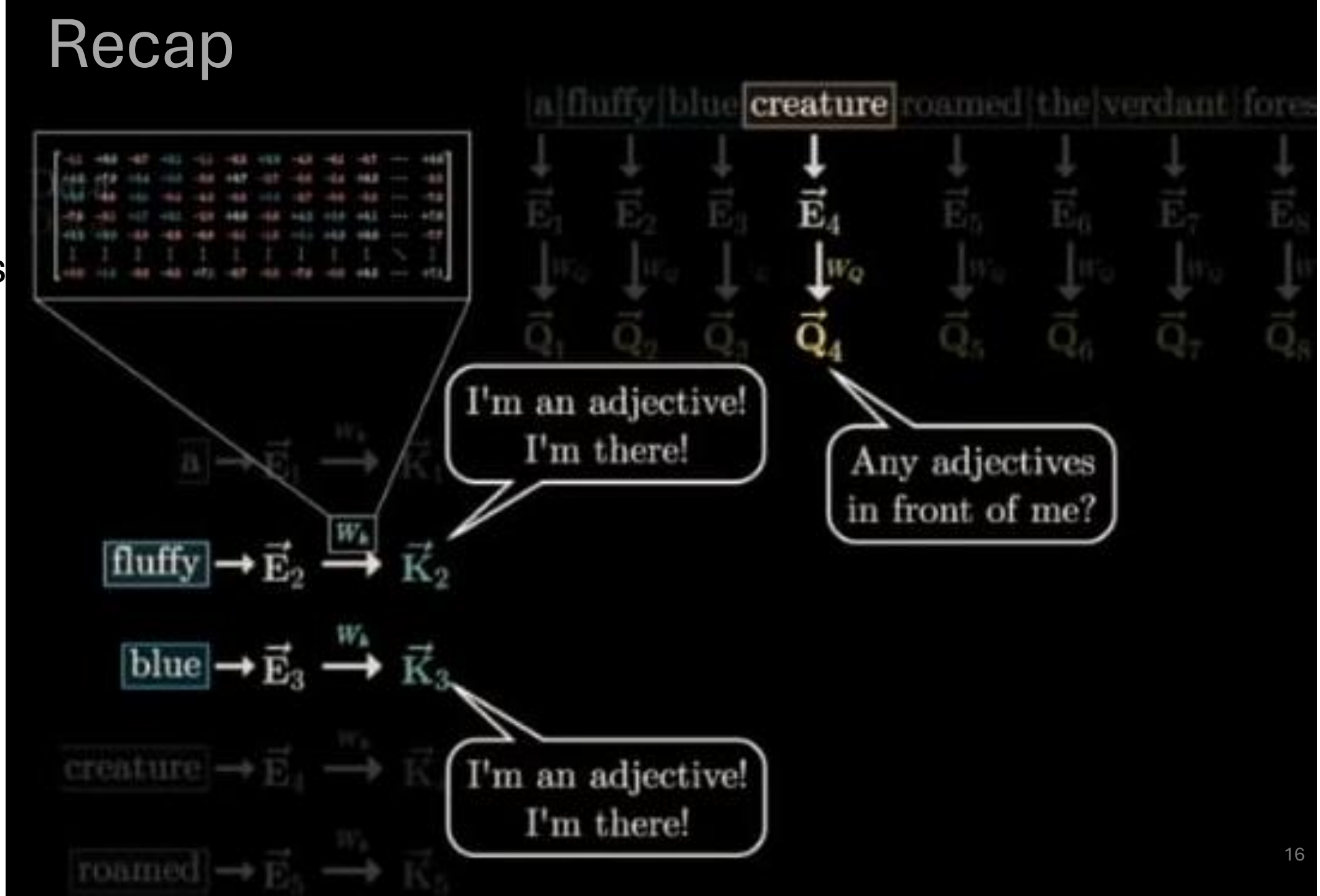
Recap



Credit: Grant Sanderson

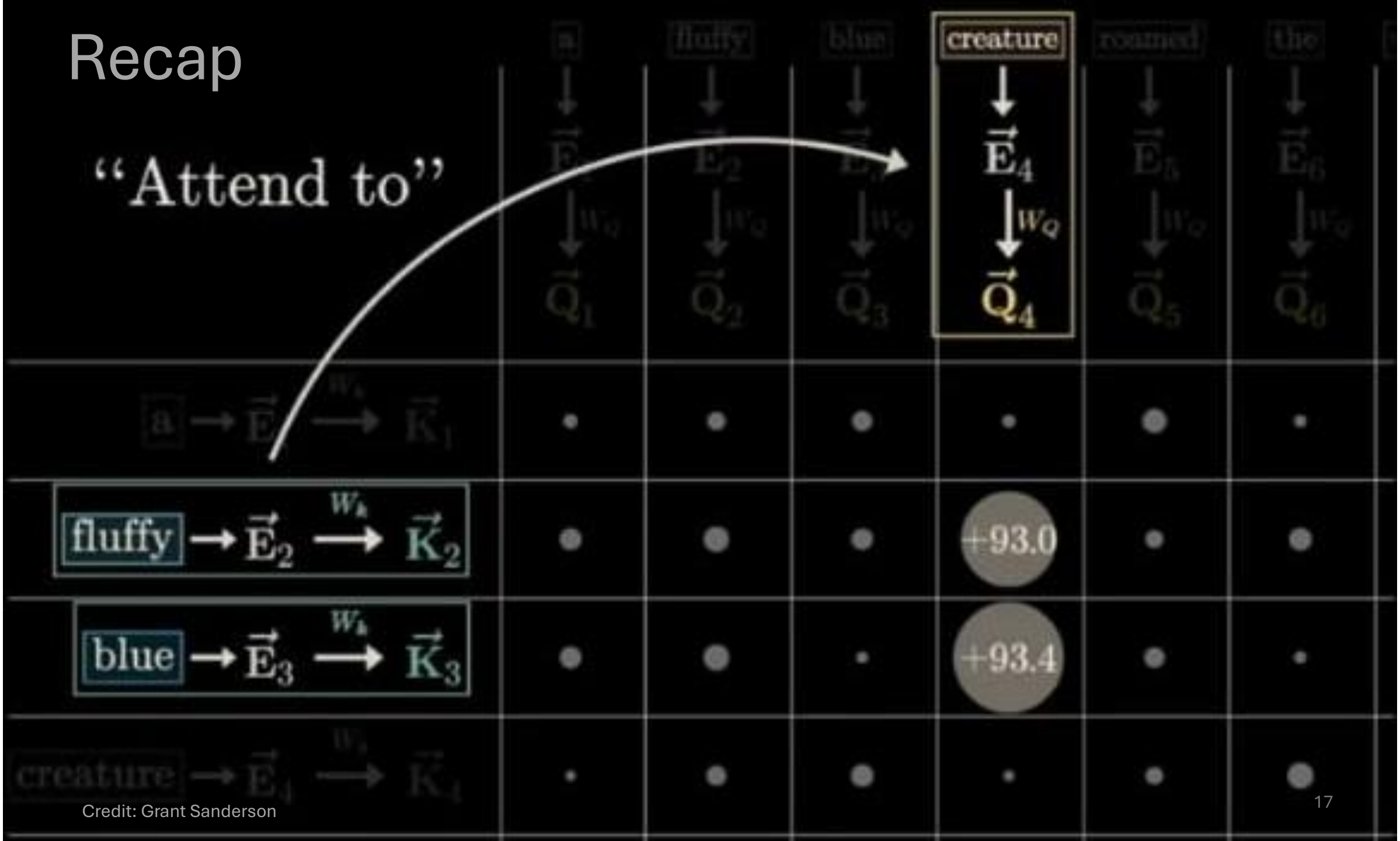
Recap

- S



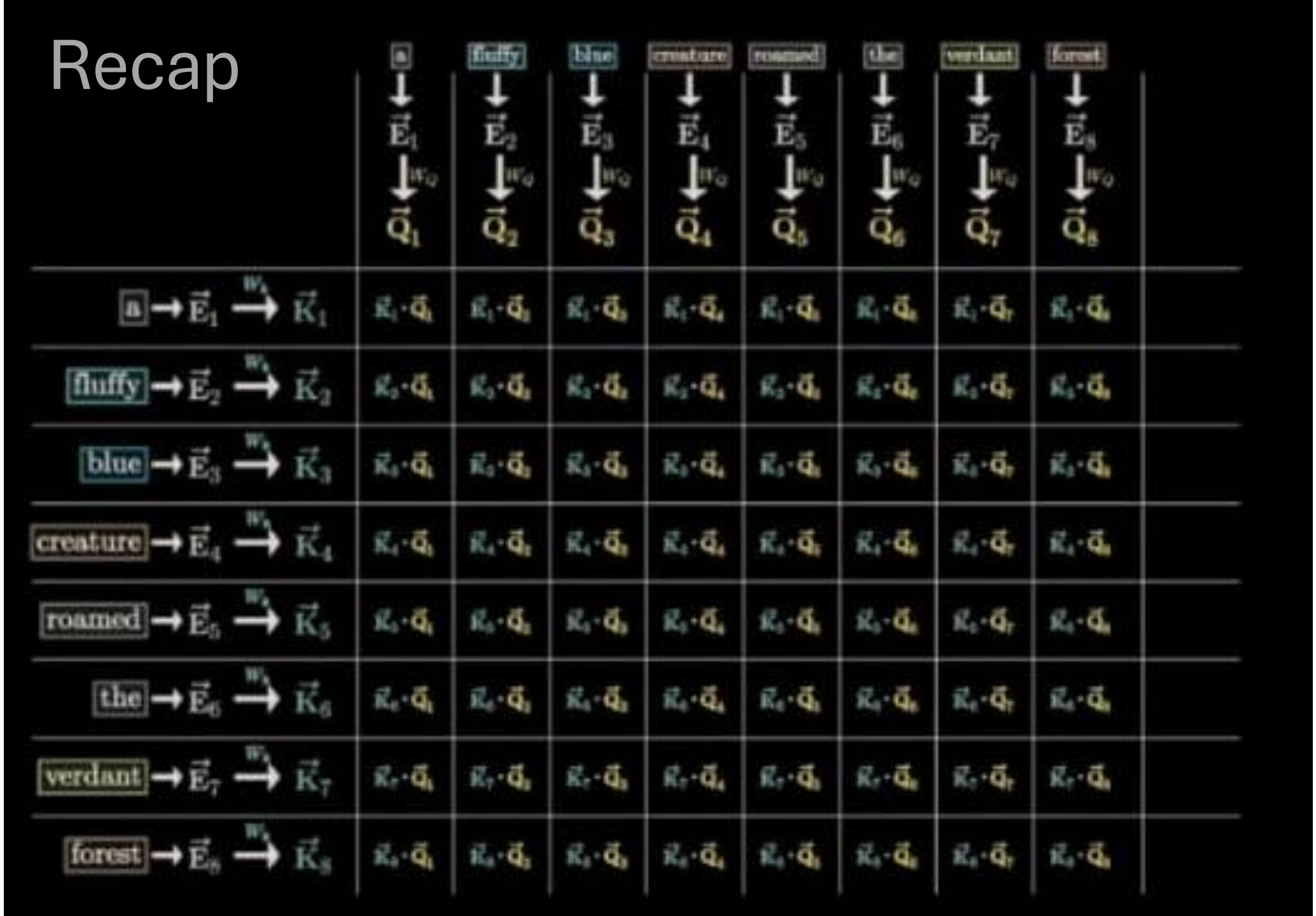
Credit:
Grant Sanderson

Recap



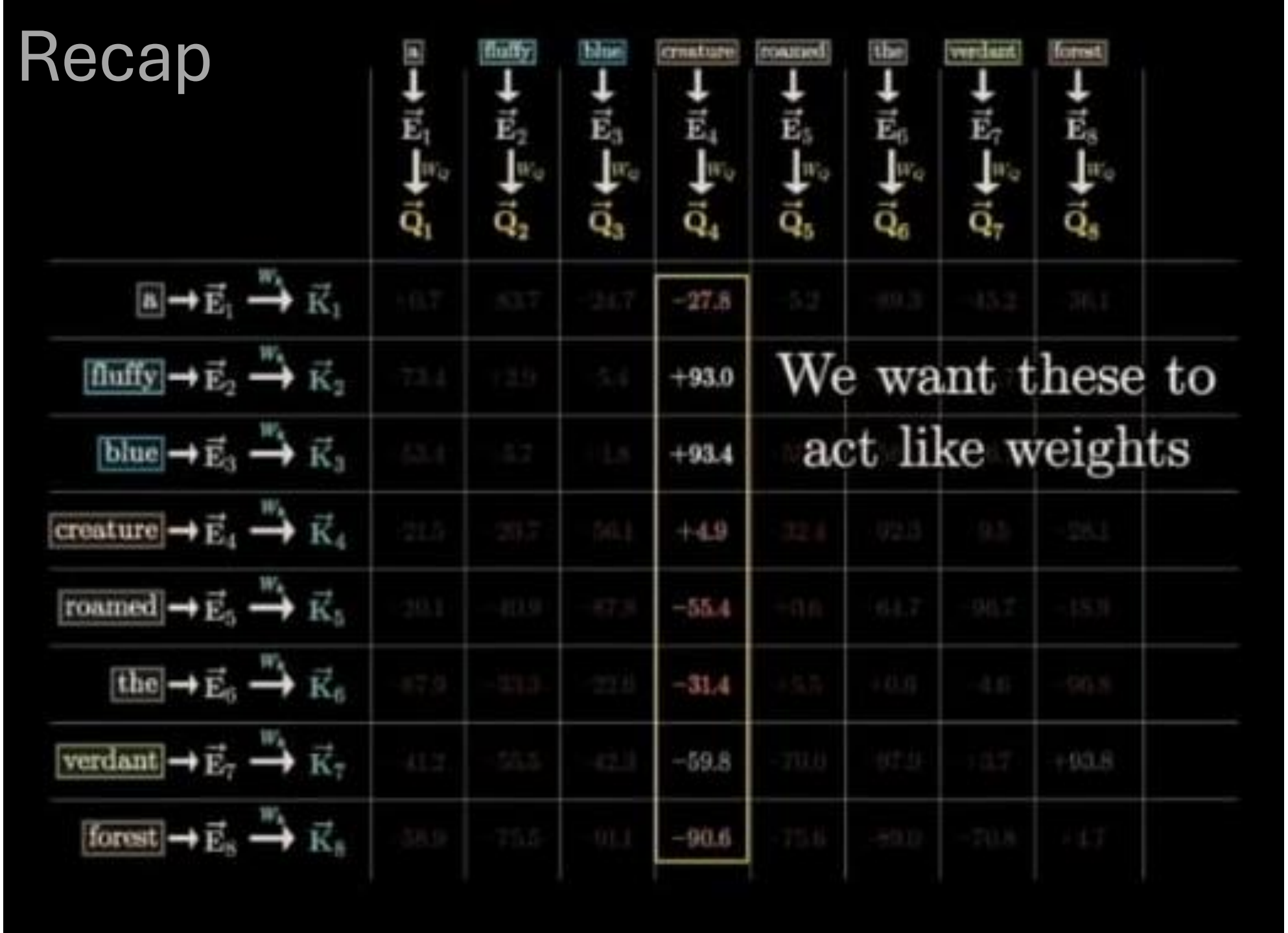
Credit: Grant Sanderson

Recap



Credit:
Grant Sanderson

Recap

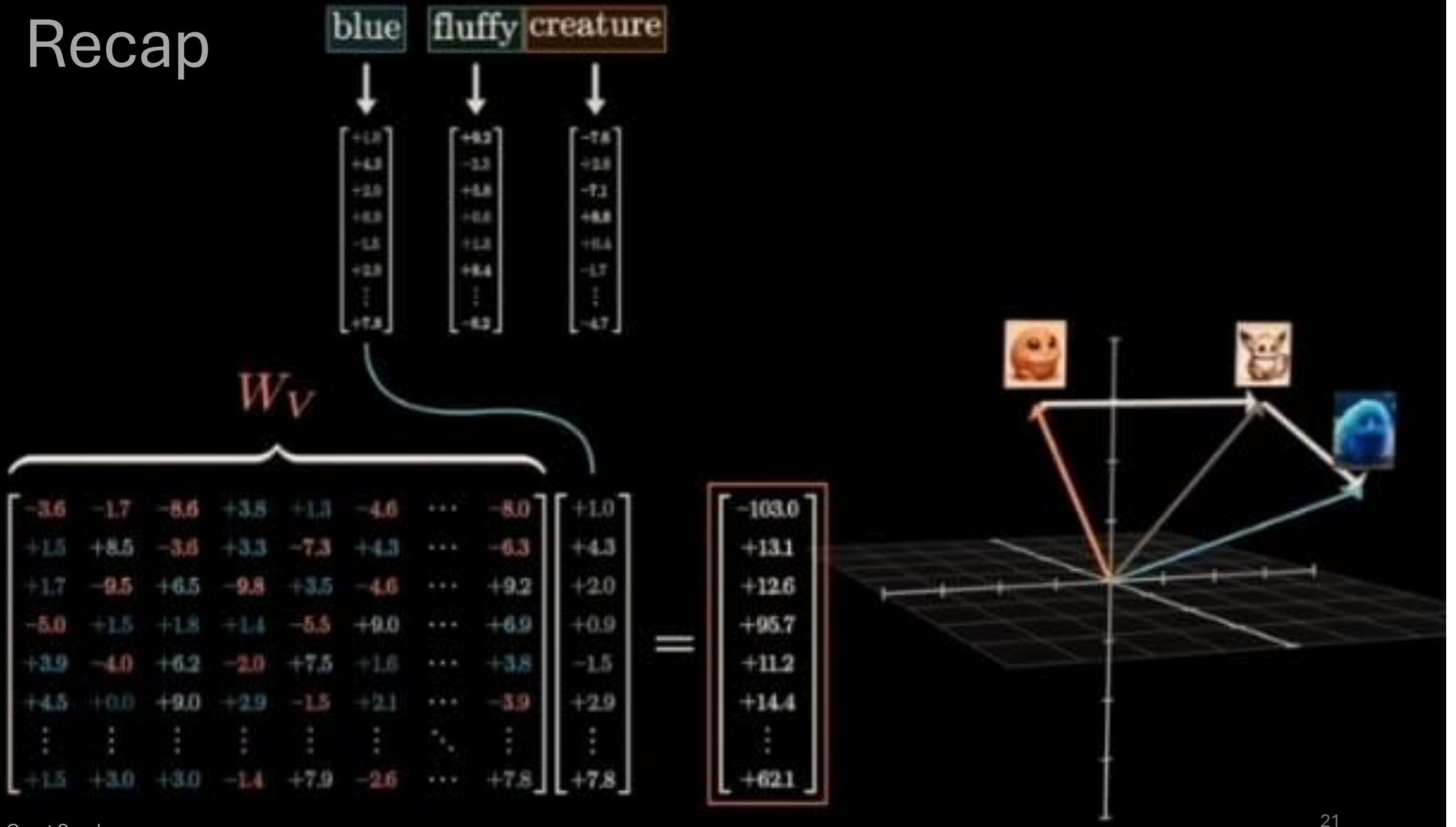


Credit:
Grant Sanderson

Recap

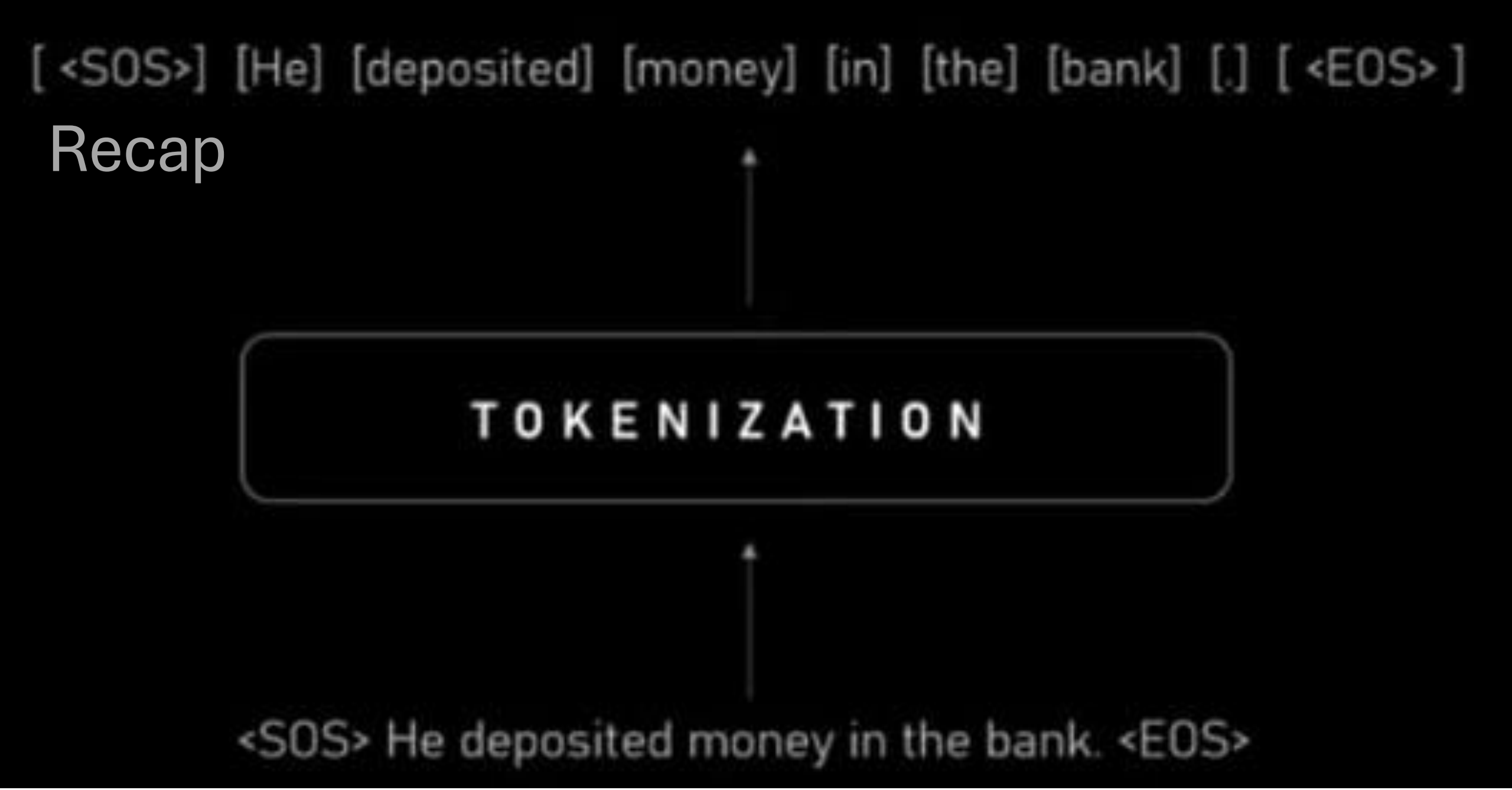
Credit:
Grant Sanderson

Recap



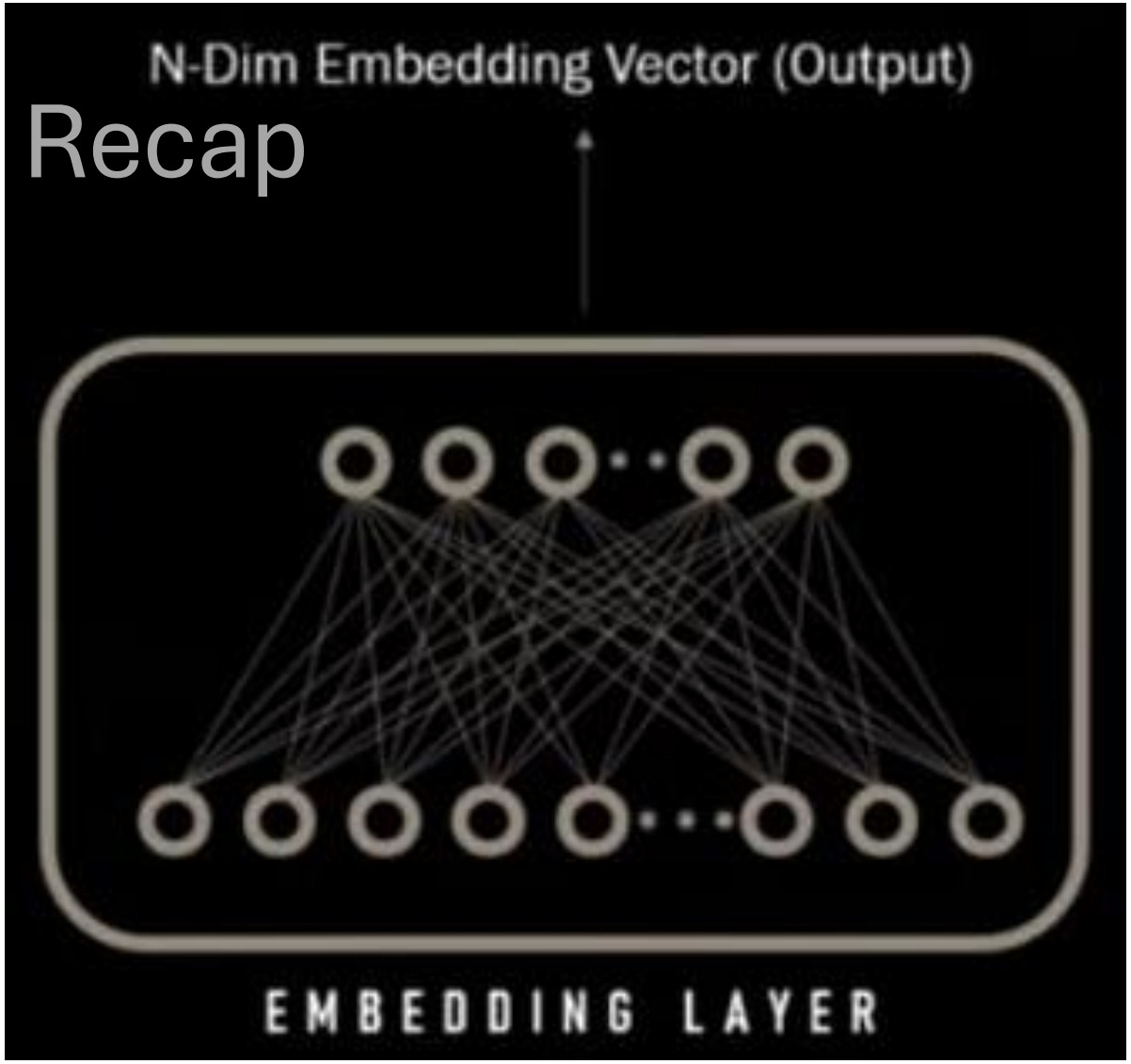
Credit: Grant Sanderson

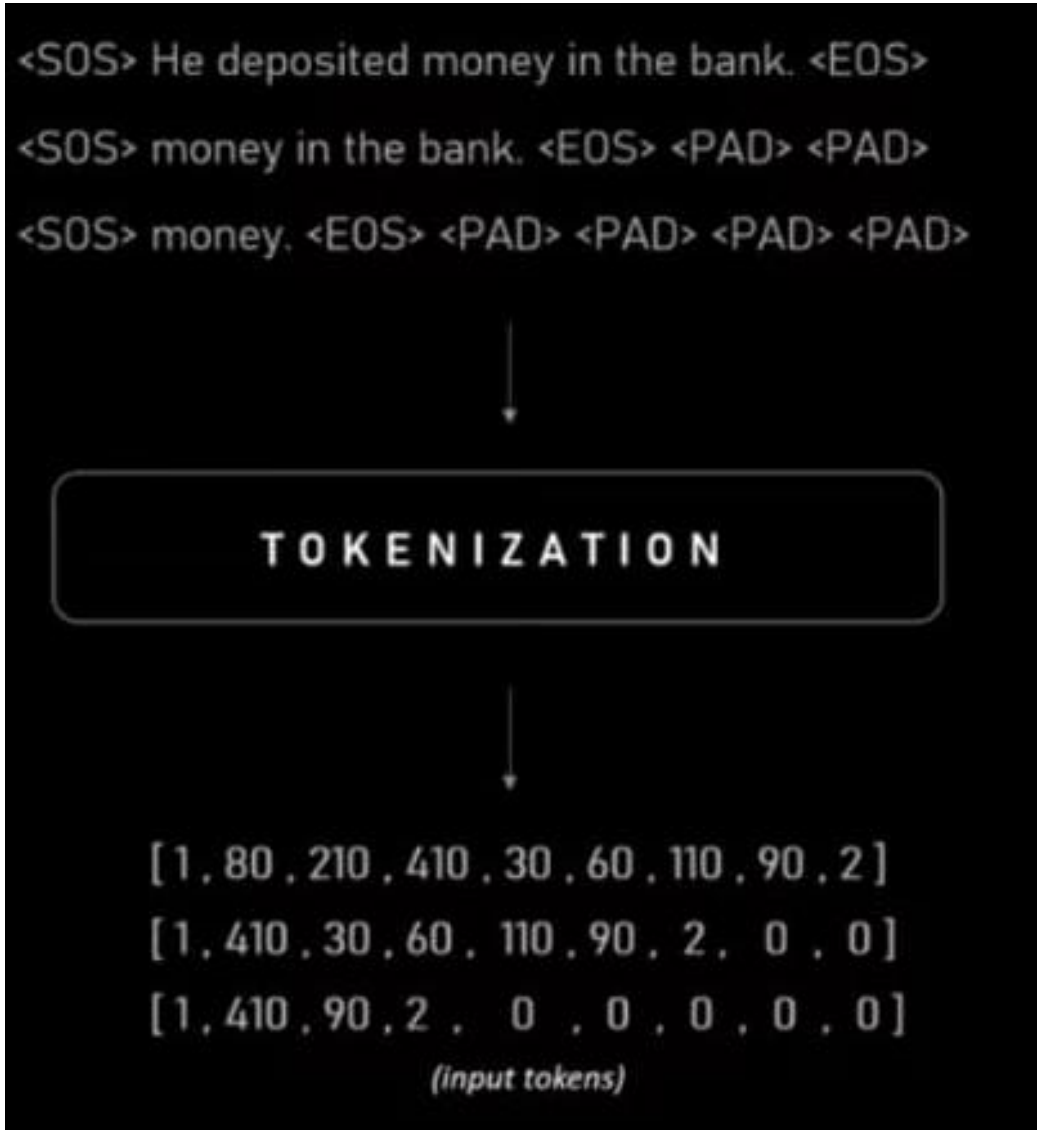
Recap



Credit: Grant Sanderson

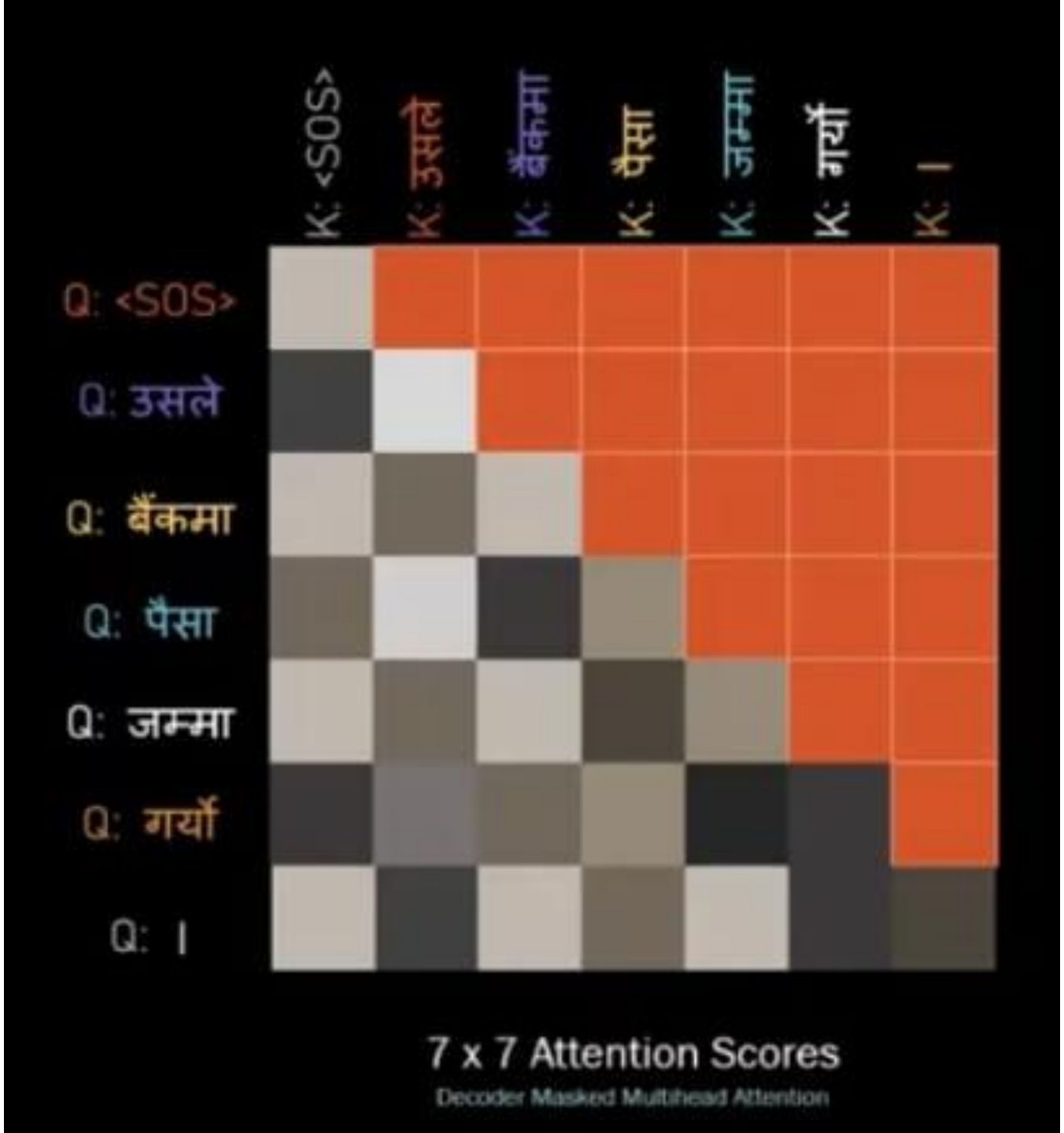
Recap





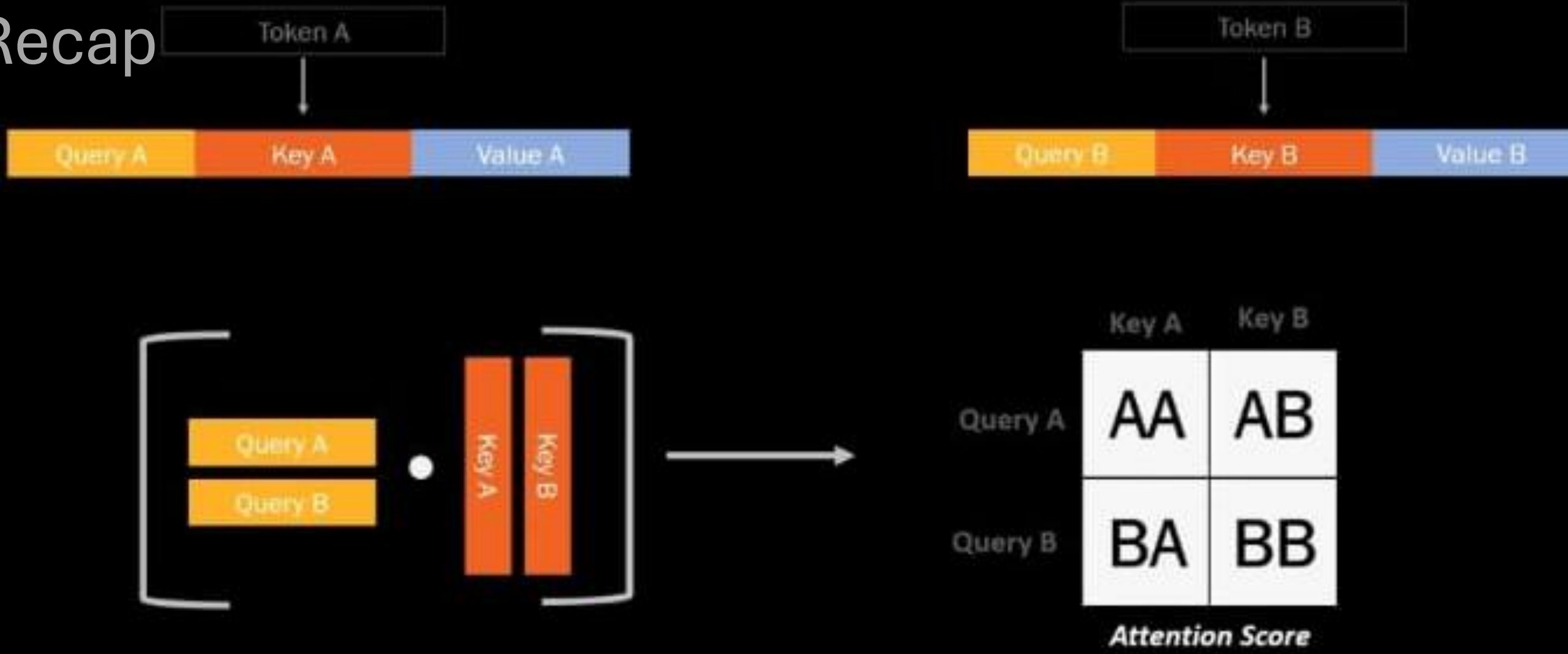
Credit: Grant Sanderson

Recap



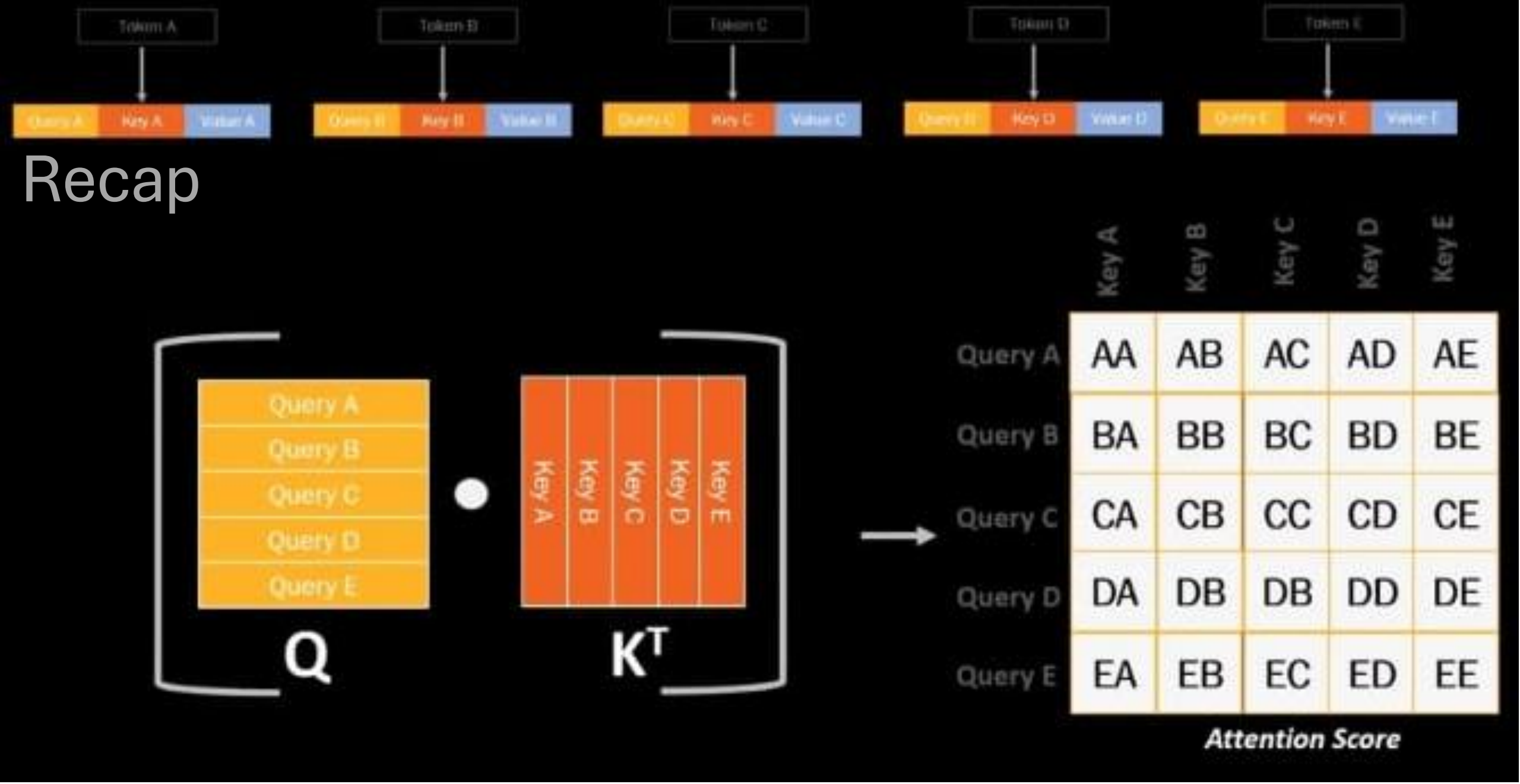
Credit: Grant Sanderson

Recap



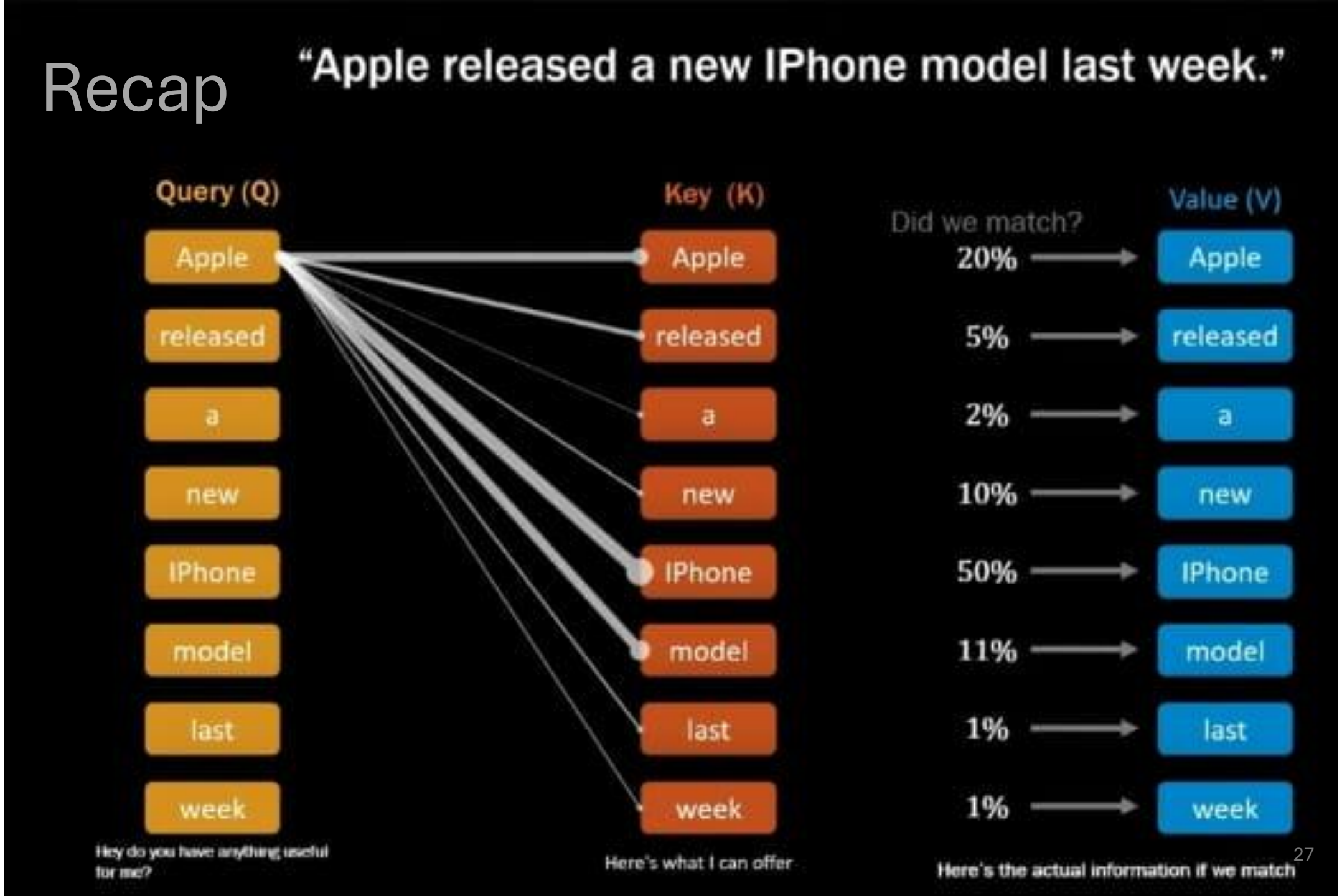
Credit: Grant Sanderson

Recap



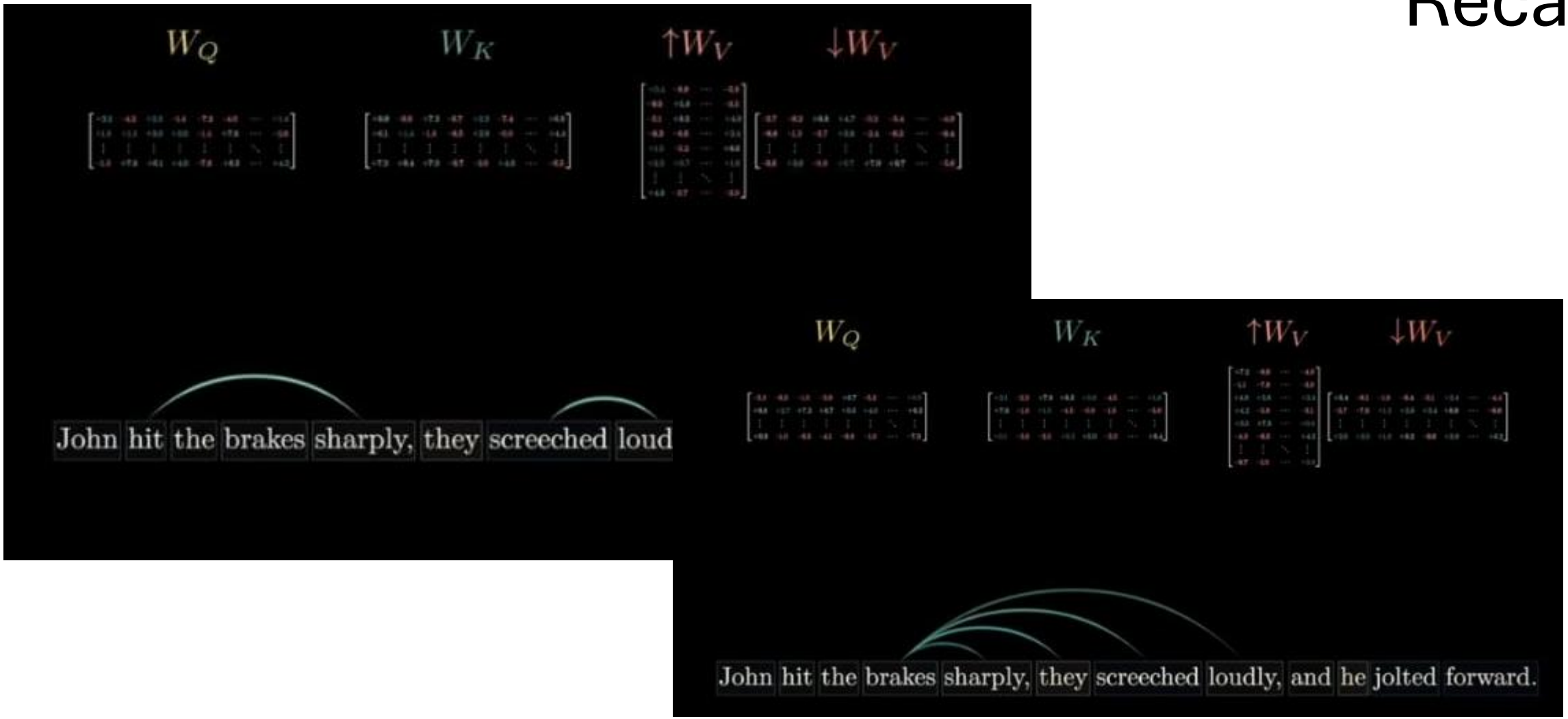
Credit: Grant Sanderson

Recap



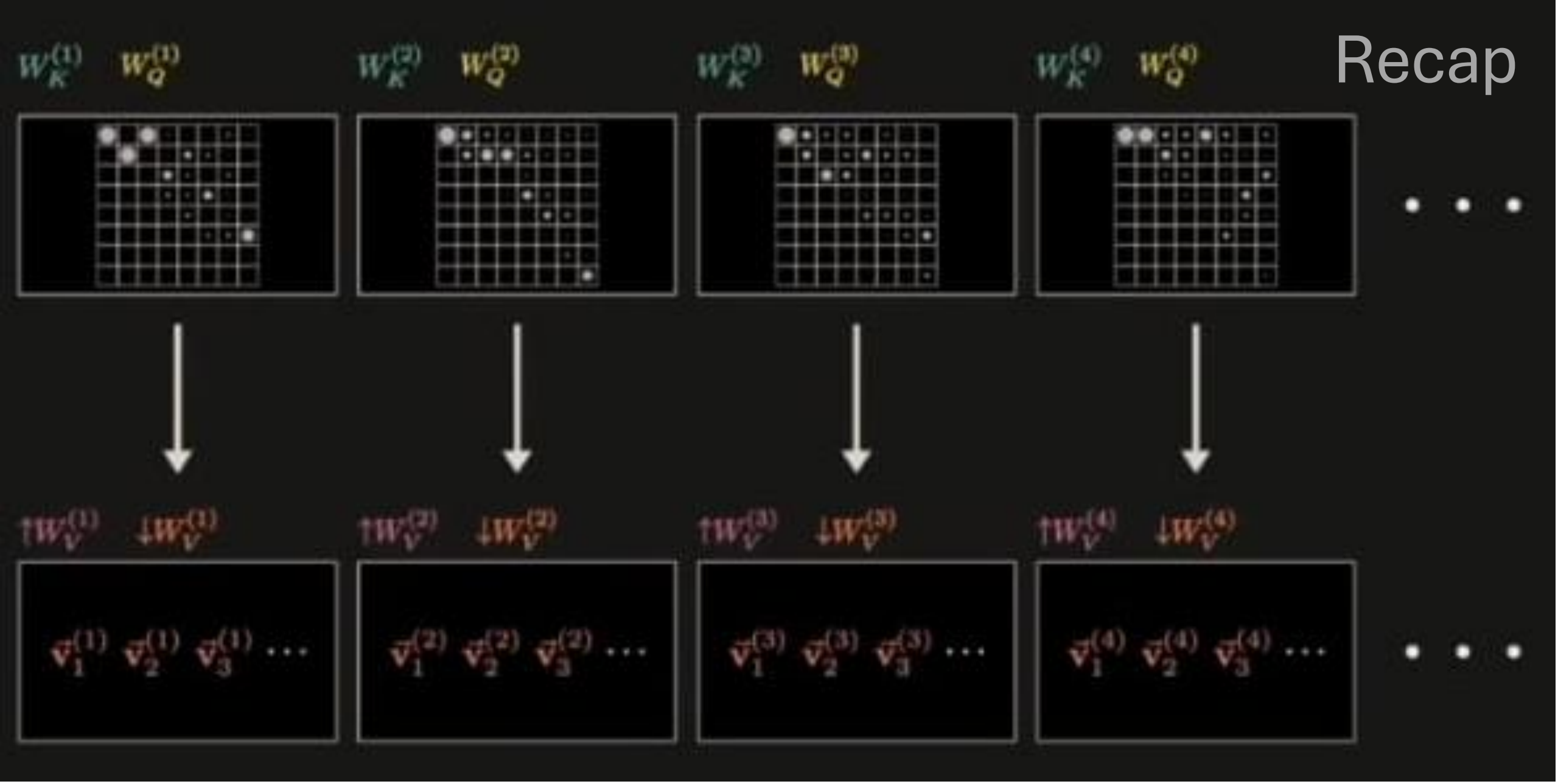
Credit:
Grant Sanderson

Recap



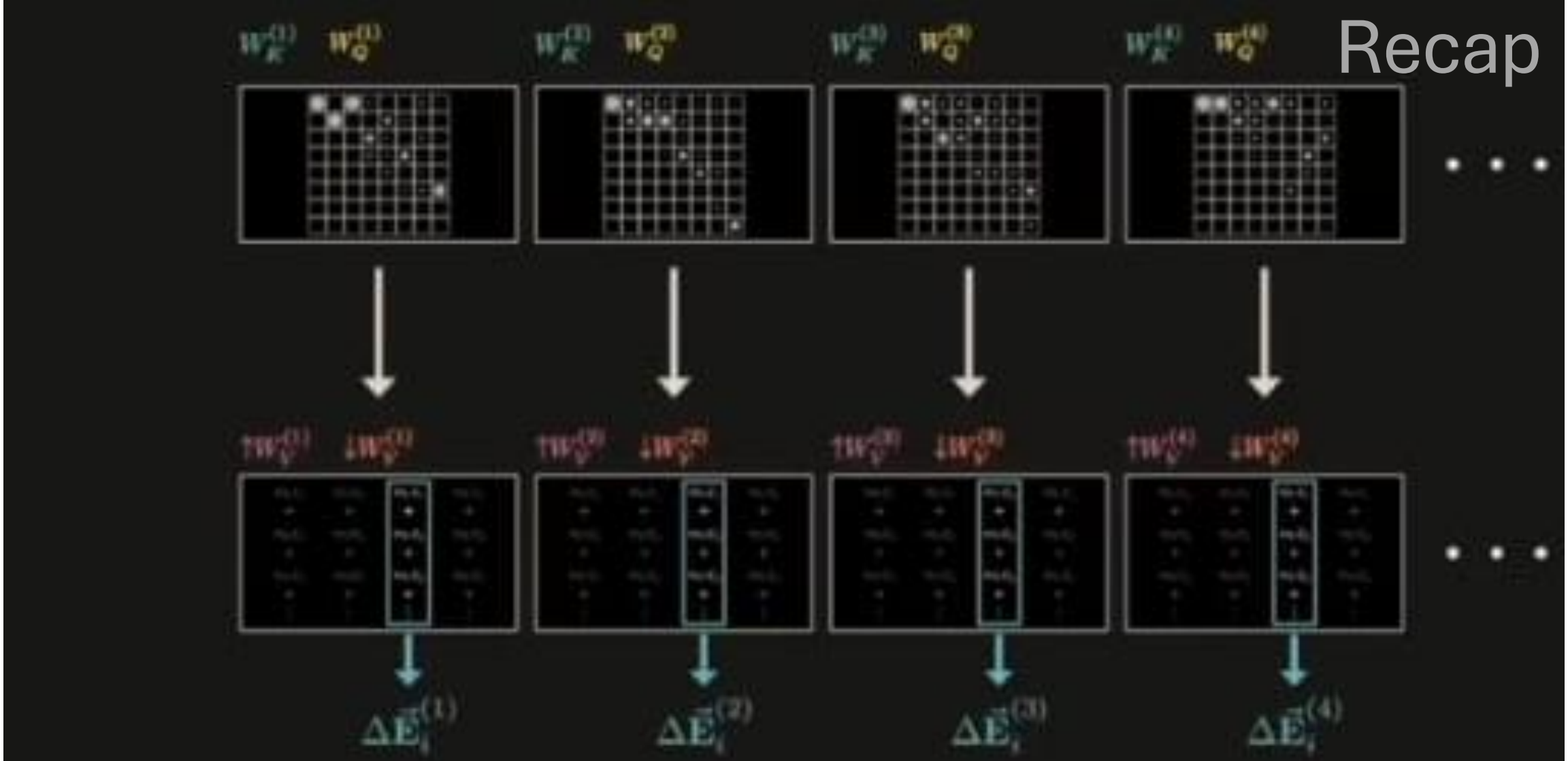
Credit: Grant Sanderson

Recap



Credit: Grant Sanderson

Recap



$$\text{Original embedding} \quad \vec{\mathbf{E}}_i + \Delta\vec{\mathbf{E}}_i^{(1)} + \Delta\vec{\mathbf{E}}_i^{(2)} + \Delta\vec{\mathbf{E}}_i^{(3)} + \Delta\vec{\mathbf{E}}_i^{(4)} + \cdots$$

Credit: Grant Sanderson

Transformer – Positional Encoding

Recap

- Transformers add sinusoid curves to the input, before the attention
  - Informs about relative position inside the sequence
  - Removes need for explicit recurrence patterns

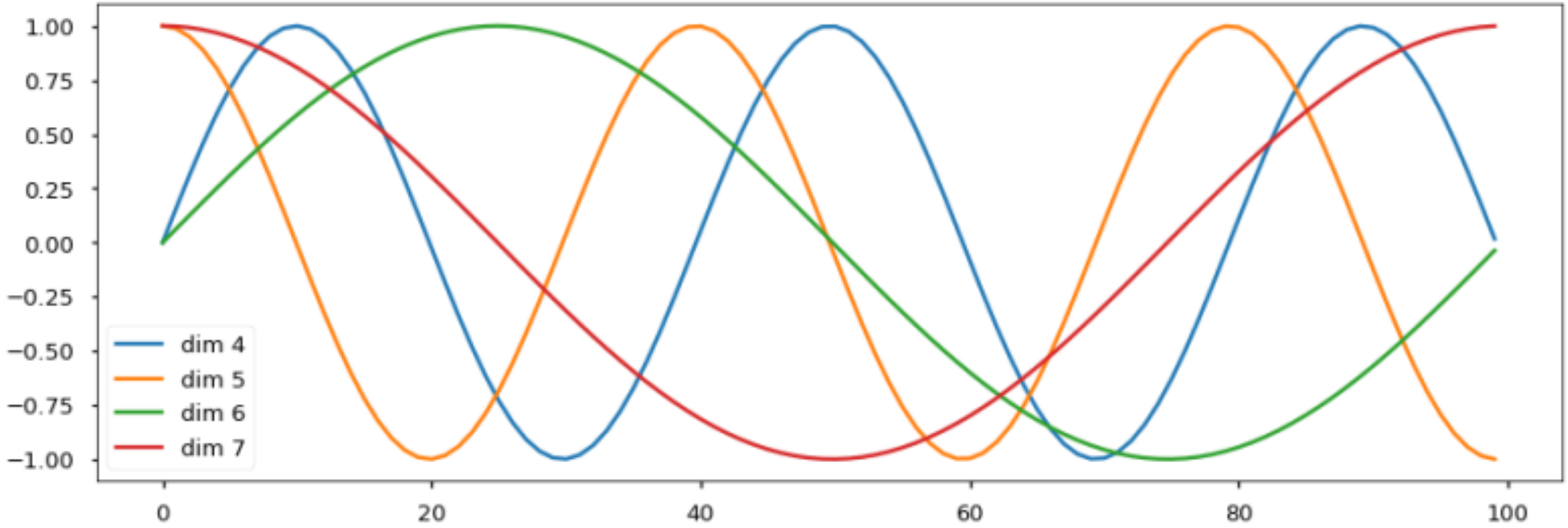
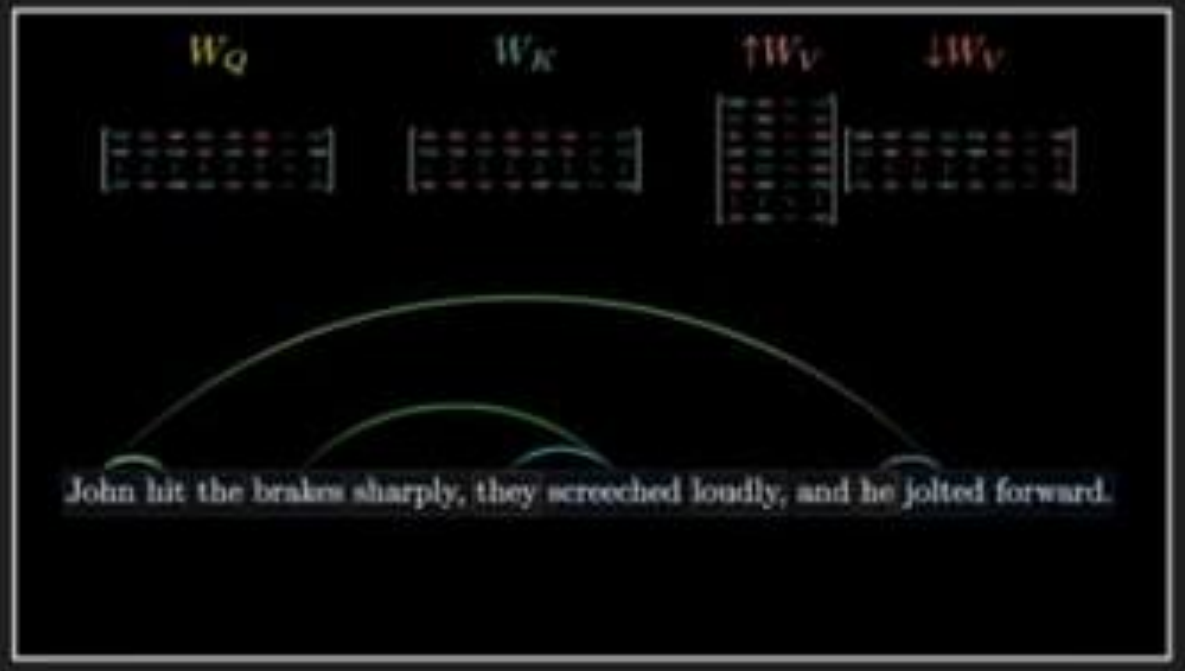


Figure from https://nlp.seas.harvard.edu/2018/04/03/attention.html

Recap

Self-attention



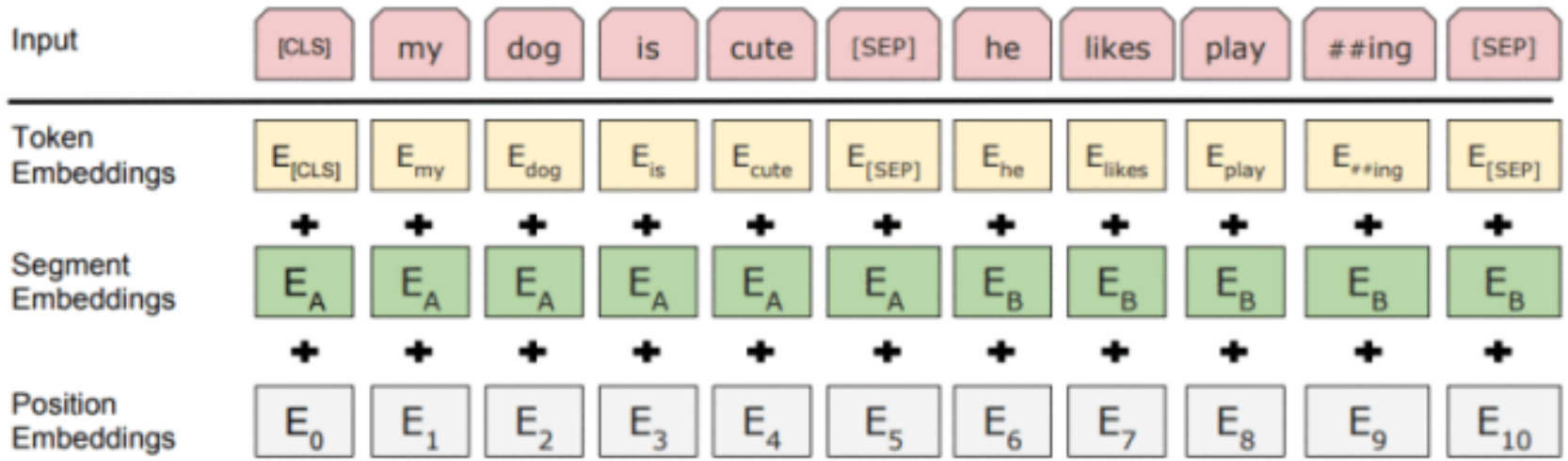
Cross-attention

Credit: Grant Sanderson

BERT - Input

Recap

- Either one or two sentences, always prepended with [CLS]
  - BERT adds trained position embeddings & sequence embeddings



More info: https://towardsml.com/2019/09/17/bert-explained-a-complete-guide-with-theory-and-tutorial/

BERT - Model

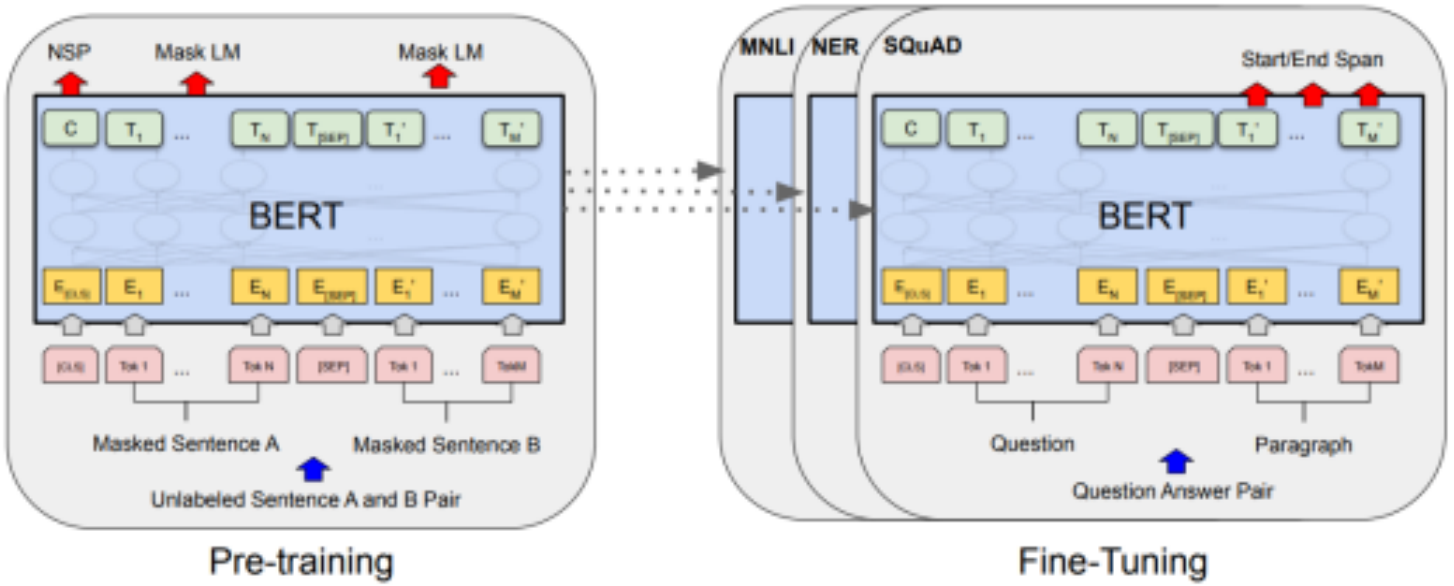
Recap

- Model itself is quite simple: n Layers of stacked Transformers
  - Using LayerNorm, GeLU activations (like ReLU, but with a grace swing under 0)
  - Task specific heads on top to pool [CLS] or individual token representations
  - Every Transformer layer receives as input the output of the previous one
- The [CLS] token itself is only special because we train it to be
  - No mechanism inside the model that differentiates it from other tokens
- Novel contributions center around pre-training & workflow

BERT - Workflow

Recap

- Someone with lots of compute or time pre-trains a large model
  - BERT uses Masked Language Modelling [MASK] and Next Sentence Prediction [CLS]
- We download it and fine-tune on our task

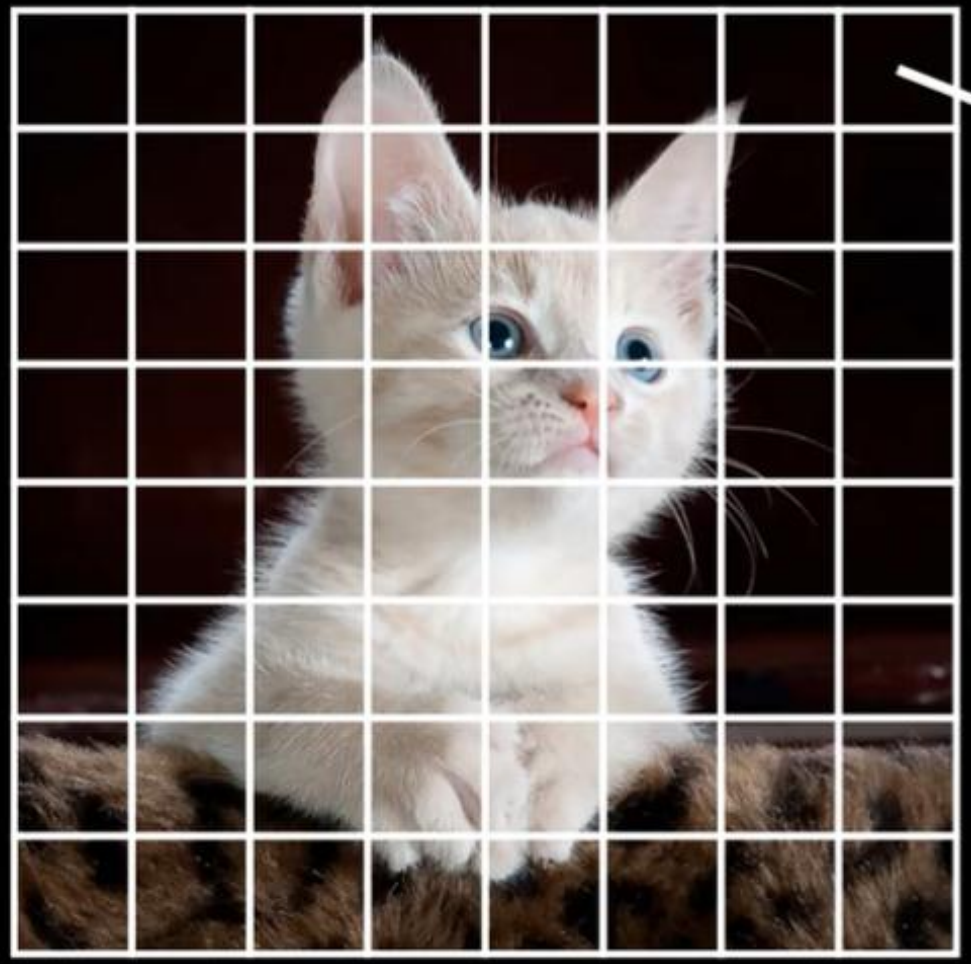


BERT++

Recap

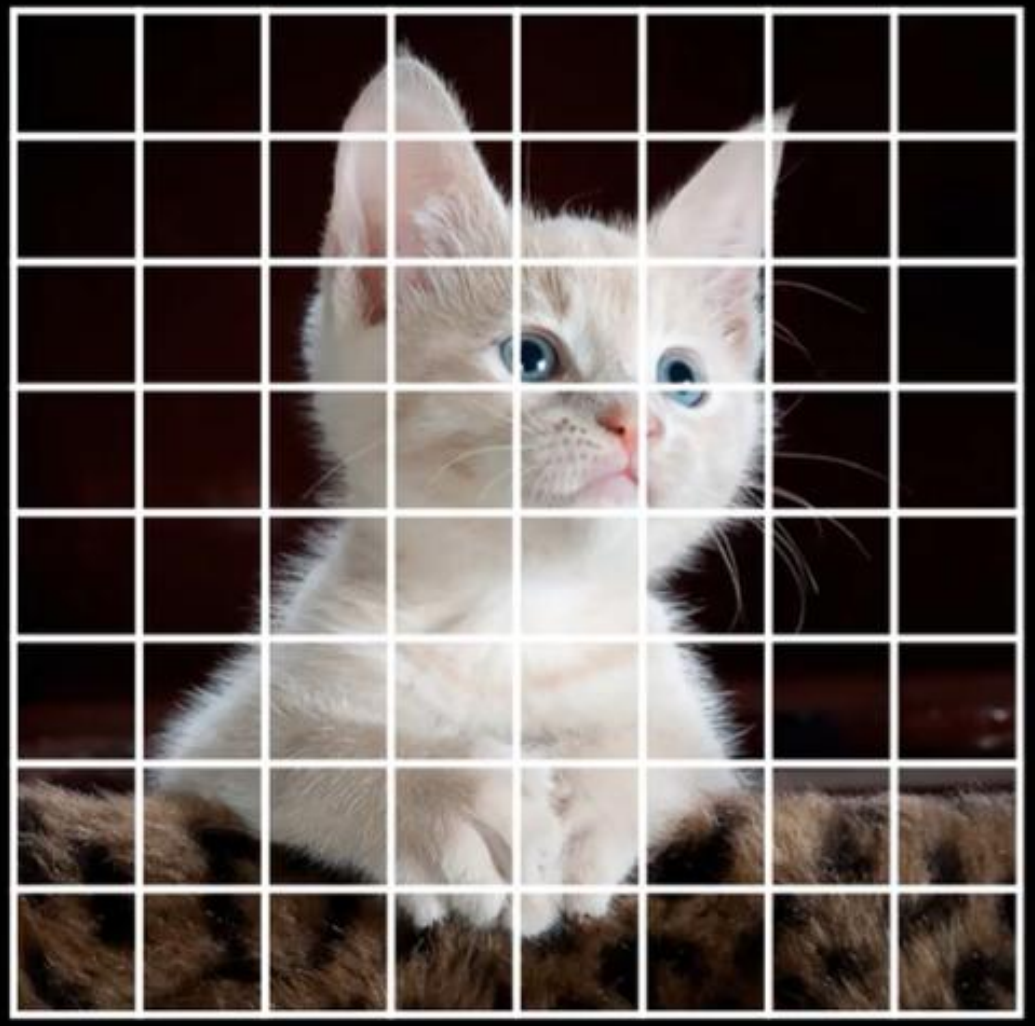
- Same as with Transformer variations, there are now many BERT variants
  - For many languages
  - Domains like biomedical publications
  - Different architectures, but similar workflow:
    Roberta, Transformer-XL, XLNet, Longformer …
- Main themes for adapted architectures:
  - Bigger
  - More efficient
  - Allowing for longer sequences (BERT is capped at 512 tokens in total)

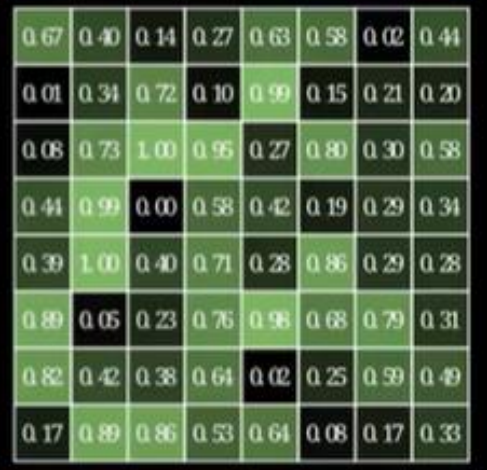
Vision Transfomer(ViT)

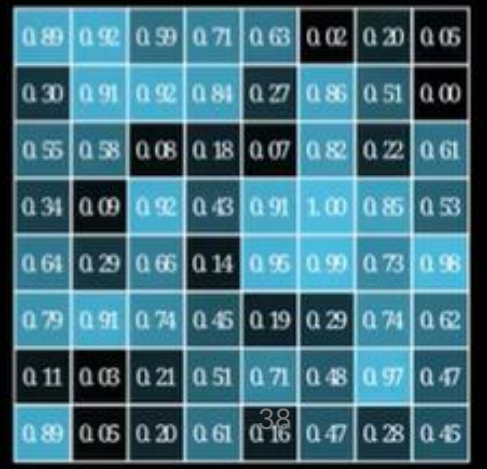


102	220	225	95	179	61	234	208
92	3	98	203	14	149	245	46
106	244	99	187	71	212	153	199
188	174	65	153	20	44	208	152
102	214	240	39	121	24	34	114
210	65	239	39	214	244	151	25
74	145	222	14	202	85	145	117
87	184	189	221	116	237	109	85

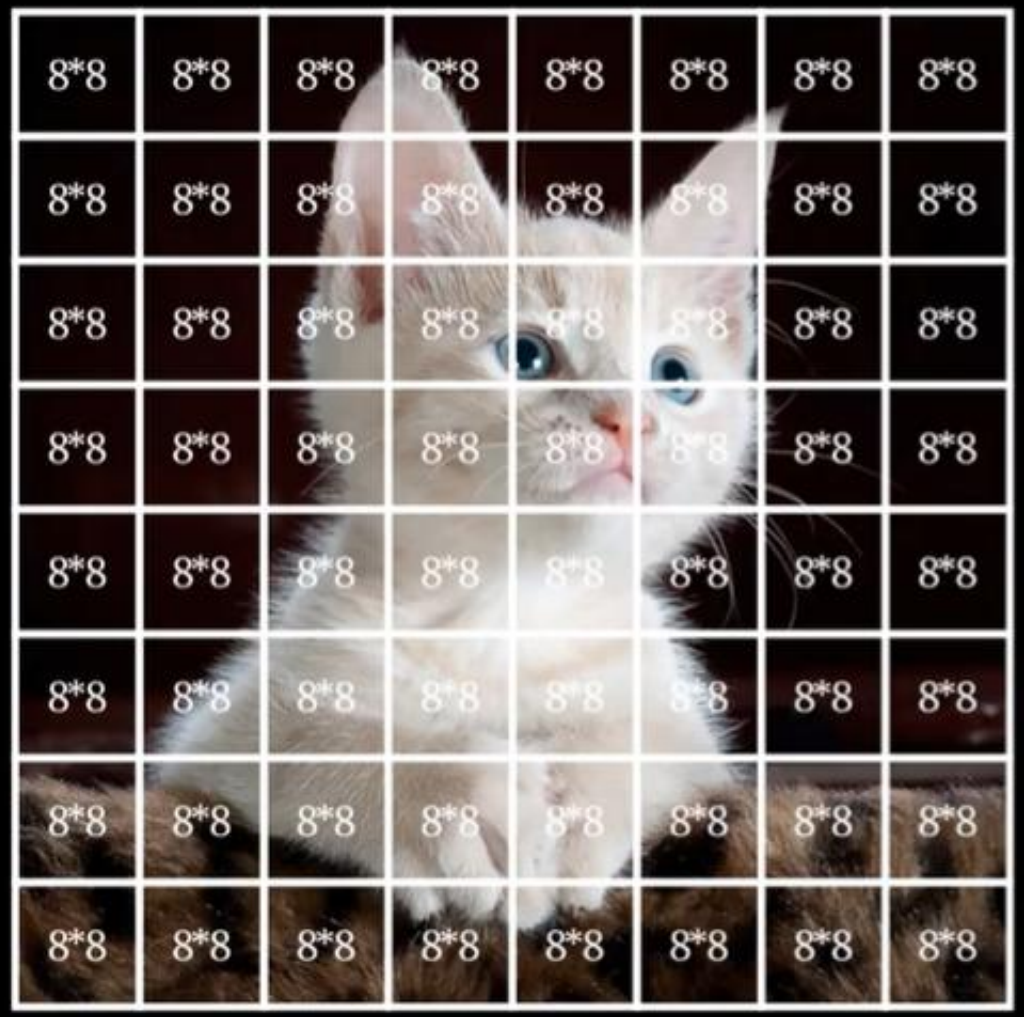
Vision Transfomer(ViT)





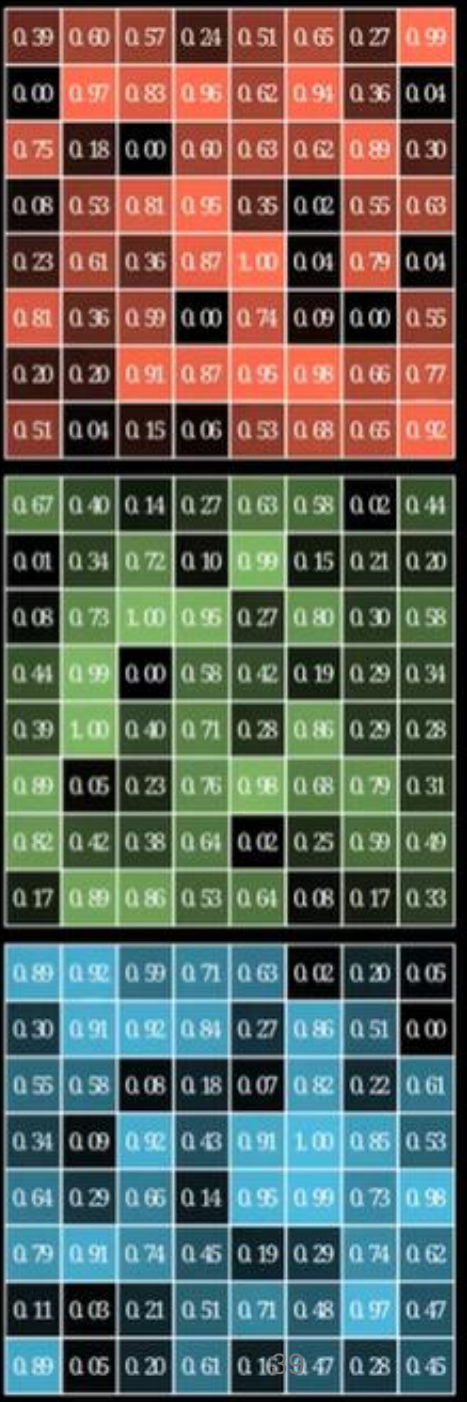


Vision Transfomer(ViT)

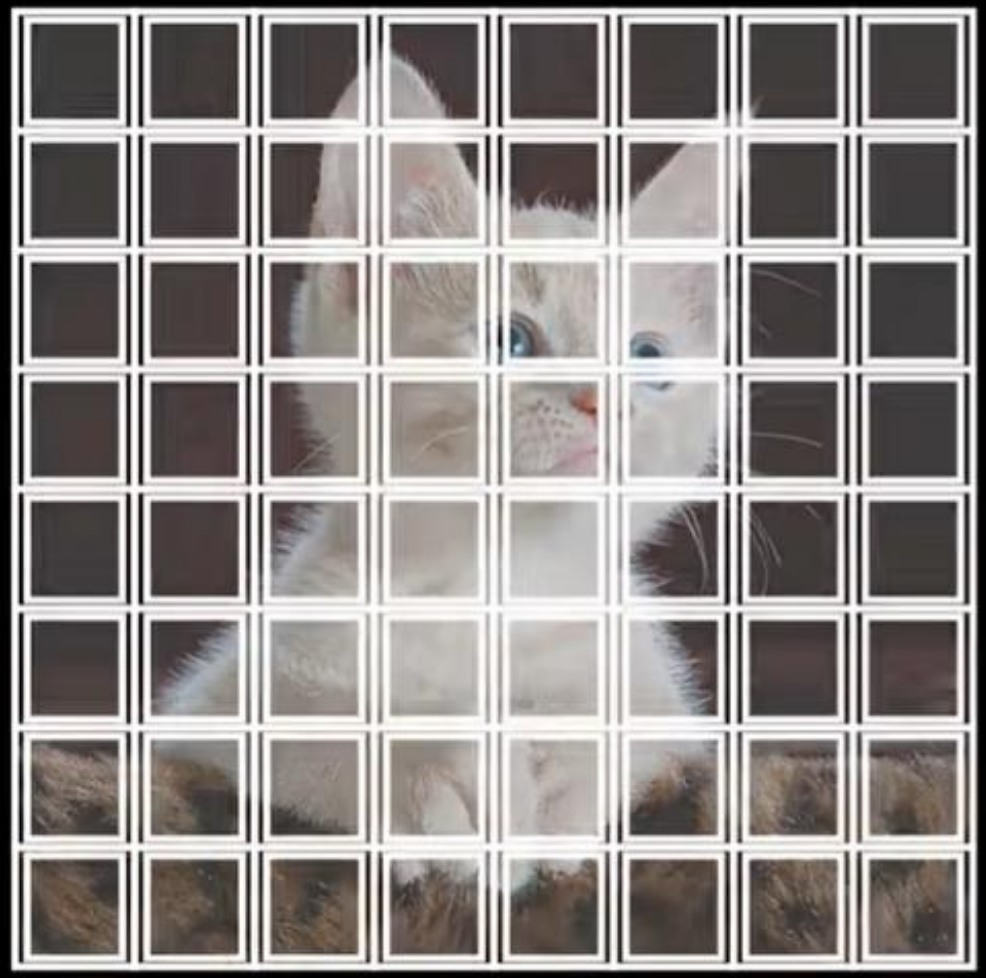


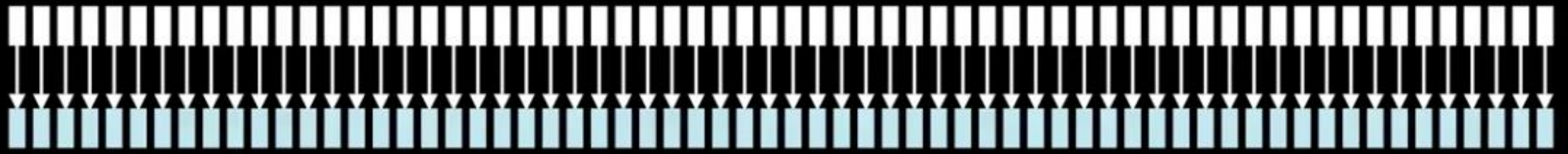
Patch size = 8*8 = 64

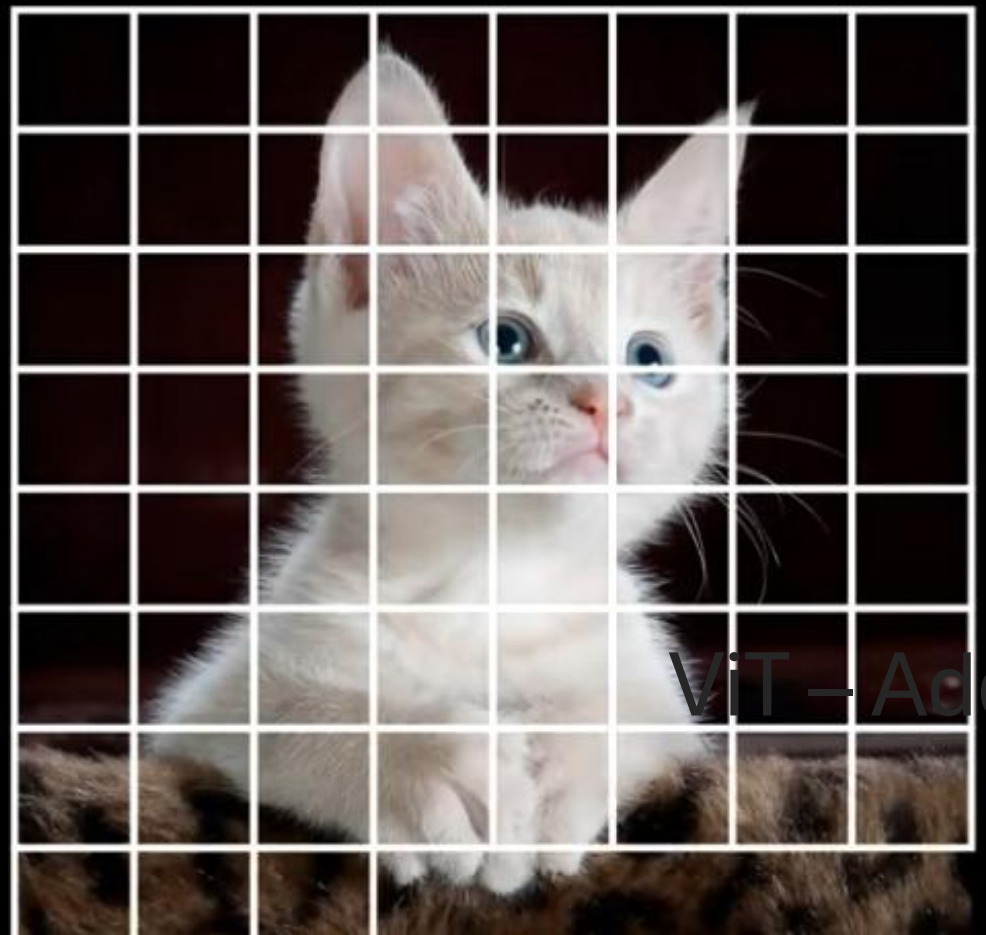
Total patches = 8*8 = 64

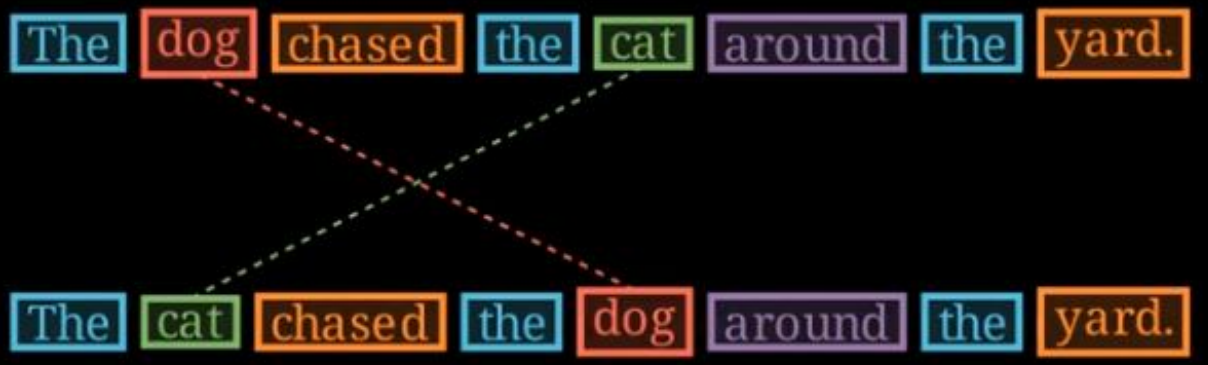


Vision Transfomer(ViT)



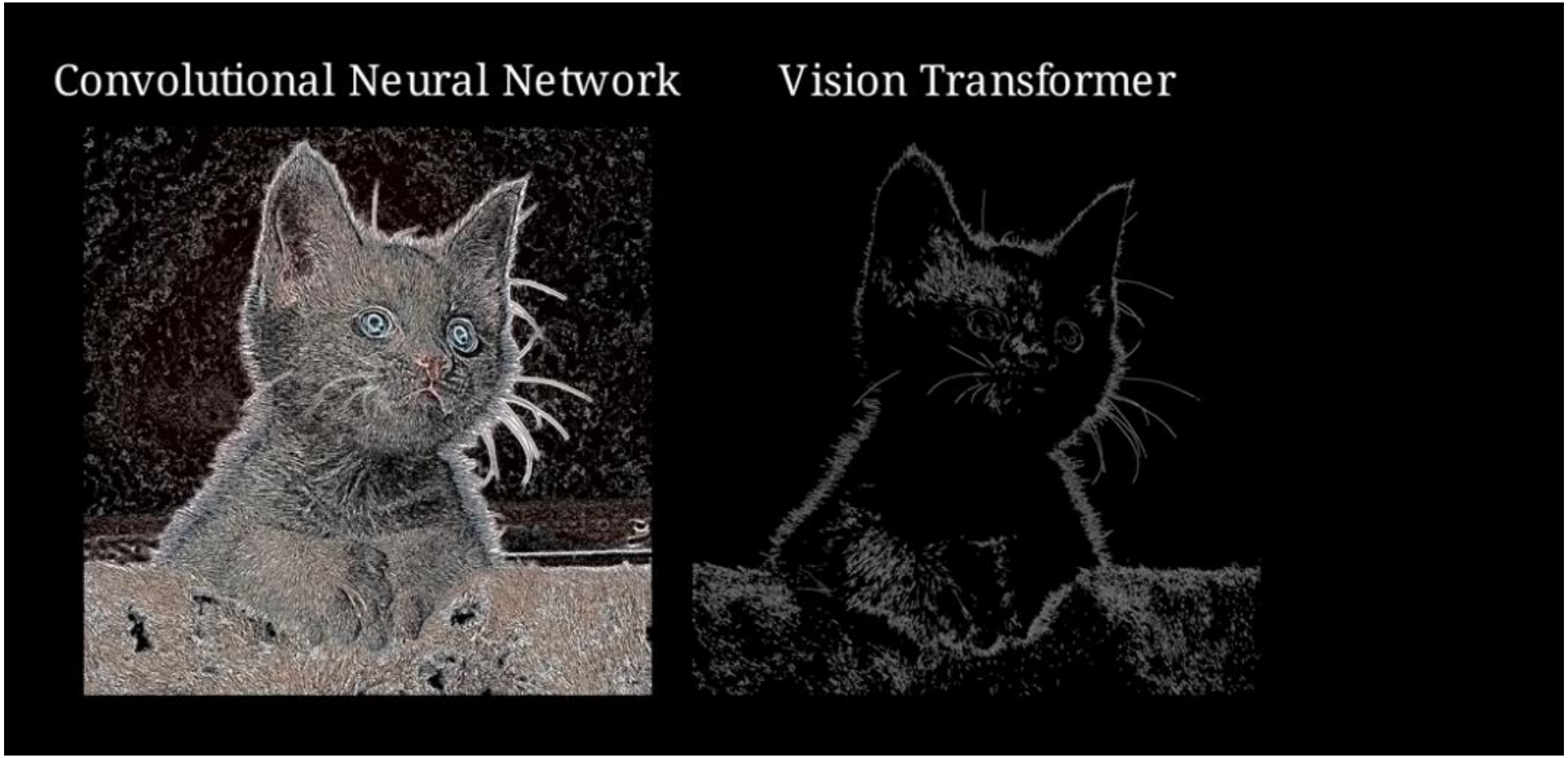






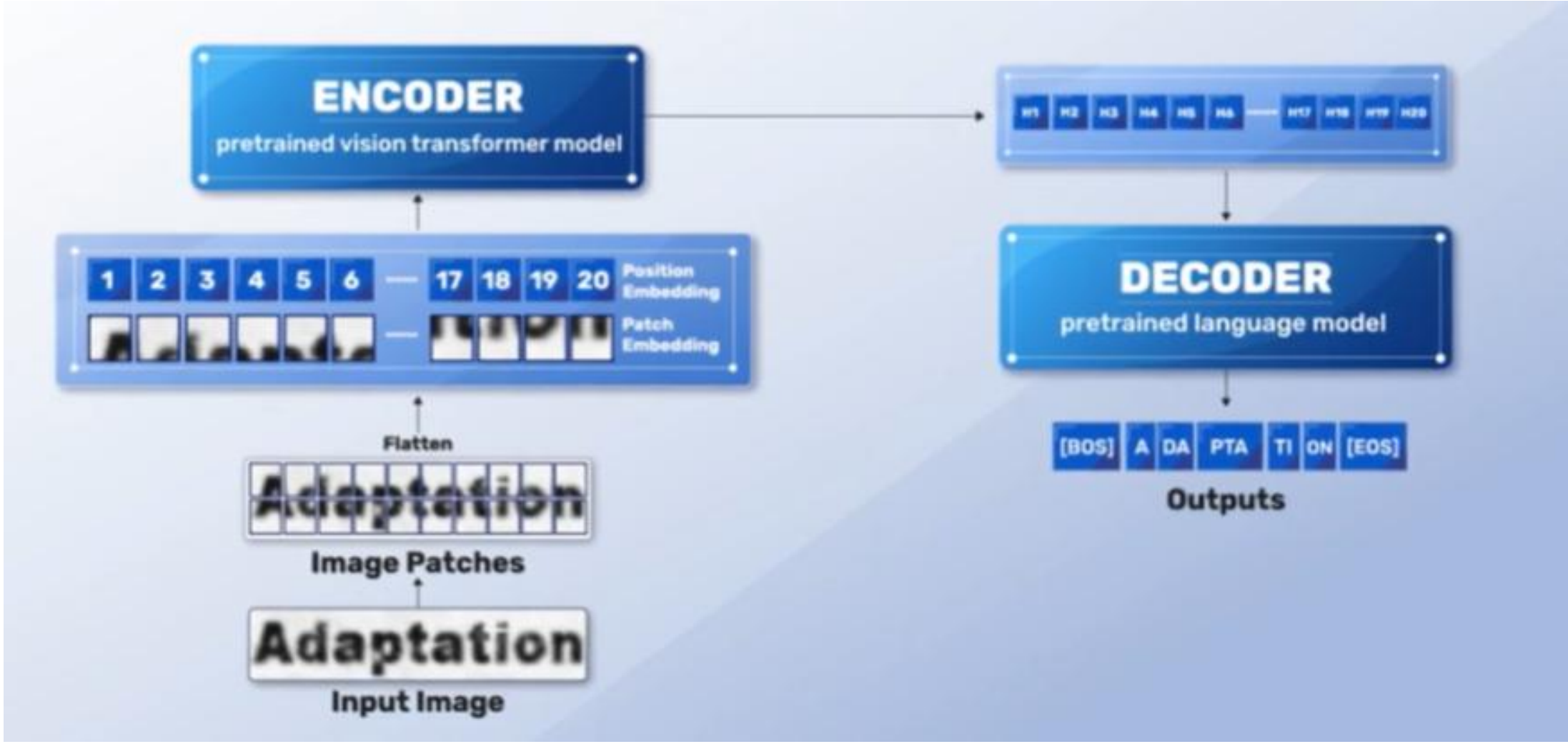
ViT – Add positional encoding

- CNN focuses on structure
- ViT focuses more on general features



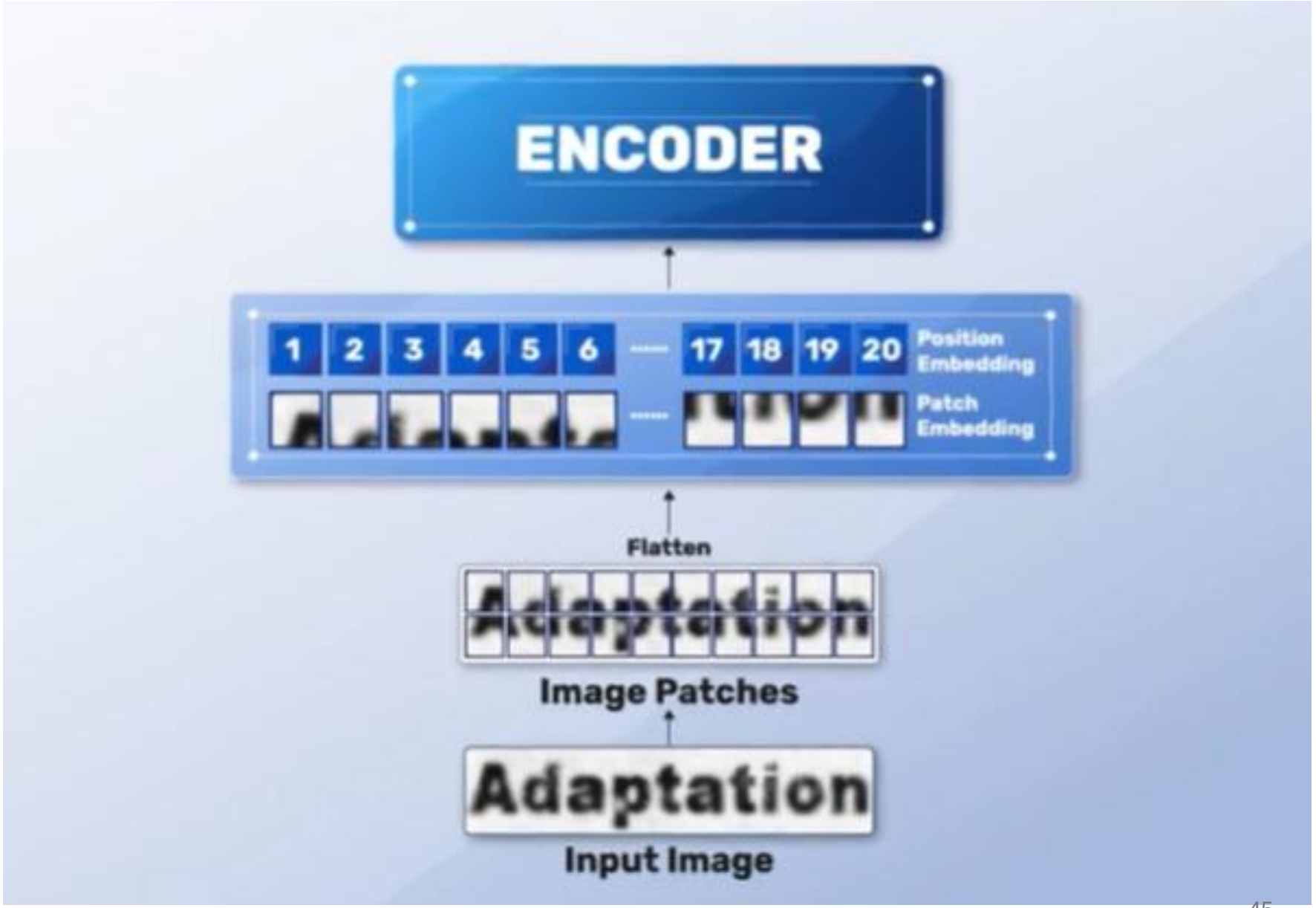
OCR

TrOCR - OCR Transformers

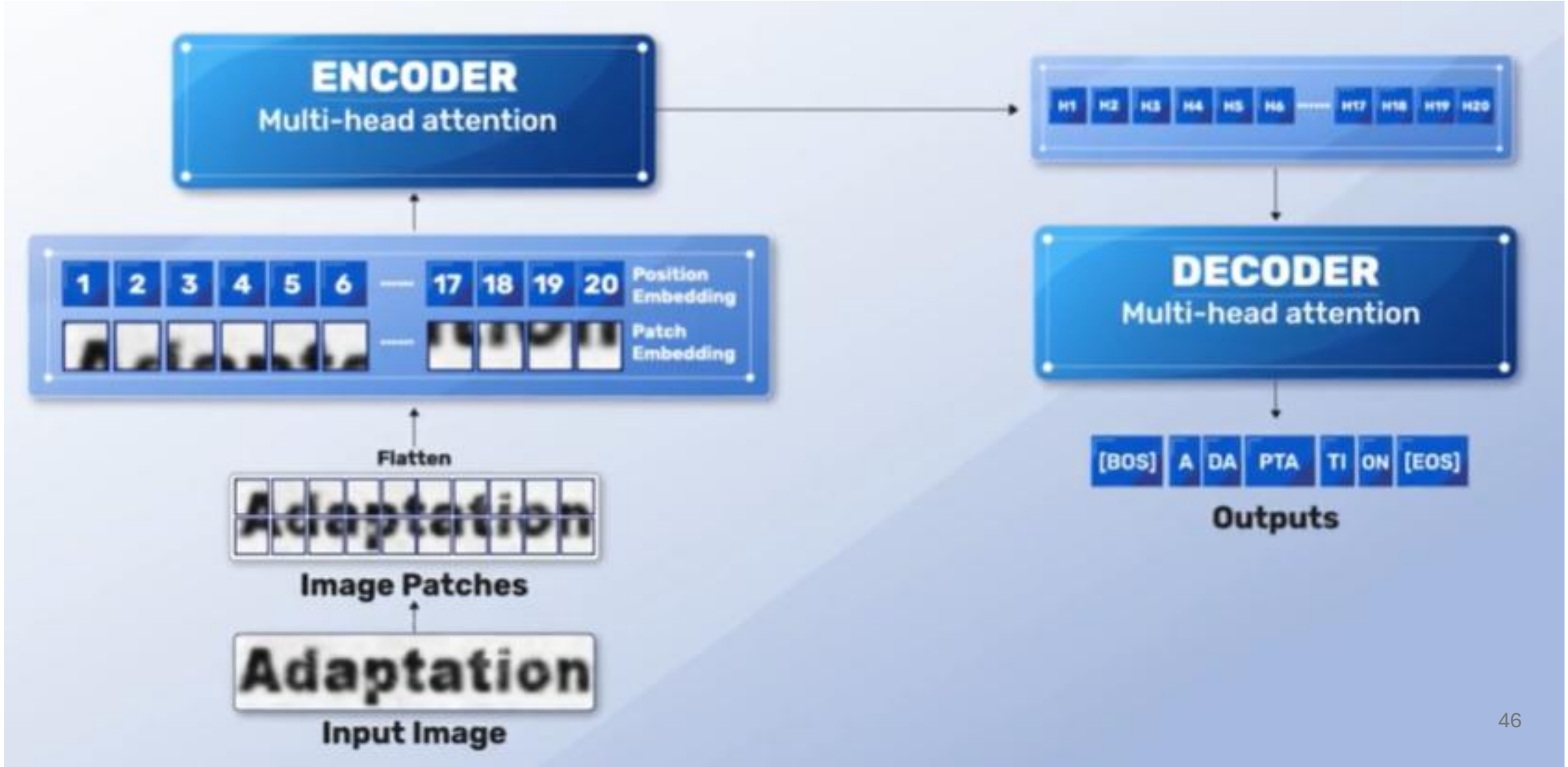


Credit: LearnOpenCV

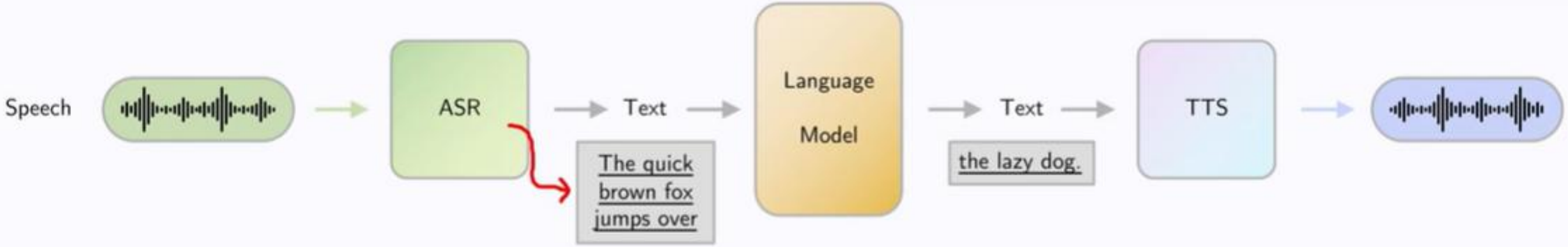
TrOCR - OCR Transformers



TrOCR - OCR Transformers



Cascading Models: Limitations



1 - Loss of information

"I can do that!"
"I can do that?!"

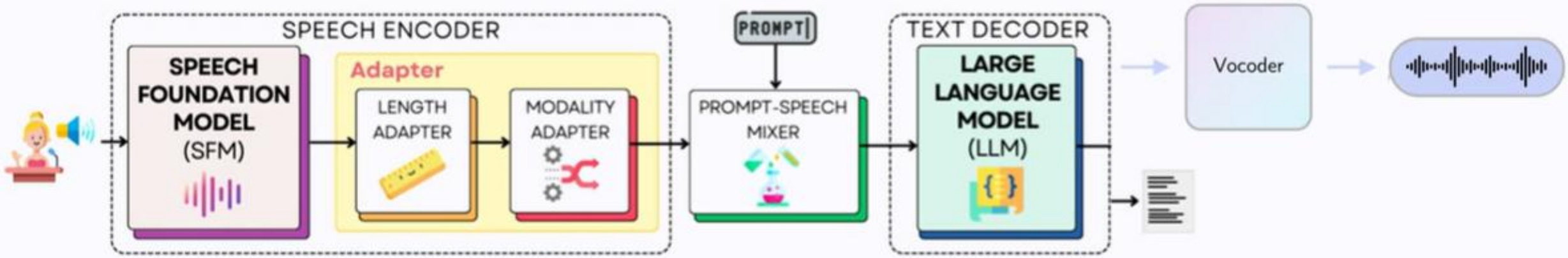
2 - Error propagation

I like beaches *I like peaches*

3 - High latency

Credit:Efficient NLP

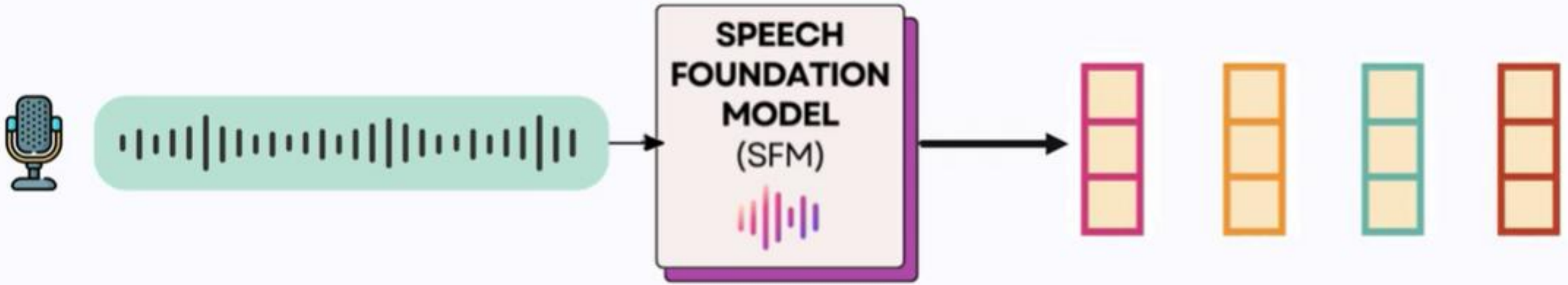
Components of a Speech LLM



1. Speech Encoder (instead of speech-to-text)
2. Large language model
3. Vocoder (instead of text-to-speech)

Credit:Efficient NLP

Speech Encoder

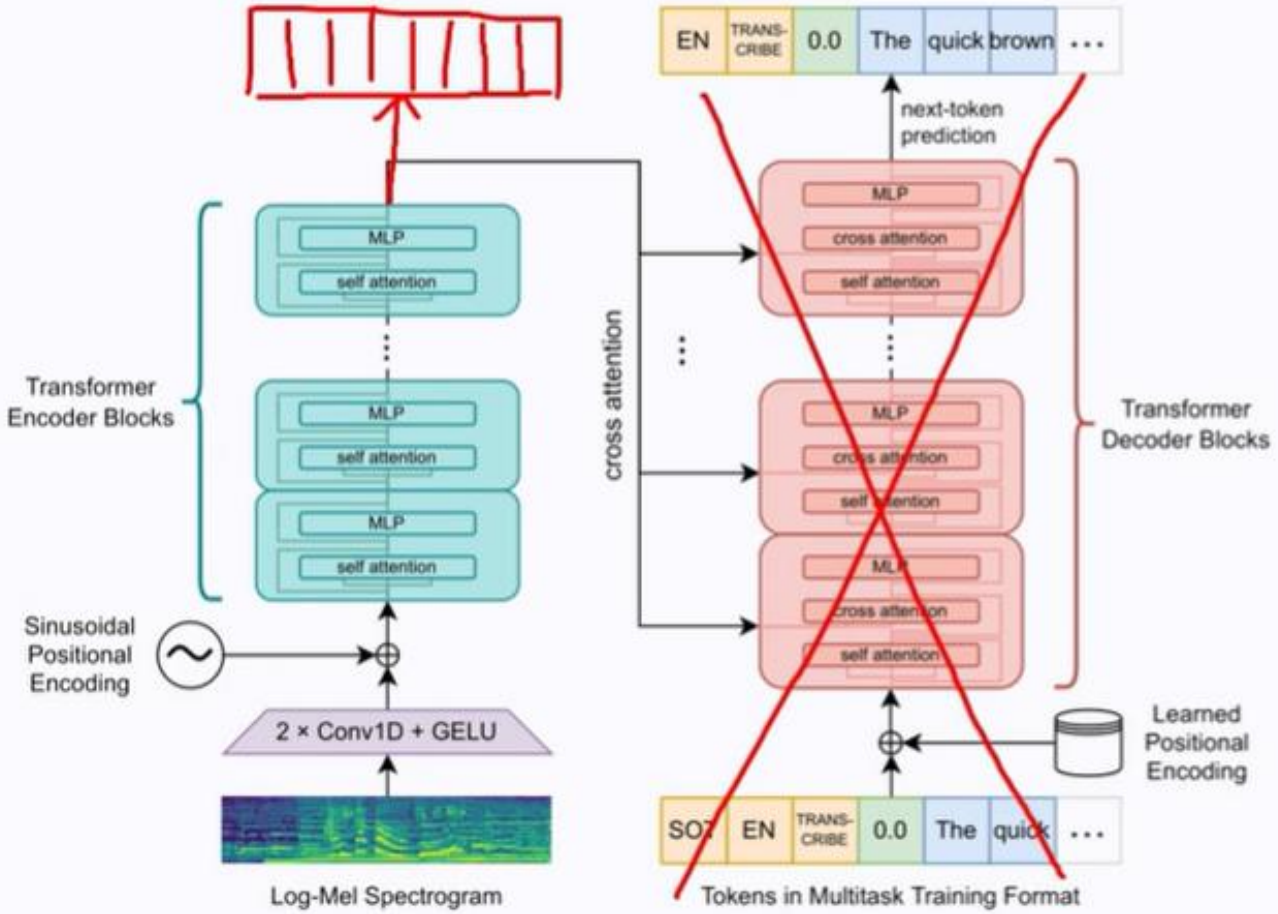


Semantic embeddings: capture meaning and content

Acoustic embeddings: capture intonation, sound quality

Credit:Efficient NLP

Speech Encoder



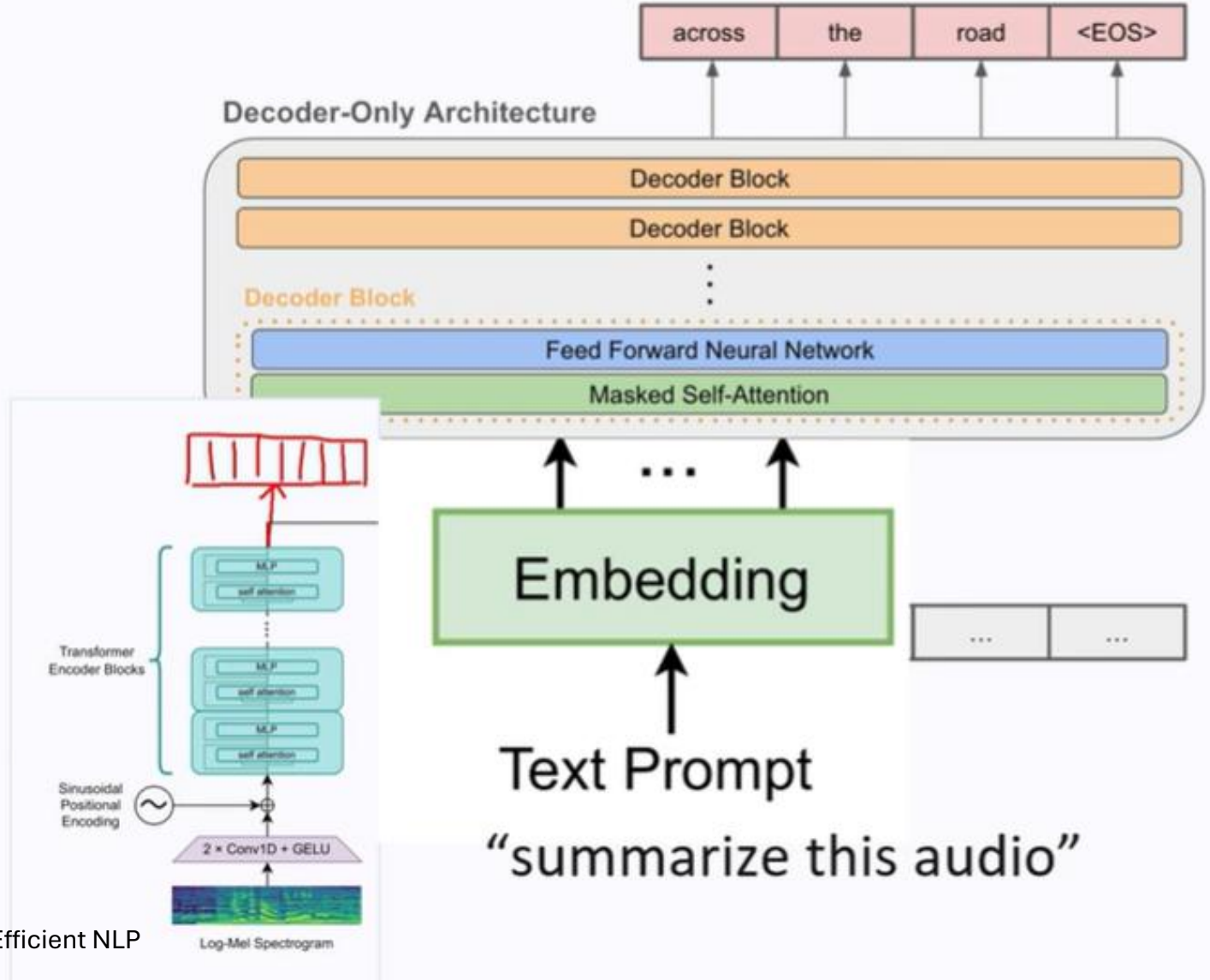
OpenAI Whisper: Encoder-decoder transformer for ASR

Other options:

- HuBERT
- Wav2vec 2.0
- W2V-BERT

Credit:Efficient NLP

Large Language Model (LLM)



LLaMA: LLMs from Meta, in various sizes (3B-70B)

Alternatives: Mistral, Vicuna, Qwen, etc

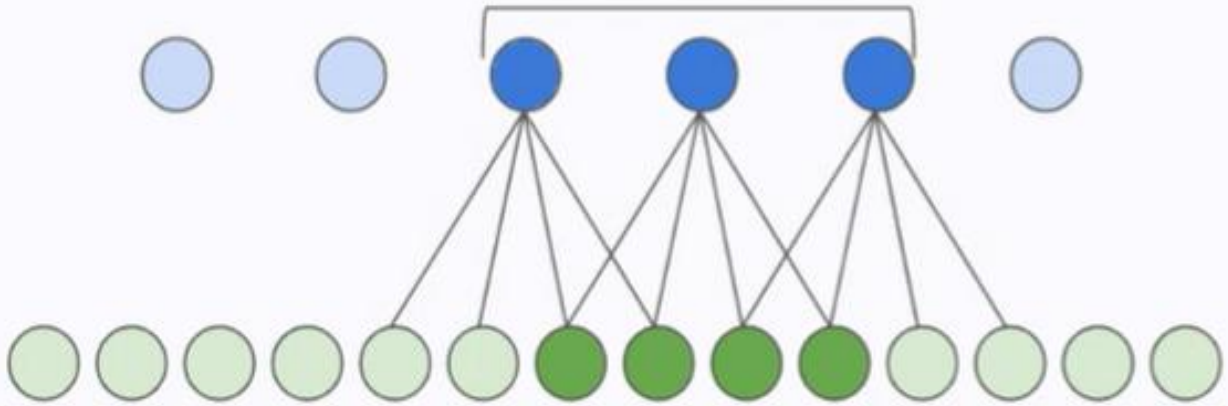
N/A: Proprietary LLMs (OpenAI GPT-4, Anthropic Claude, Gemini, etc)

Credit:Efficient NLP

Length Adaptation

1 minute of speech

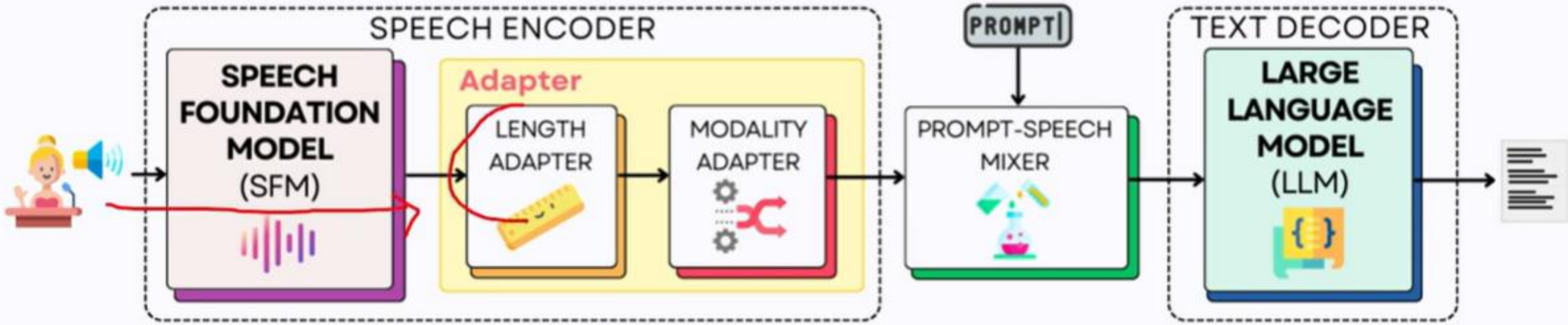
- Text: 150 words = 150 tokens = 150 input vectors
- Speech: 40 frames/sec = 2400 input vectors



- Downsampling Conv1D layer
- CTC (Connectionist Temporal Classification)
- Averaging consecutive frames

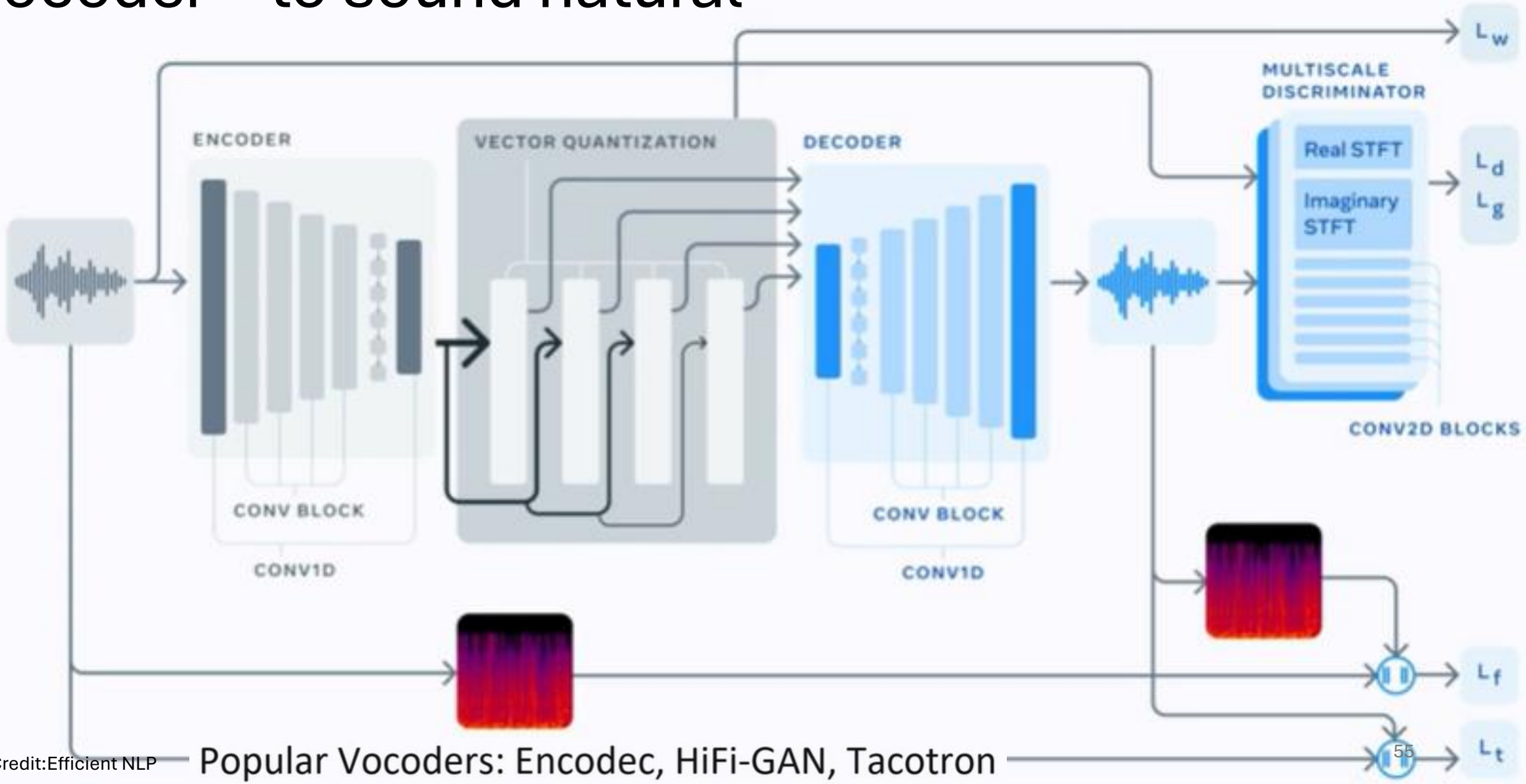
Credit:Efficient NLP

Speech LLM



Credit:Efficient NLP

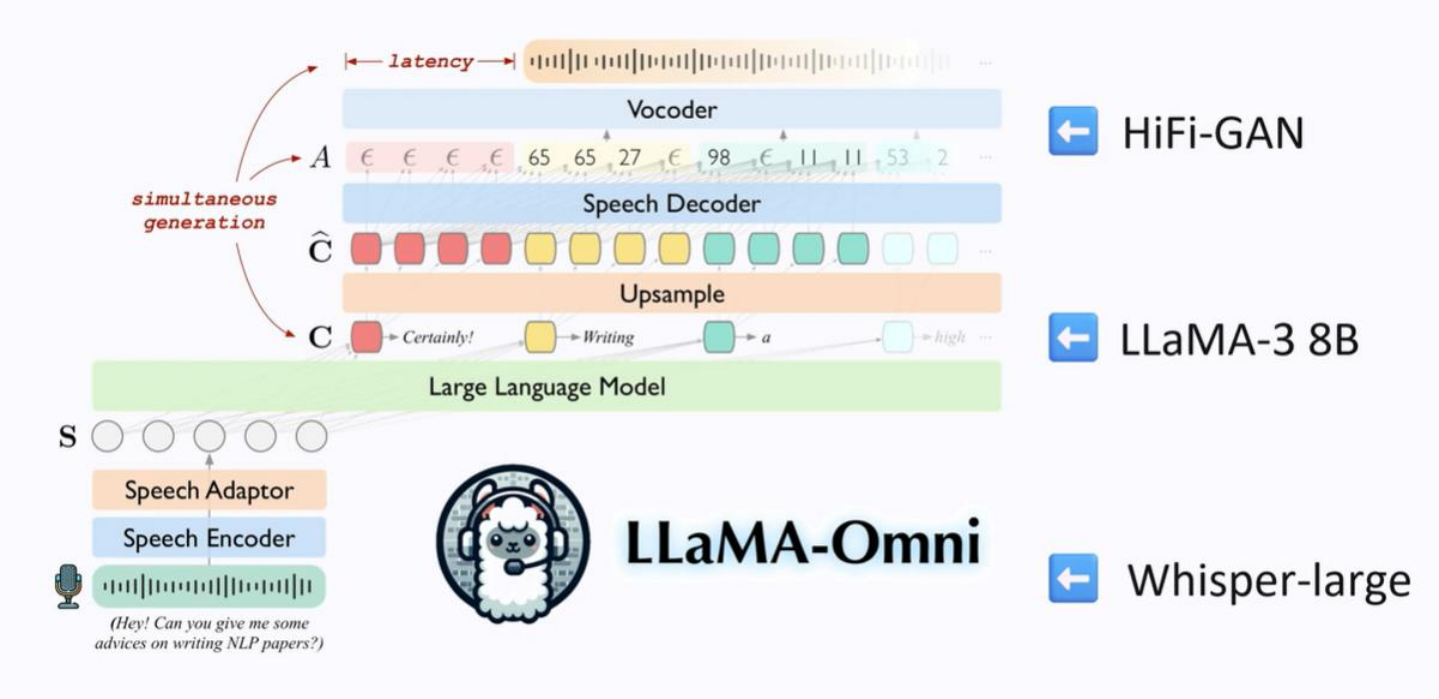
Vocoder – to sound natural



Popular Vocoders: Encodec, HiFi-GAN, Tacotron

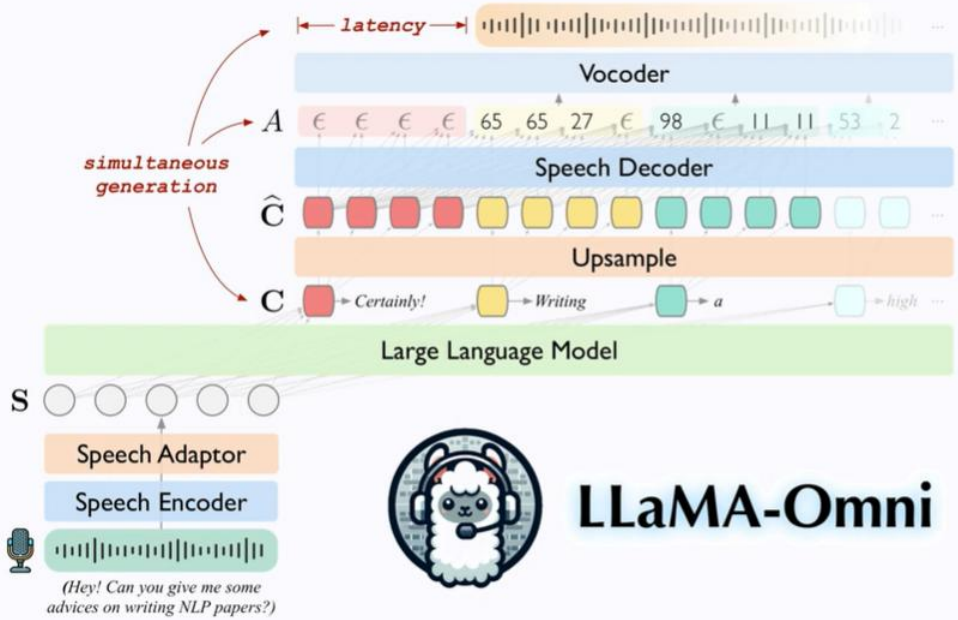
Credit:Efficient NLP

Speech LLM



Credit:Efficient NLP

Speech LLM



HiFi-GAN

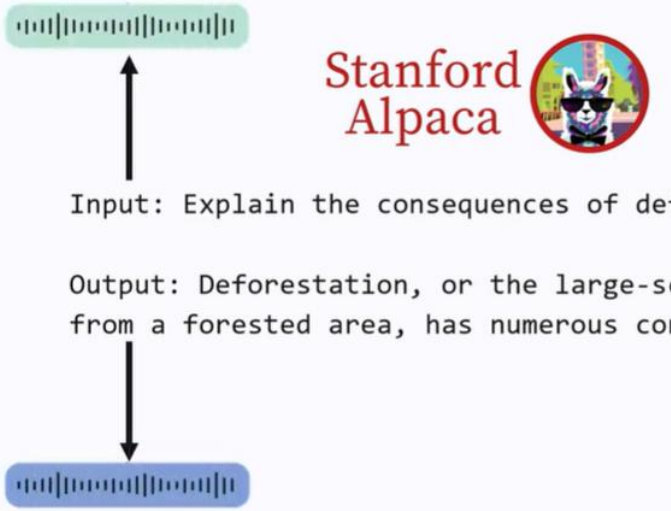
LLaMA-3 8B

Whisper-large

LLaMA-Omni

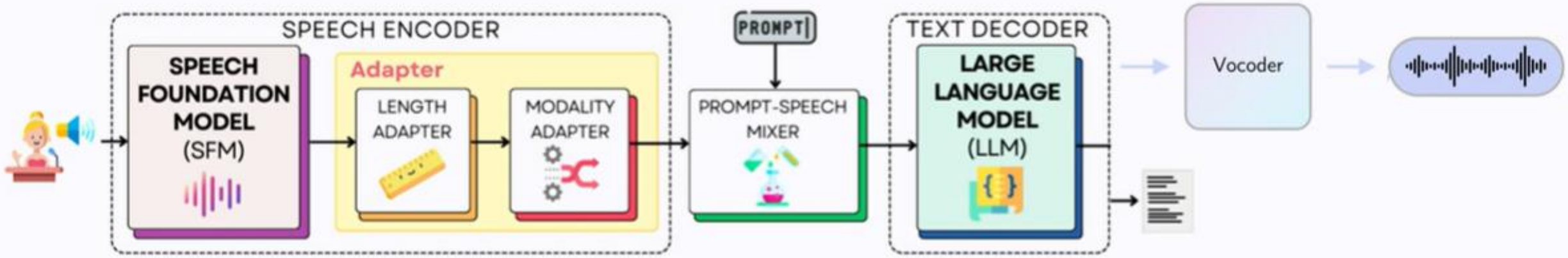
Fine-tune end-to-end

Optional: parameter efficient fine tuning (PEFT), LoRA



Credit:Efficient NLP

Components of a Speech LLM

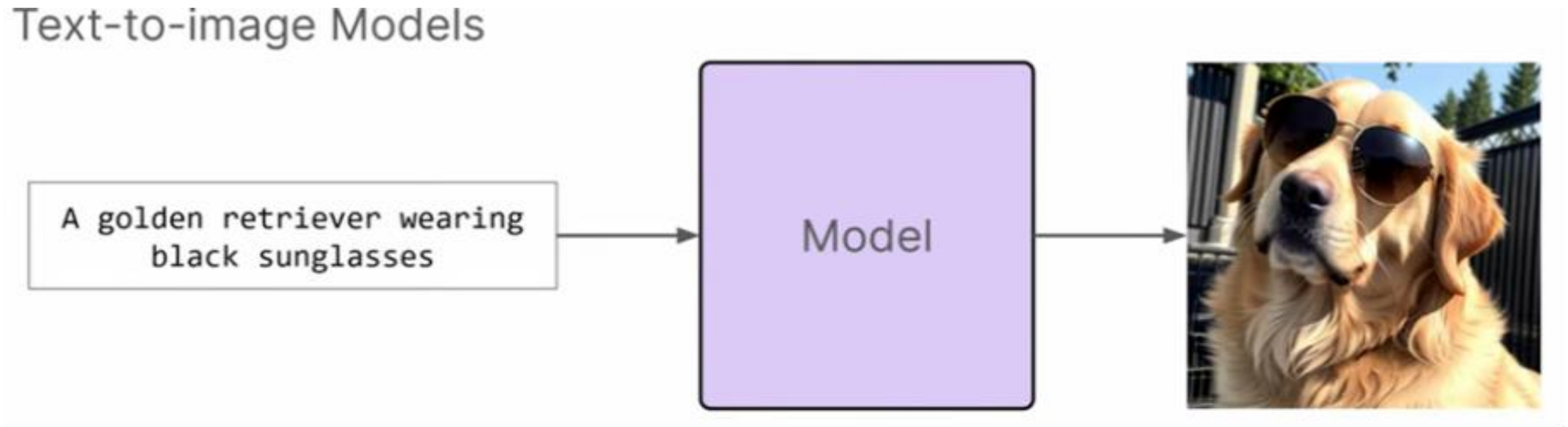


1. Speech Encoder (instead of speech-to-text)
2. Large language model
3. Vocoder (instead of text-to-speech)

Credit:Efficient NLP

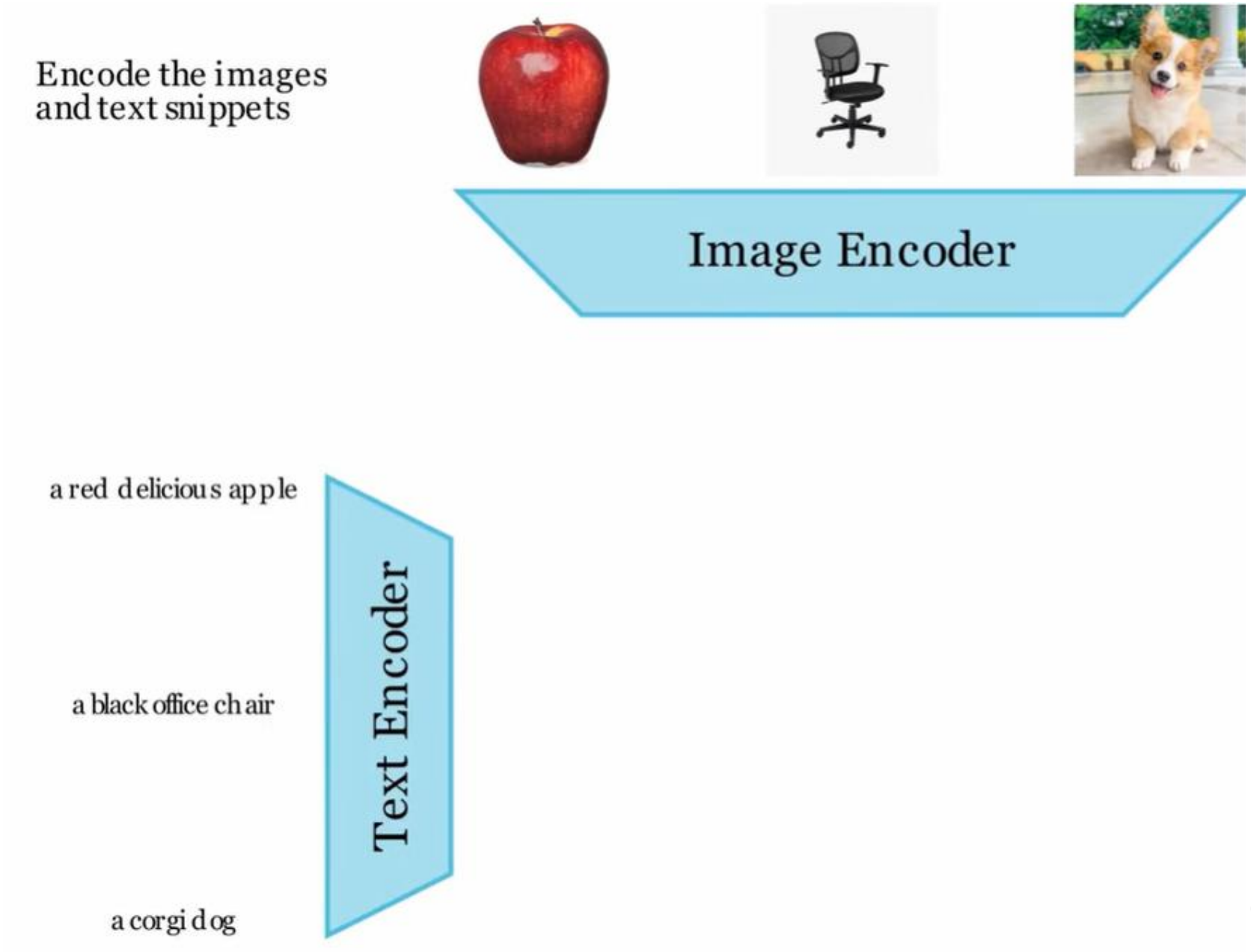

AI

Multimodal Models



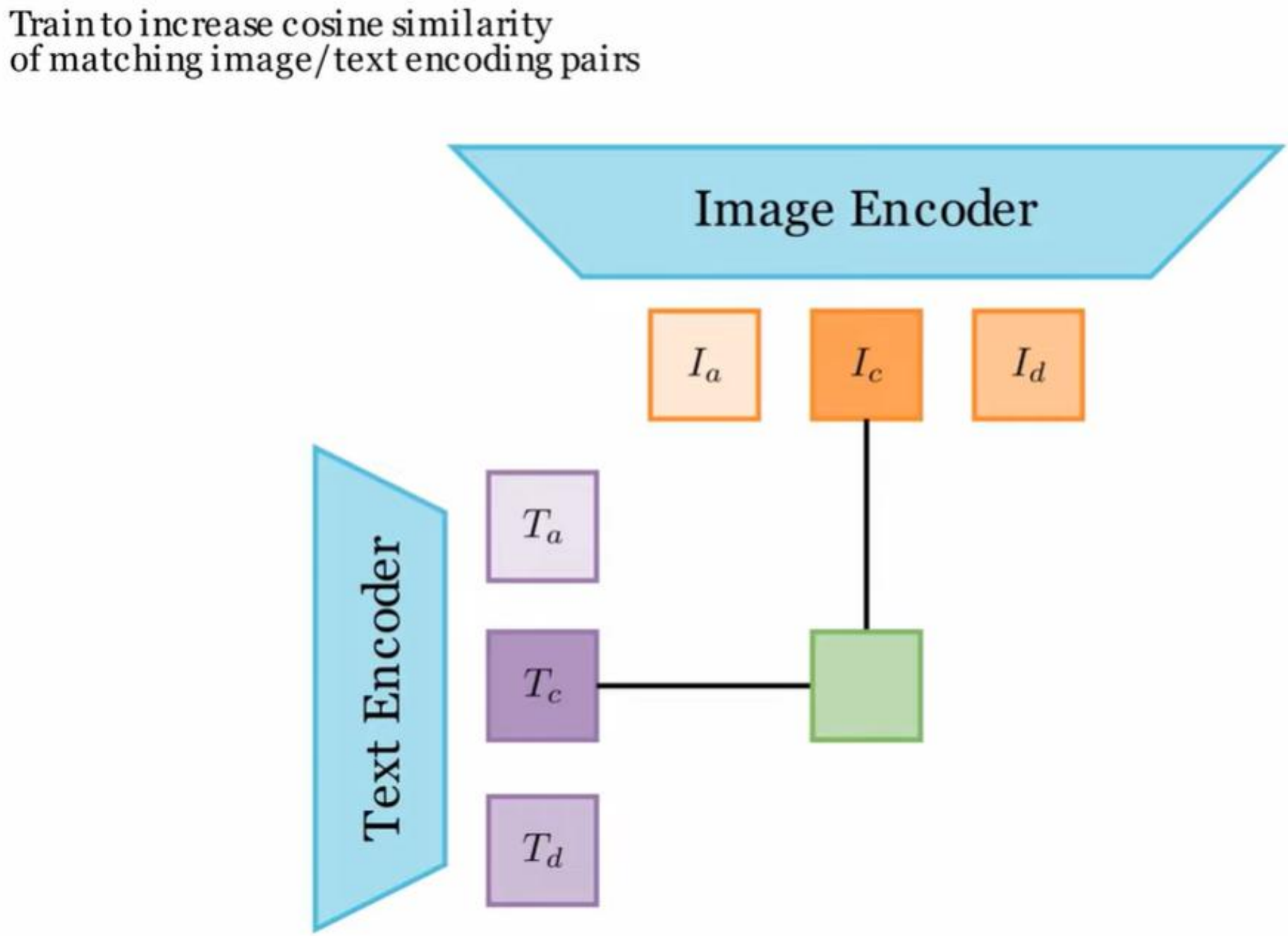
Credit: AssemblyAI

Multimodal Models



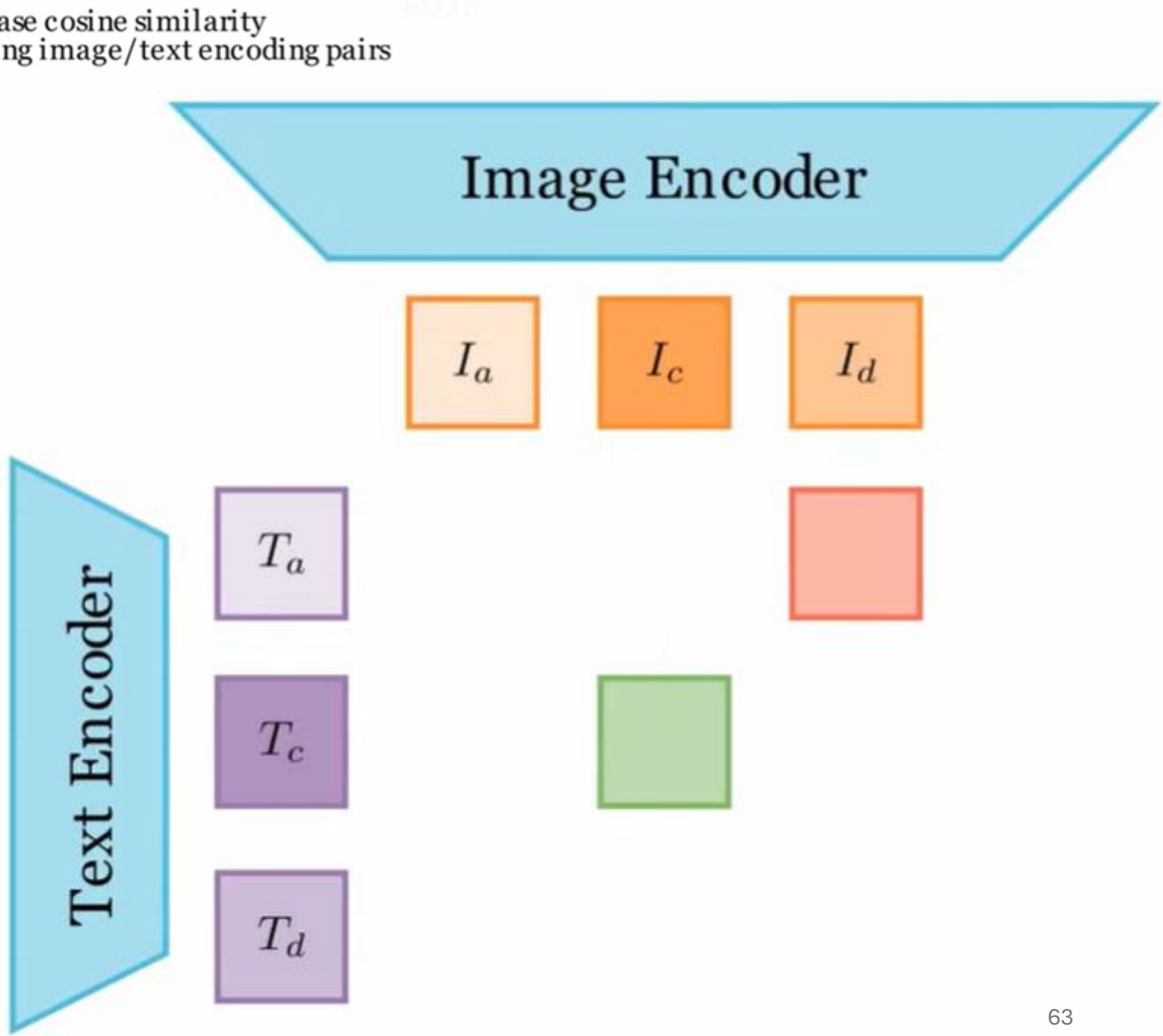
Credit: AssemblyAI

Multimodal Models



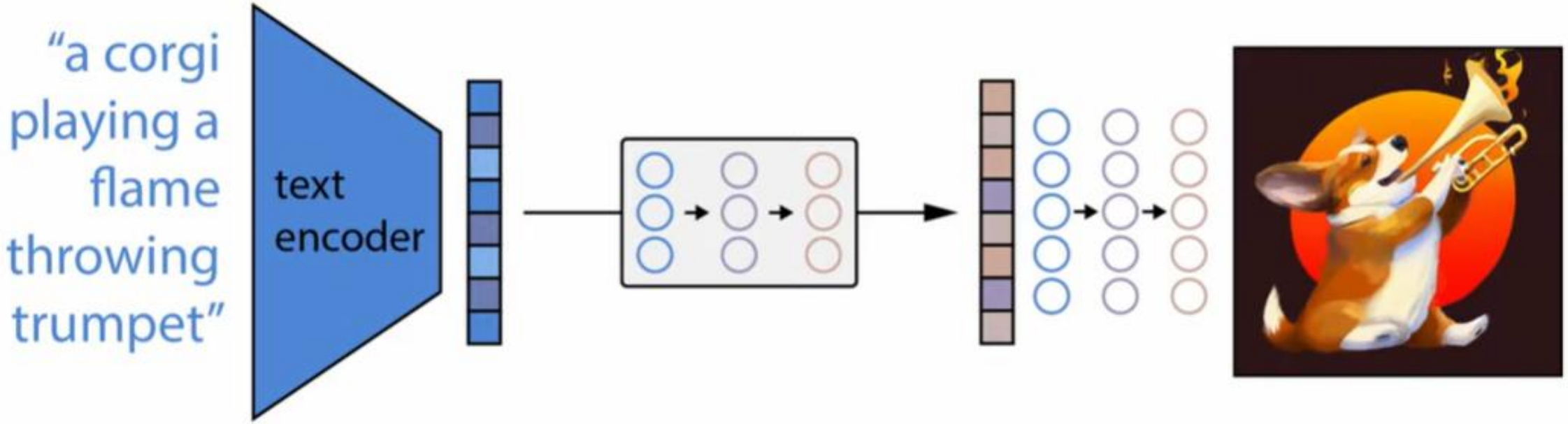
Credit: AssemblyAI

Multimodal Models



Credit: AssemblyAI

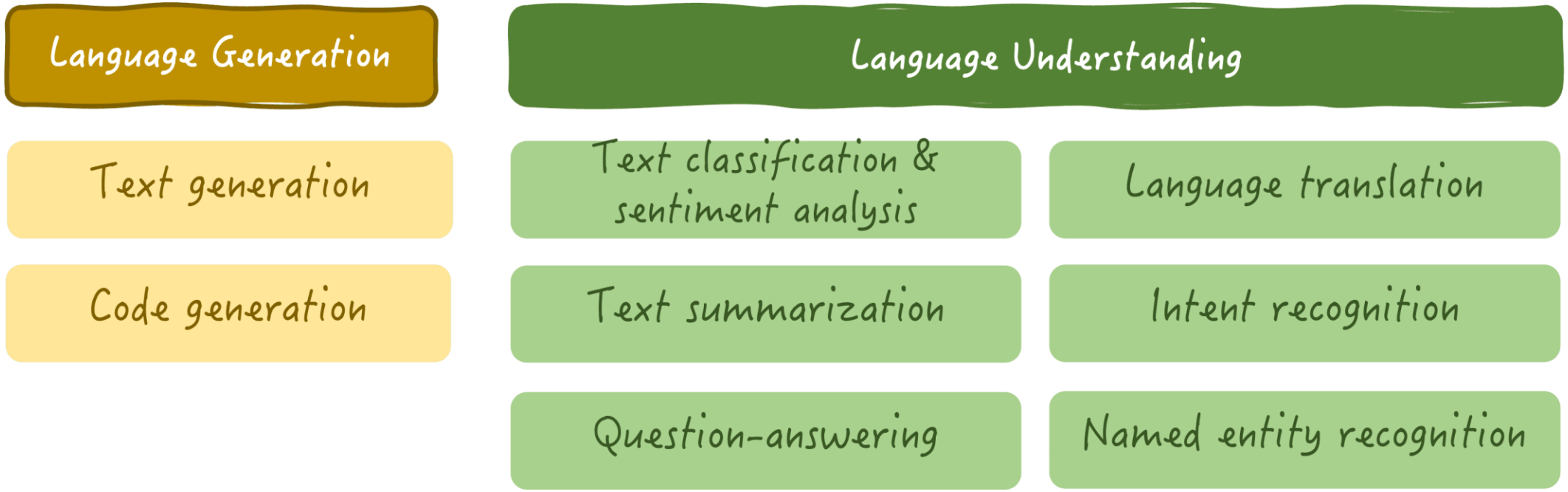
Multimodal Models



Encode the meaning of the text and decode to an image

Credit: AssemblyAI

LLM frequent tasks



Problem with classic IR

- The **vocabulary mismatch problem**
- tf-idf or BM25 cosine similarities only work if there is **exact word overlap** between query and doc!
- But query-writer can't know the exact words the doc might include!

Dense retrieval

- Instead of representing query and documents with count vectors
- Represent both with embeddings!

Hypothetical version of dense retrieval: static embeddings

Replace tf-idf vectors with, Static Word Embeddings

- Query: the mean of the embeddings of each query word
- Doc: the mean of the doc word embeddings
- Now just compute query-doc cosine as normal.

We don't do this because **contextual embeddings** work much better!

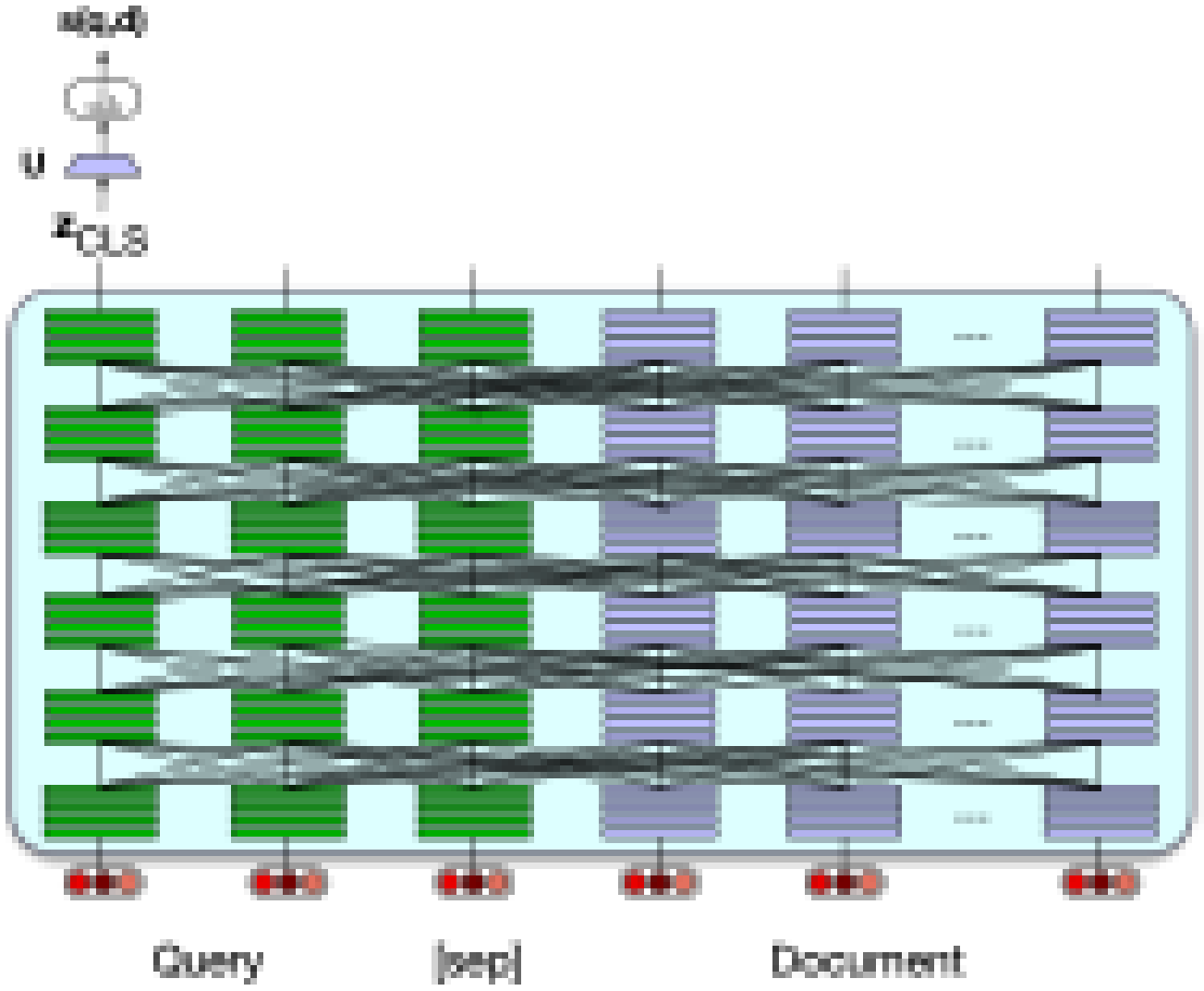
Dense IR – Structural Problems

- QA - How do we produce the answer? Could we have a conversation?
- Vectors couldn’t/shouldn’t represent full document
- How do we find the vector representation and similar vectors?
- How do we rank the results?

Dense IR – Structural Problems

- QA - How do we produce the answer? Could we have a conversation?
  - Solution: Use a dedicated LLM that is trained to do that
- Vectors couldn’t/shouldn’t represent full document
  - Solution: Chucking (will be discussed later)
- How do we find the vector representation and similar vectors?
  - Solution: VectorDB (will be discussed later)
- How do we rank the results?
  - Solution: Train the LLM to produce a score

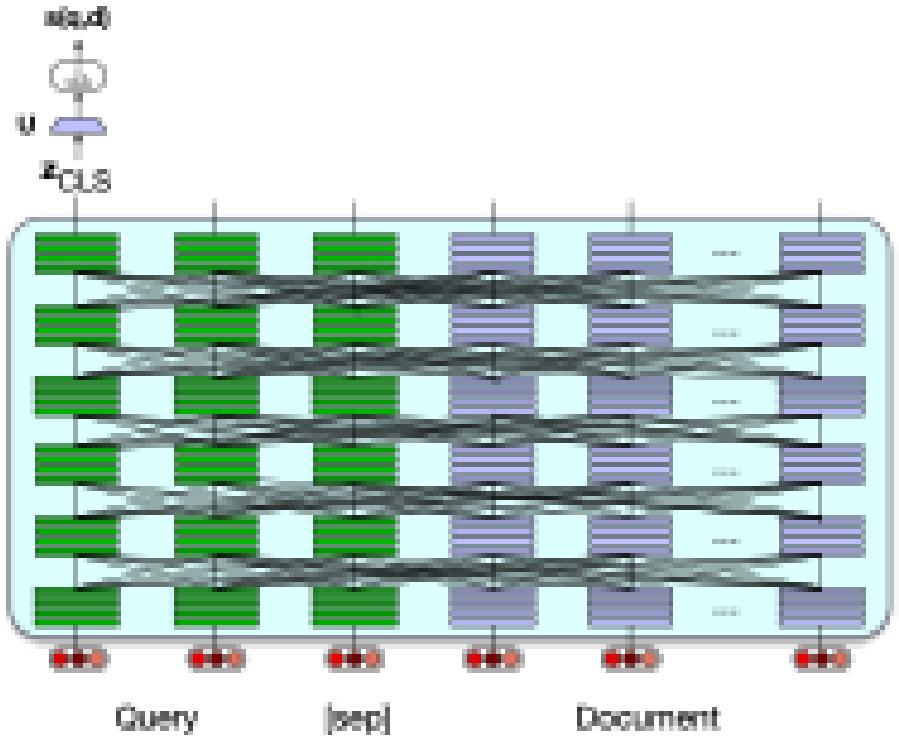
Dense retrieval #1: Single encoder



$$\mathbf{z} = \text{BERT}(q; [\text{SEP}]; d)[\text{CLS}]$$

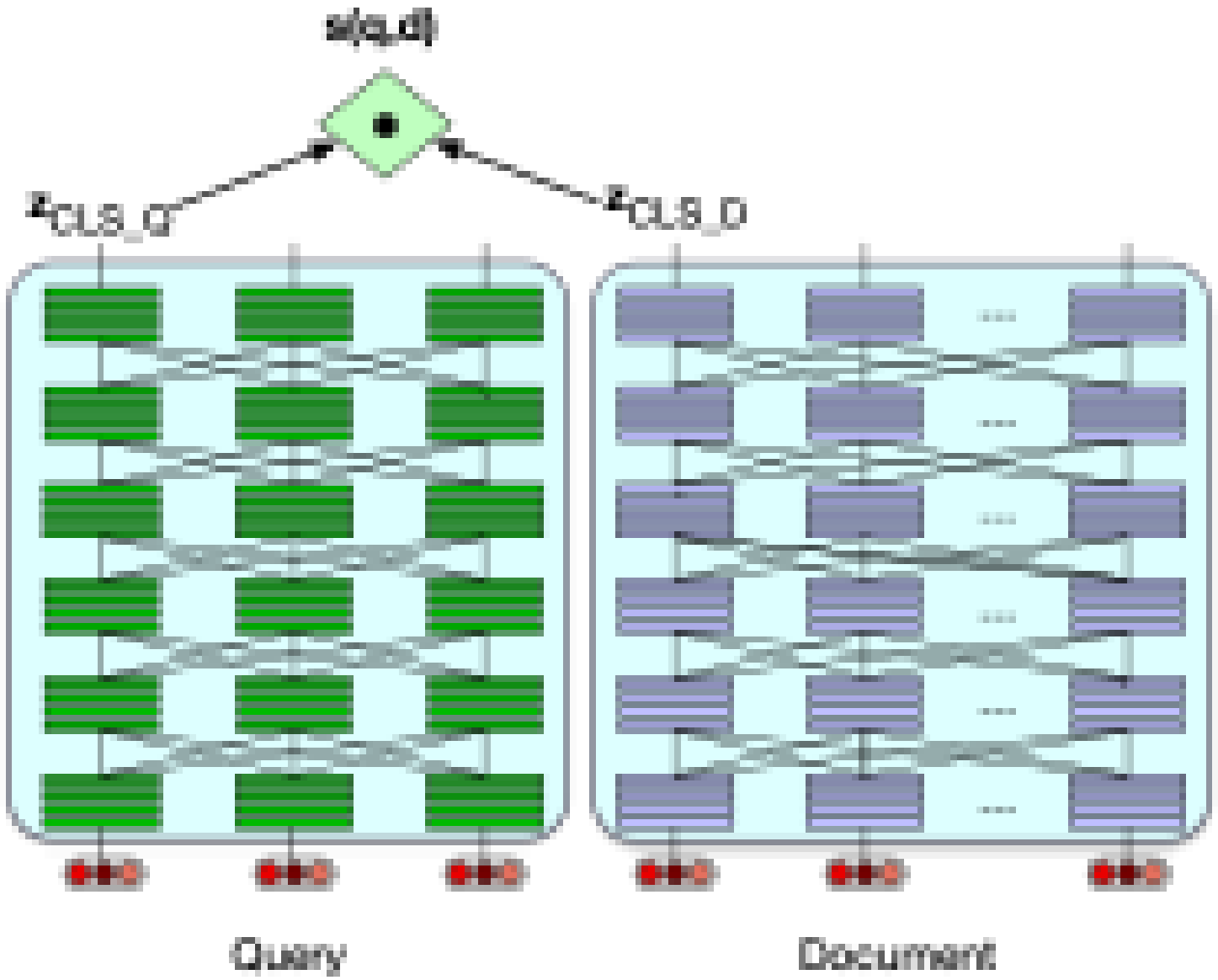
$$\text{score}(q, d) = \text{softmax}(\mathbf{U}(\mathbf{z}))$$

Dense retrieval #1: Single encoder



- System is run on passages (say 100 tokens) instead of whole documents
- **Training:**
- BERT and linear layer U can then fine-tuned for relevance
- Creating a tuning dataset of relevant and non-relevant passages.

Dense retrieval #2 (**biencoder**)

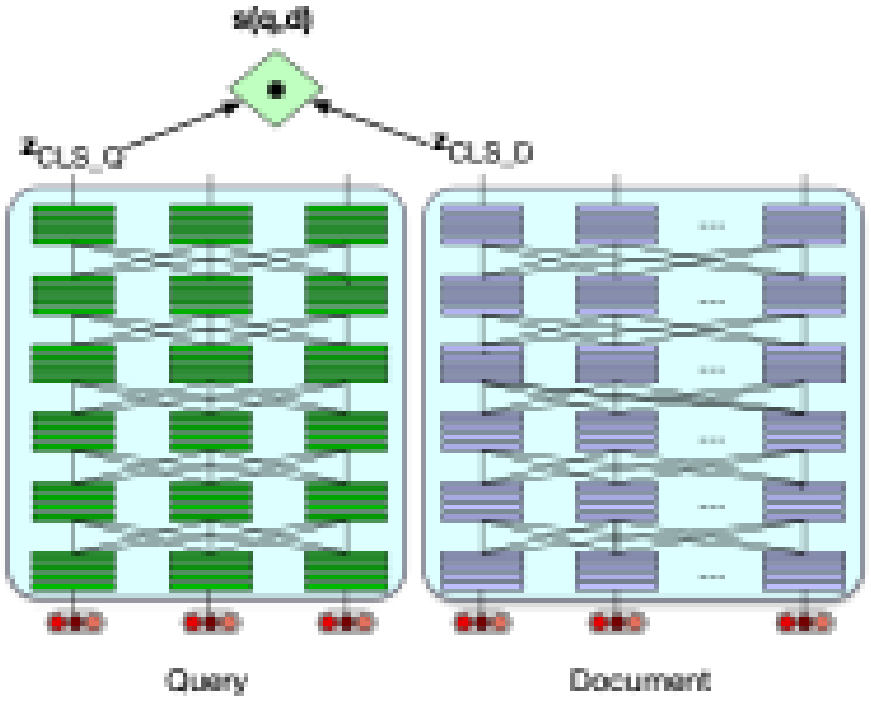


$$\mathbf{z}_q = \text{BERT}_Q(\mathbf{q})[\text{CLS}]$$

$$\mathbf{z}_d = \text{BERT}_D(\mathbf{d})[\text{CLS}]$$

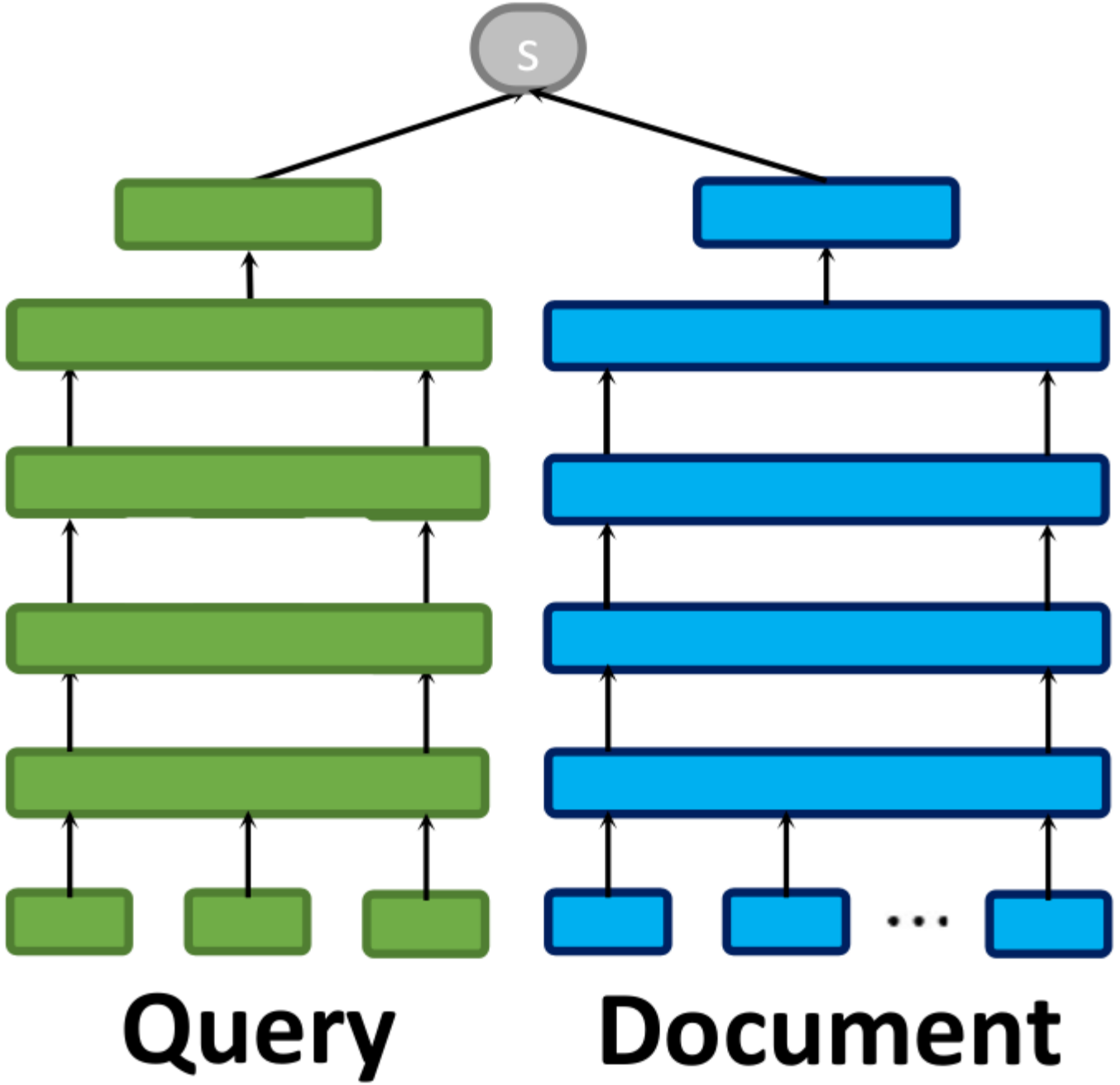
$$\text{score}(q,d) = \mathbf{z}_q \cdot \mathbf{z}_d$$

Dense retrieval #2 (**biencoder**)

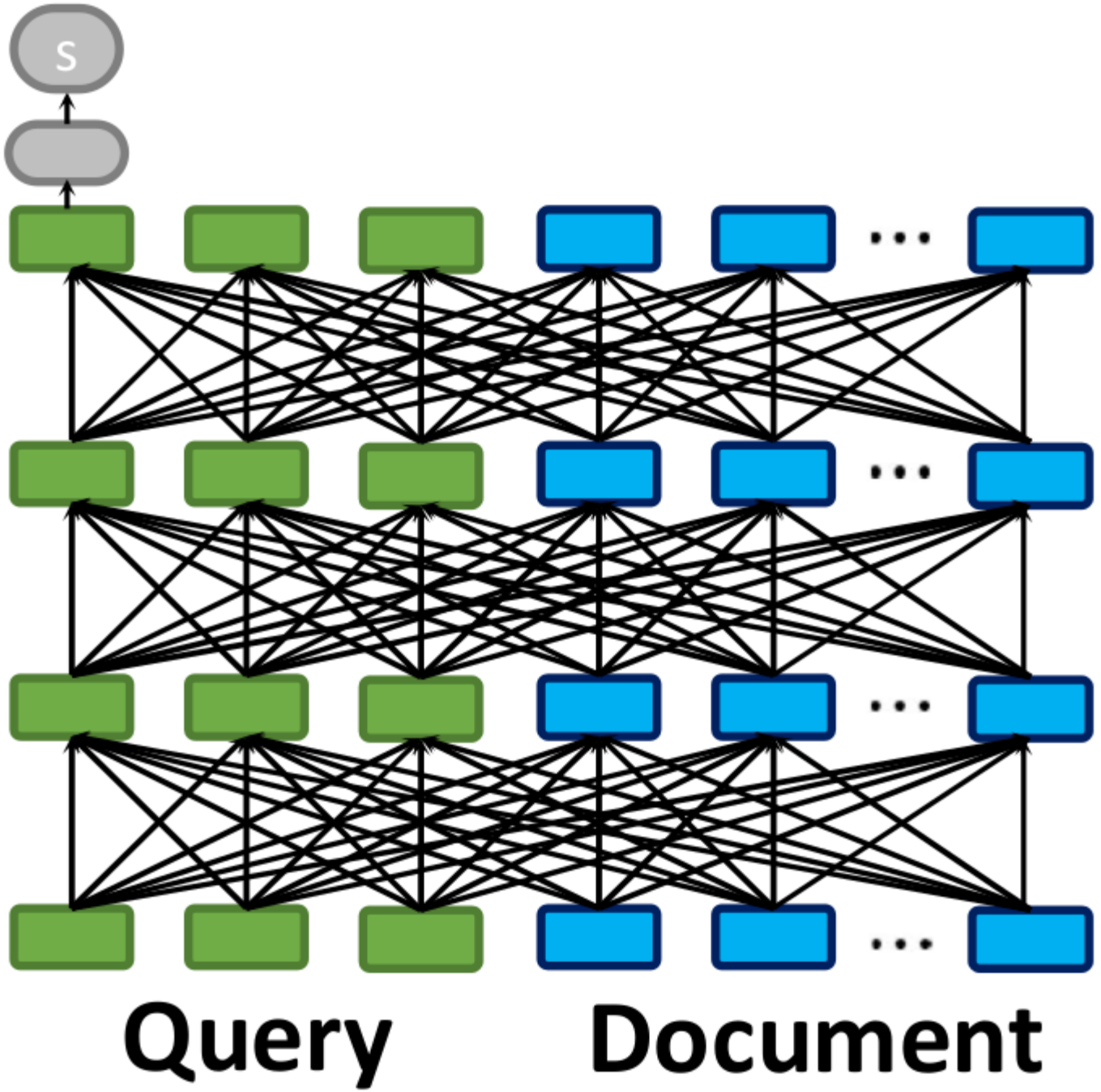


- Encode doc vectors in advance.
- Encode query when it arrives
- Score is dot product between query vector and precomputed doc vector
- Cheaper but less accurate

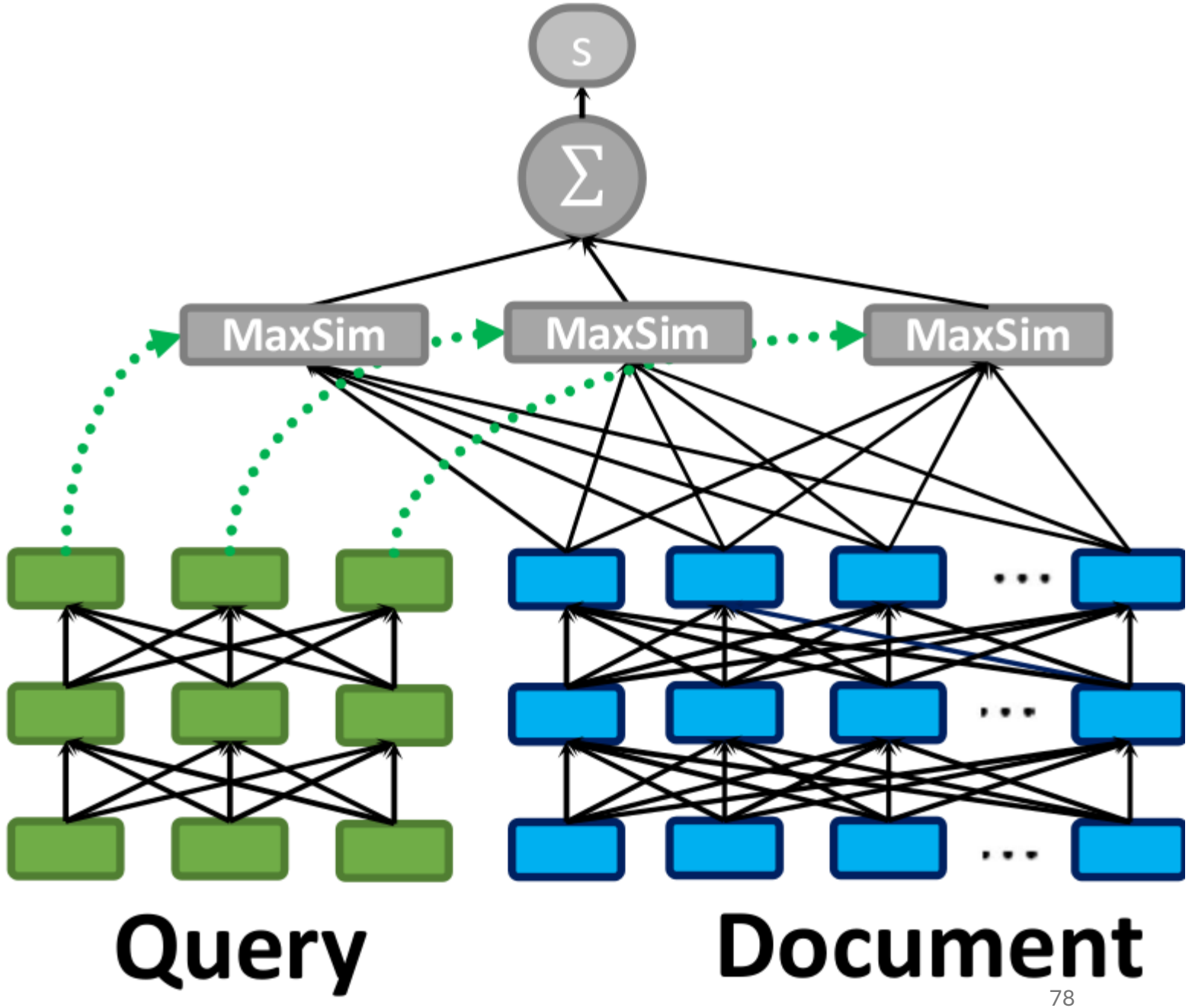
Bi-Encoders

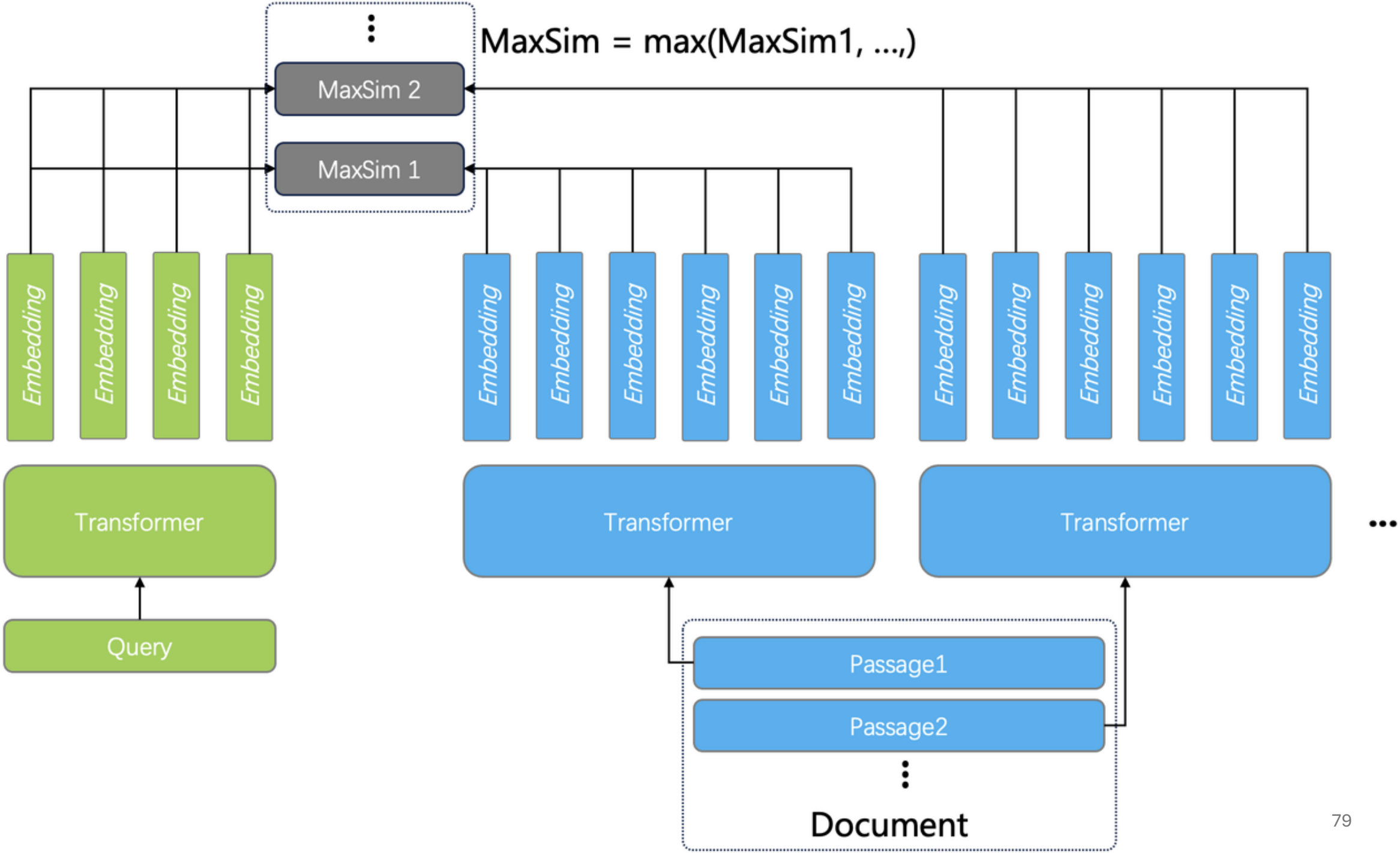


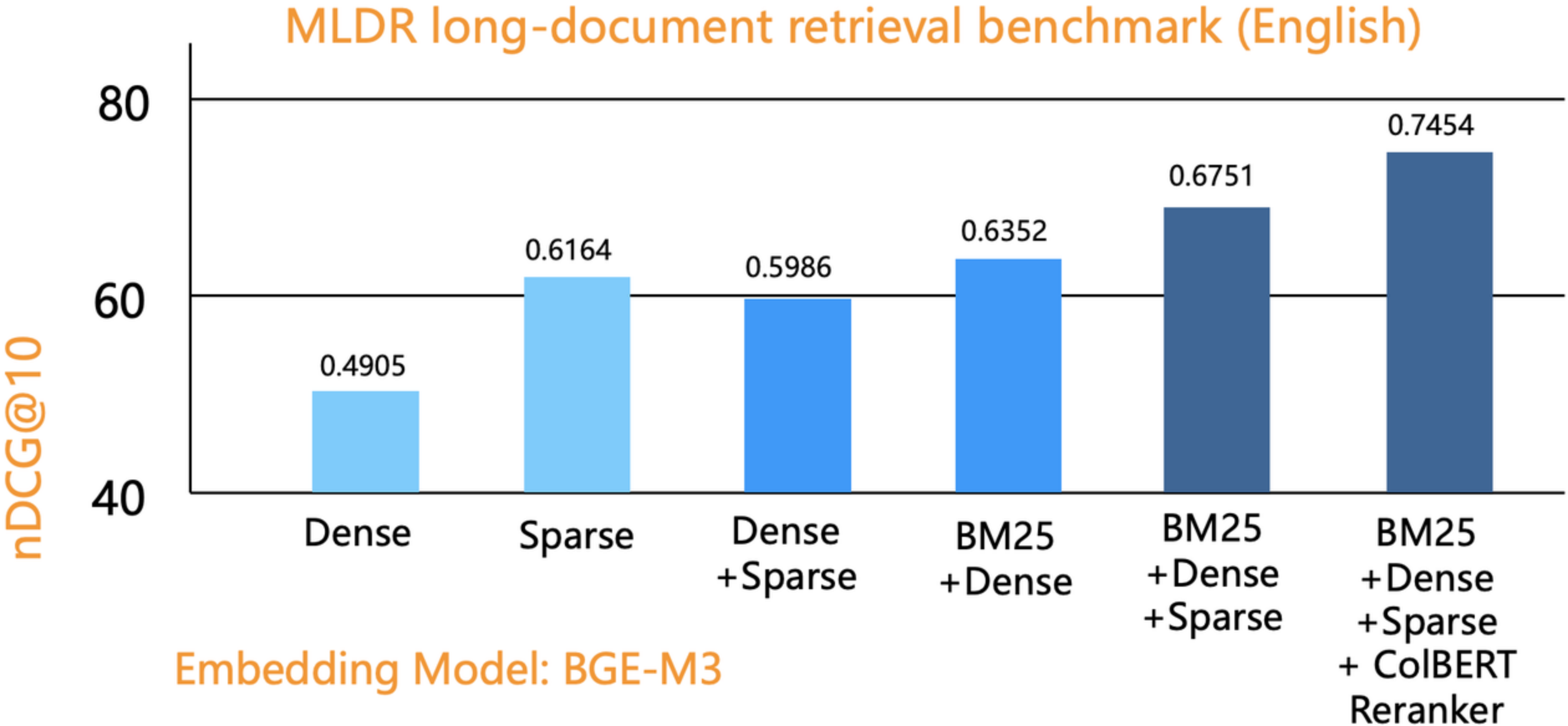
Cross-Encoder



ColBERT (Contextualized Late Interaction)




MaxSim = max(MaxSim1, ...,)
MaxSim 2
MaxSim 1
Embedding
Transformer
Query
Passage1
Passage2
Document


MLDR long-document retrieval benchmark (English)
nDCG@10
80
60
40
0.4905
0.6164
0.5986
0.6352
0.6751
0.7454
Dense
Sparse
Dense +Sparse
BM25 +Dense
BM25 +Dense +Sparse
BM25 +Dense +Sparse + ColBERT Reranker
Embedding Model: BGE-M3

ColBERT

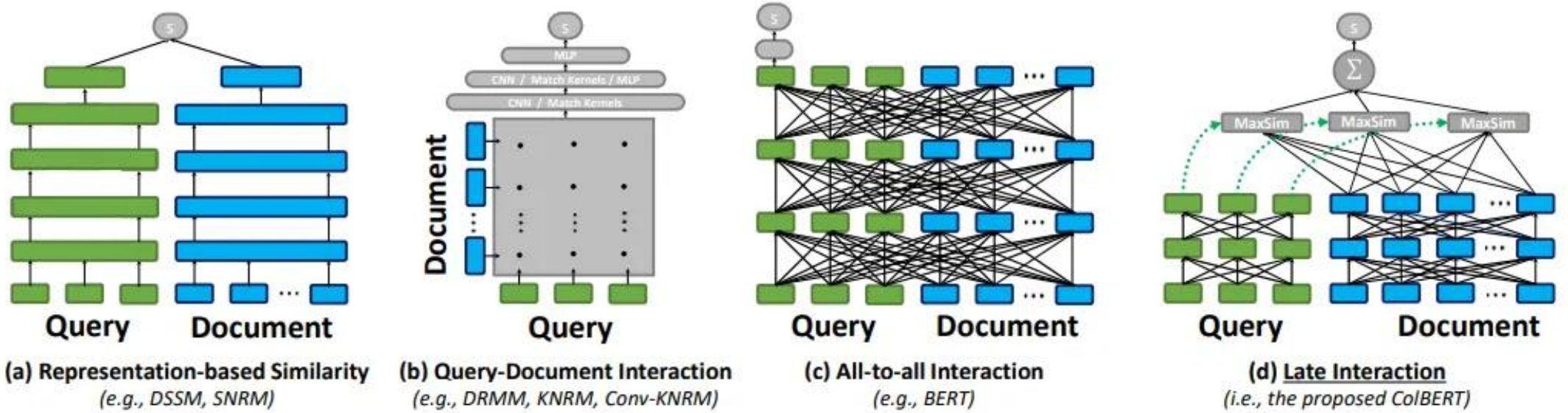
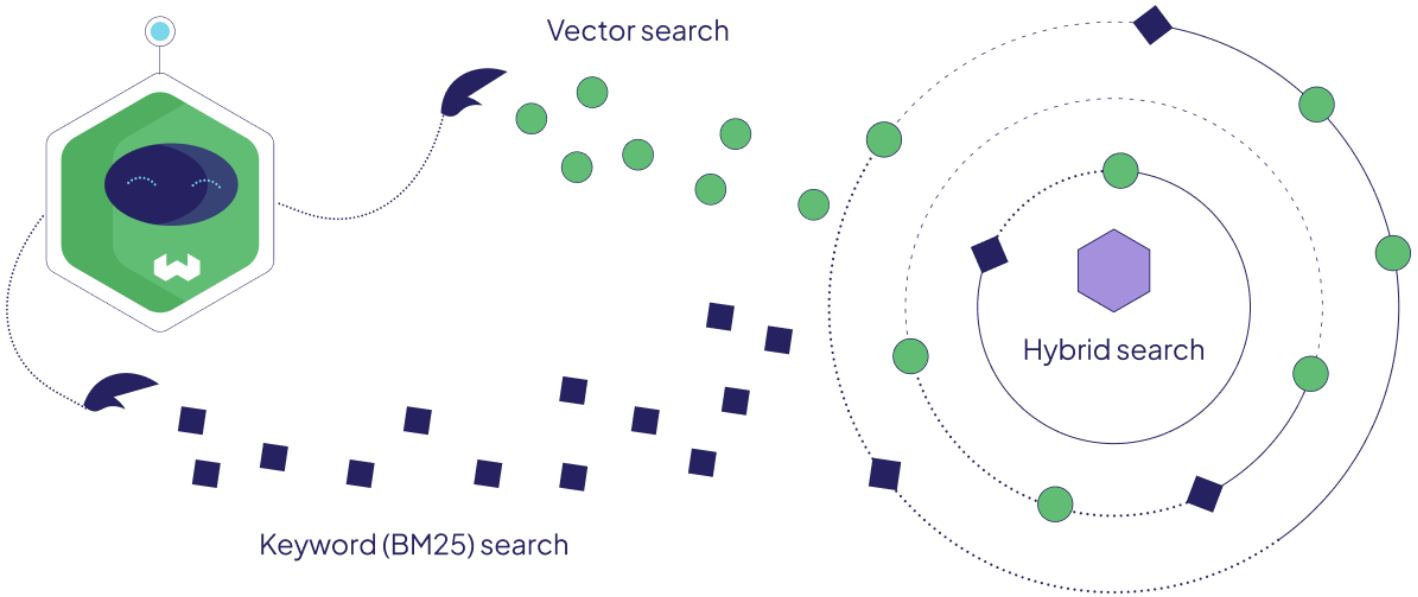


Figure 2: Schematic diagrams illustrating query–document matching paradigms in neural IR. The figure contrasts existing approaches (sub-figures (a), (b), and (c)) with the proposed late interaction paradigm (sub-figure (d)).

Hybrid Search



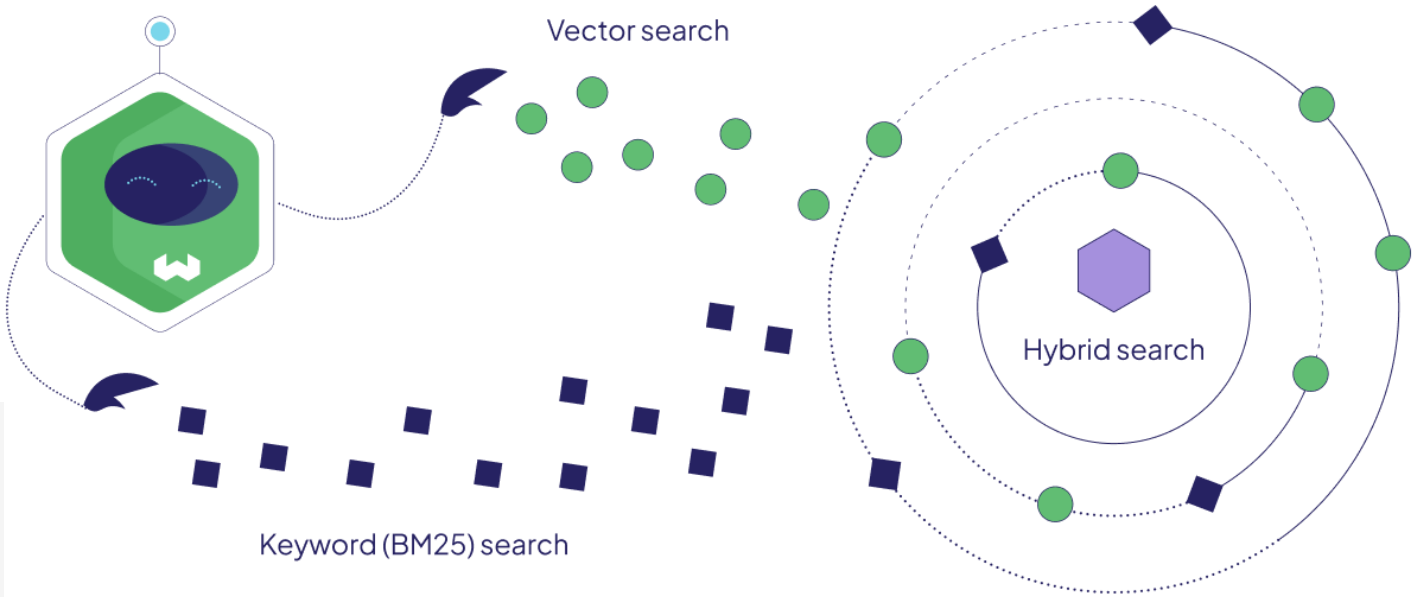
Advantages

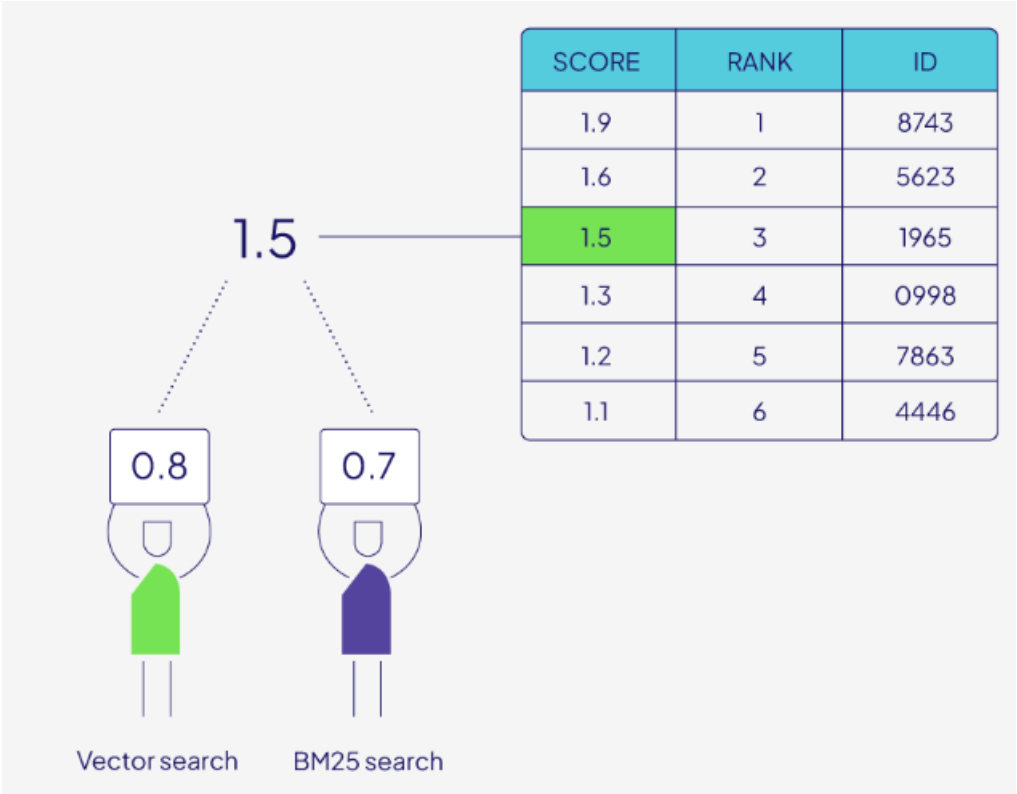
- keywords still needed:
  - ambiguous phrases,
  - domain-specific terms
  - misspellings
- Understanding the semantics
- improves user experience – non perfectly phrased queries

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_i) \frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgd_1})}$$

Source: Wikipedia page on Okapi BM25

Hybrid Search – Simple Fusion





SCORE	RANK	ID
1.9	1	8743
1.6	2	5623
1.5	3	1965
1.3	4	0998
1.2	5	7863
1.1	6	4446

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_i) \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgd_1})}$$

Source: Wikipedia page on Okapi BM25

LLMs for IR - problems

Problems traditionally found in LLMs:

- **Data obsolescence (outdated)** - RAG can retrieve up-to-date information
- **Re-training costs:** don’t need frequent and costly re-training
- **Hallucinations:** grounding responses in factual information retrieved from real documents

LLMs Hallucinate

- Hallucination: a response that is not faithful to the facts of the world.
- In the legal domain LLMs were shown to hallucinate up to 88% of the time!

Dahl et al. (2024)

Hallucination - Factual

Category	Type	Example Dialog	Explanation
Factuality Hallucination	Factual Contradiction	**Instruction:** What are Thomas Edison's main contributions to science and technology? **Response:** Thomas Edison developed the first practical telephone and invented the light bulb.	The response is factually incorrect. In reality, **Edison improved the light bulb, building on earlier designs by others**, and **Alexander Graham Bell invented the telephone**.
	Factual Fabrication	**Instruction:** What are the major environmental impacts of the construction of the Eiffel Tower? **Response:** The construction of the Eiffel Tower in 1889 led to the extinction of the Parisian tiger, a species that played a crucial role in the region's ecosystem. Additionally, it is widely recognized as the event that sparked the global green architecture movement.	The response erroneously states that the construction of the Eiffel Tower led to the extinction of the 'Parisian tiger' and sparked the global green architecture movement. In fact, **there is no evidence of a 'Parisian tiger' ever existing, making this a fabricated claim.** Moreover, **attributing the origins of green architecture to the Eiffel Tower is an exaggeration**, as this movement has diverse roots and cannot be traced back to a single event.

Hallucination - Faithfulness

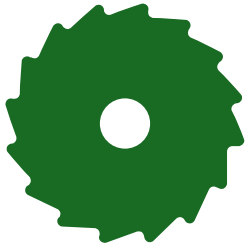
Category	Type	Example Dialog	Explanation
Faithfulness Hallucination	Instruction Inconsistency	**Instruction:** Translate the English question into Spanish: "What is the capital of France?" **Response:** The capital of France is Paris.	The user explicitly provided an instruction to translate the English question into Spanish. **Instead of performing the translation as instructed, the LLM chose to answer the question.**
	Context Inconsistency	**Retrieved context:** The Nile is a major north-flowing river in northeastern Africa. It's traditionally considered the longest river in the world. Originating in the Great Lakes region of central Africa, it flows through several countries before emptying into the Mediterranean Sea. **Instruction:** Summarize the retrieval document above. **Response:** The Nile originates from the mountain ranges of central Africa and flows into the Mediterranean.	The summary incorrectly states that the Nile originates from the "mountain ranges of central Africa", while the **paragraph mentions that the Nile originates in the Great Lakes region of central Africa.**
	Logical Inconsistency	**Instruction:** Solve the equation 2x + 3 = 11 step by step. **Response:** Step 1: Subtract 3 from both sides to get: 2x = 8. Step 2: Divide both sides by 2 to get: x = 3. So the result of the equation is x=3.	The first step correctly isolates the term with x to get 2x = 8. However, **the next step inaccurately divides 8 by 2 to yield a result of x = 3**, which is inconsistent with the earlier reasoning.

Hallucination - Misinformation and Biases

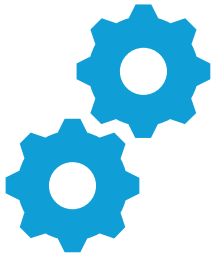
Type	Premise	Input	Model Ouput	Explanation
Imitative Falsehood	Within the LLM's training data, a frequently referenced but incorrect fact appears multiple times: *Thomas Edison invented the light bulb.*	Who is credited with the invention of the light bulb?	Thomas Edison is credited with the invention of the light bulb.	**While Edison improved the light bulb's design, he was not its sole inventor.** The model's answer reflects a misconception prevalent in its training data, exemplifying an imitative falsehood.
Societal Biases	In the model's training data, certain names may frequently be associated with specific nationalities.	Provide a summary of the following news: Dr. Kim recently published a breakthrough paper on quantum computing. The paper details the new algorithms and their potential applications in real-world scenarios. It has been hailed as revolutionary by experts.	Dr. Kim from South Korea recently published a paper on quantum computing, detailing new algorithms and their applications.	**The model erroneously added "South Korea" based on the name "Kim"** revealing its bias in linking certain names to specific nationalities, which can cause hallucinations in the summary.

Possible Solutions

Fine tuning



RAG

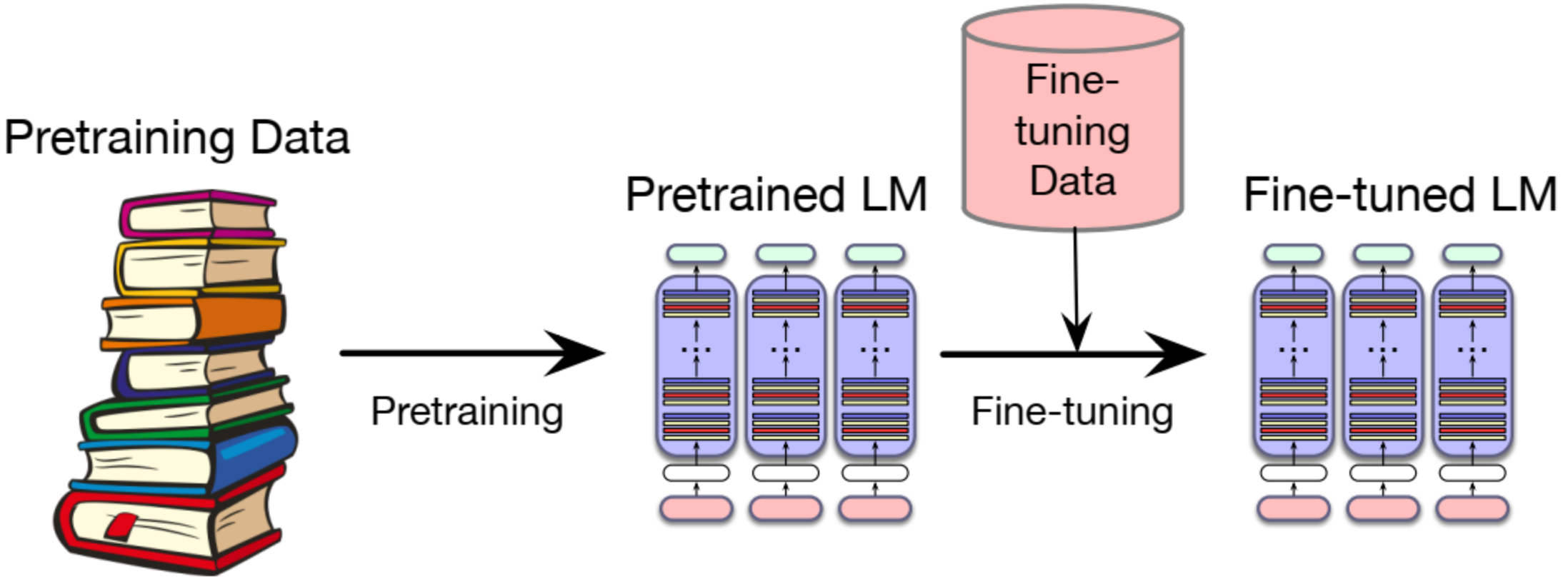


Prompt Engineering

What is LLM Fine-Tuning

- **fine-tuning** an existing (and previously trained) LLM consists of **adjusting its model parameters** using **additional, specialized training data** to enhance its performance in a specific use case or application domain.
- Fine-tuning is an important part of LLM development, maintenance, and reuse, for two reasons:
- It allows the model to adapt to a more domain-specific, often smaller, dataset, improving its accuracy and relevance in specialized areas such as legal, medical, or technical fields. See the example in the image below.
- It ensures the LLM stays up-to-date on evolving knowledge and language patterns, avoiding issues like outdated information, hallucinations, or misalignment with current facts and best practices

Fine tuning process



Finetuning as "continued pretraining" on new data

- Further train all the parameters of model on new data
- using the same method (word prediction) and loss function
- (cross-entropy loss) as for pretraining.
- as if the new data were at the tail end of the pretraining data
- Hence sometimes called **continued pretraining**

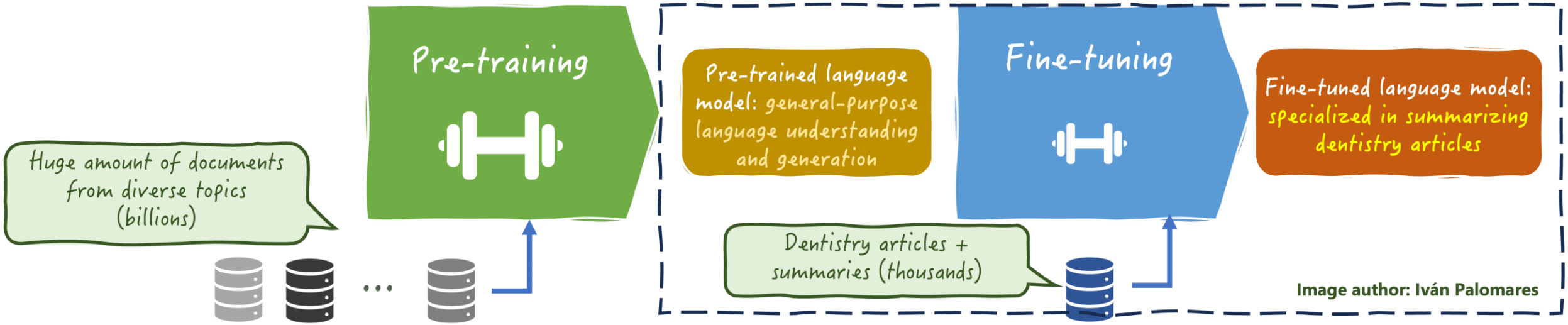
Fine tuning for specialized domain

Fine-tuning involves training the LLM further using your proprietary data. It essentially “teaches” the model to understand your specific domain better.

How It Works

1. Take an existing LLM (like GPT-4).
2. Feed it your domain-specific data (e.g., medical records, customer support logs).
3. Update the model’s parameters so it “learns” to generate more accurate, relevant responses for your use case.

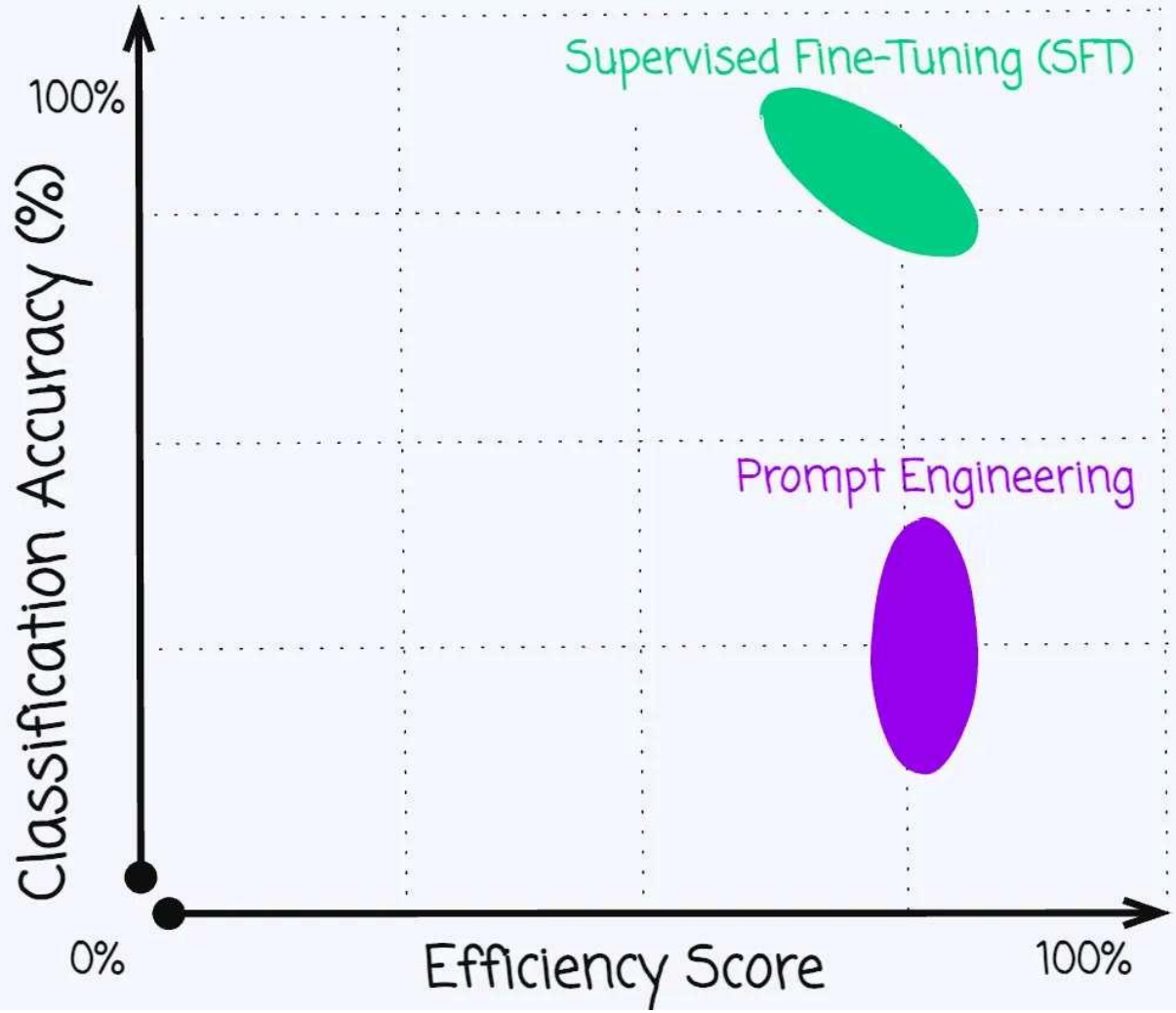
Fine Tuning for specialized task



Fine Tuning

- **Domain-Adaptive Pre-training (DAP)**
- Despite its name, DAP can be used as an intermediate strategy between general model pretraining and task-specific fine-tuning of base LLMs inside RAG. It consists in utilizing a domain-specific corpus to have the model gain a better understanding of a certain domain, including jargon, writing styles, etc. Unlike conventional fine-tuning, it may still use a relatively large dataset, and it often is done before integrating the LLM with the rest of the RAG system, after which more focused and task-specific fine-tuning on smaller datasets would take place instead.
- **Retrieval Augmented Fine-Tuning**
- This is an interested and more RAG-specific fine-tuning strategy, whereby the LLM is specifically retrained on examples that incorporate both the retrieved context — augmented LLM input — and the desired response. This makes the LLM more skilled at leveraging and optimally utilizing retrieved knowledge, generating responses that will better integrate that knowledge. In other words, through this strategy, the LLM gets more skilled in properly using the RAG architecture it sits on.

Small Language Models for Text Classification



Source: https://arxiv.org/abs/2505.16078

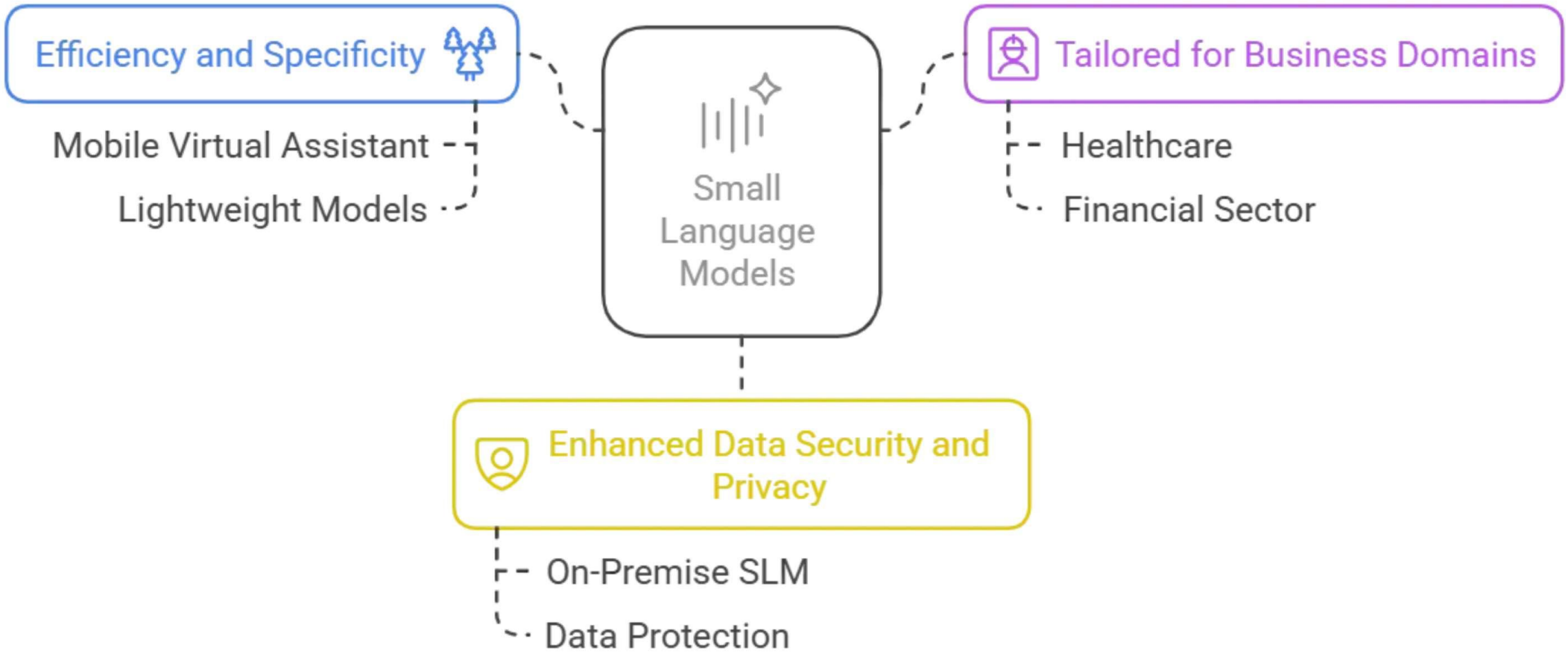
Best Choice ⭐
Supervised Fine-Tuning on 1B models achieves 89% accuracy with optimal efficiency.

Most Efficient 🛢️
Prompt Engineering (few-shot) is fast and lightweight but limited performance.

💡 Key Insight: 1B models with supervised fine-tuning for near-perfect accuracy while using 5-10x less memory than larger alternatives.

COBUS GREYLING & AI

SLMs (Small Language Models)

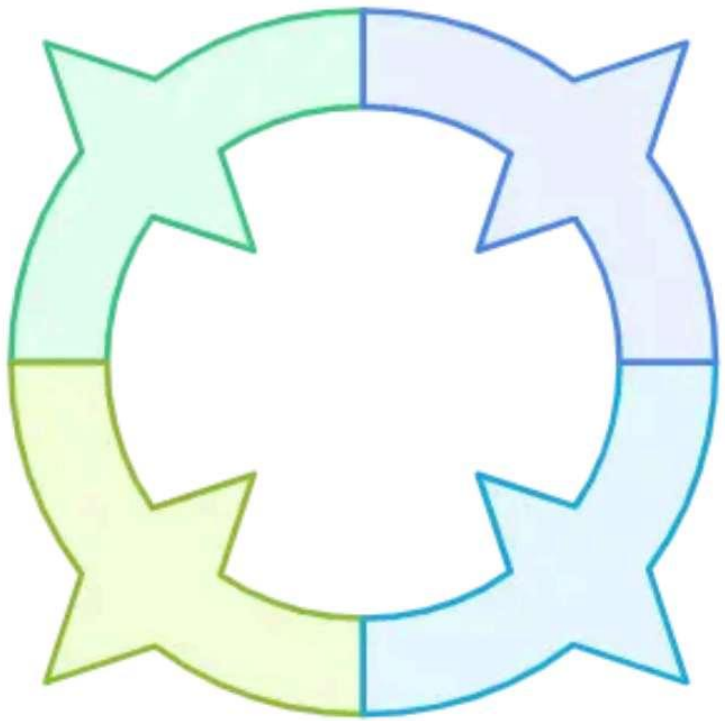


BioBERT in Clinical Decision-Making

BioBERT enhances medical research with precise language processing.

Healthcare

Domain-Specific Models



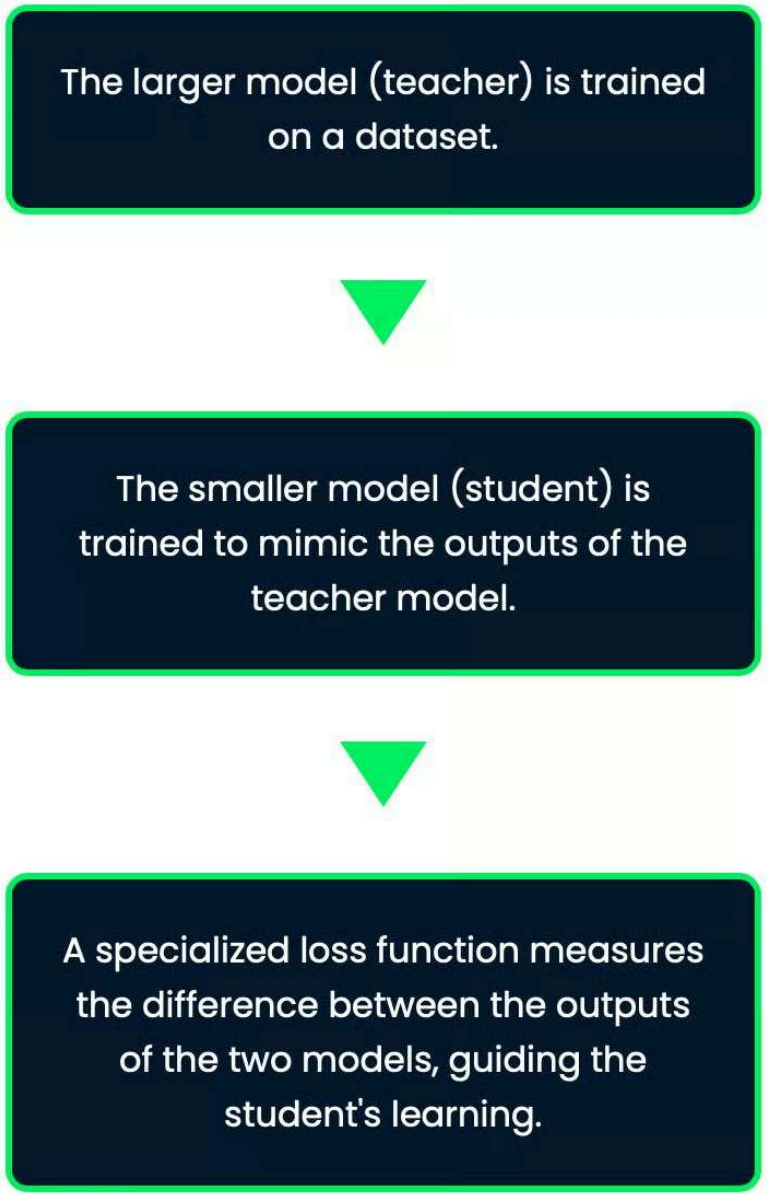
Micro Models

Customer Support

E-commerce Chatbot

E-commerce chatbot improves customer satisfaction with quick responses.

SLM Creation – via distillation



SLM Creation – via distillation

Response-Based

The student model learns to replicate the final output layer of the teacher model, often using "soft targets" for more nuanced information.

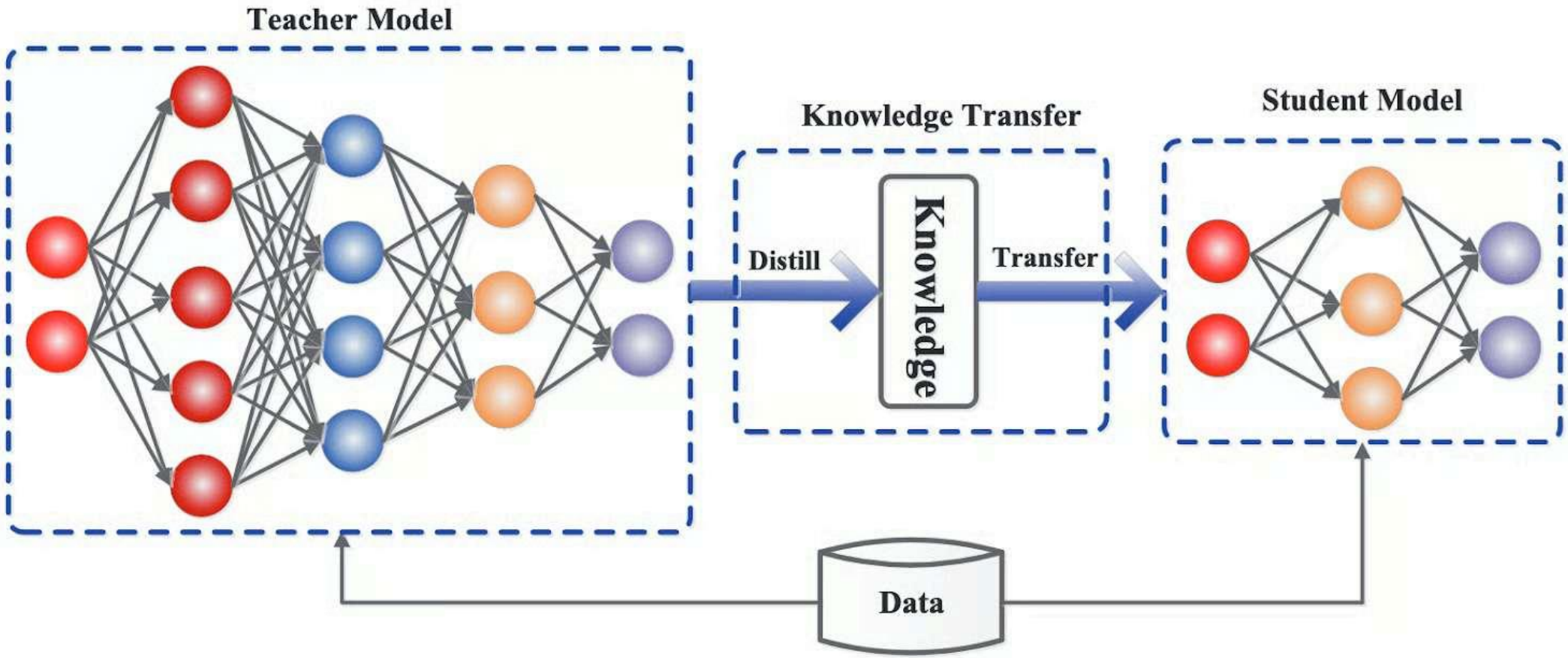
Feature-Based

Focuses on replicating the intermediate layers of the teacher model, helping the student extract similar patterns from data.

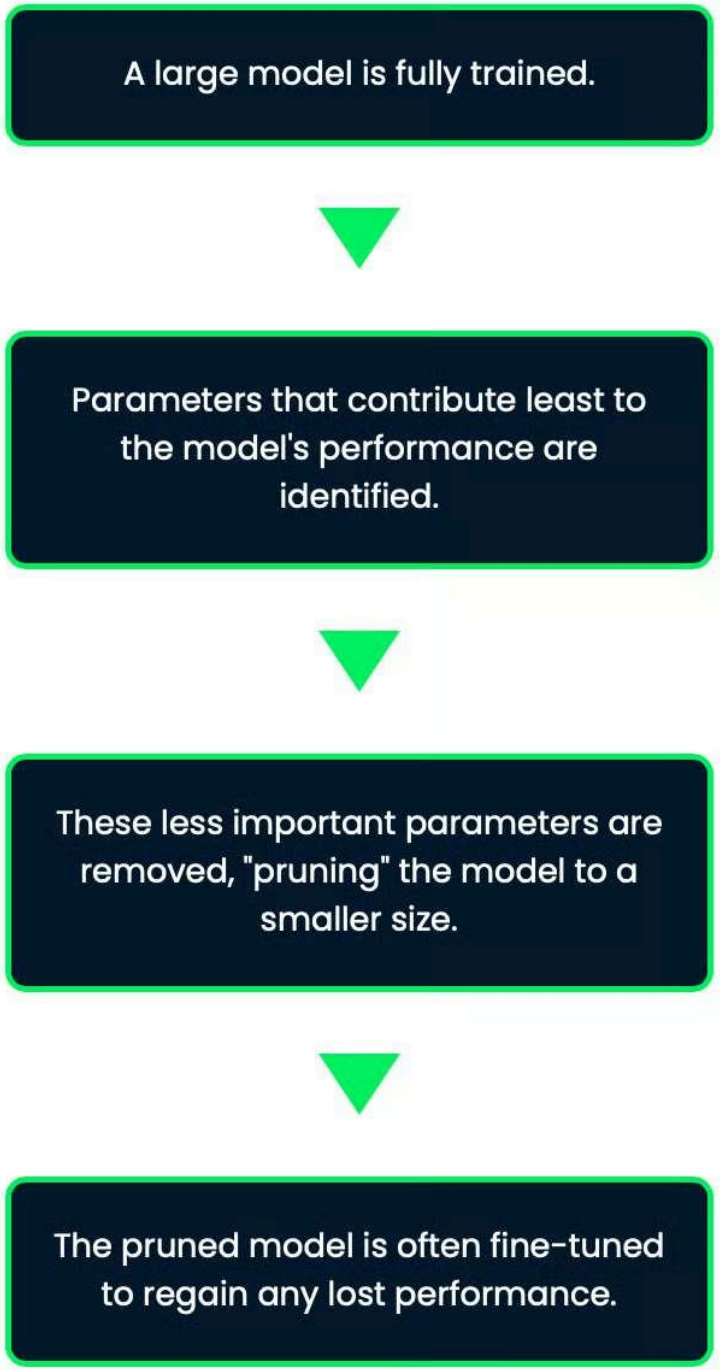
Relation-Based

Trains the student to understand relationships between different parts of the teacher model, emulating complex reasoning processes.

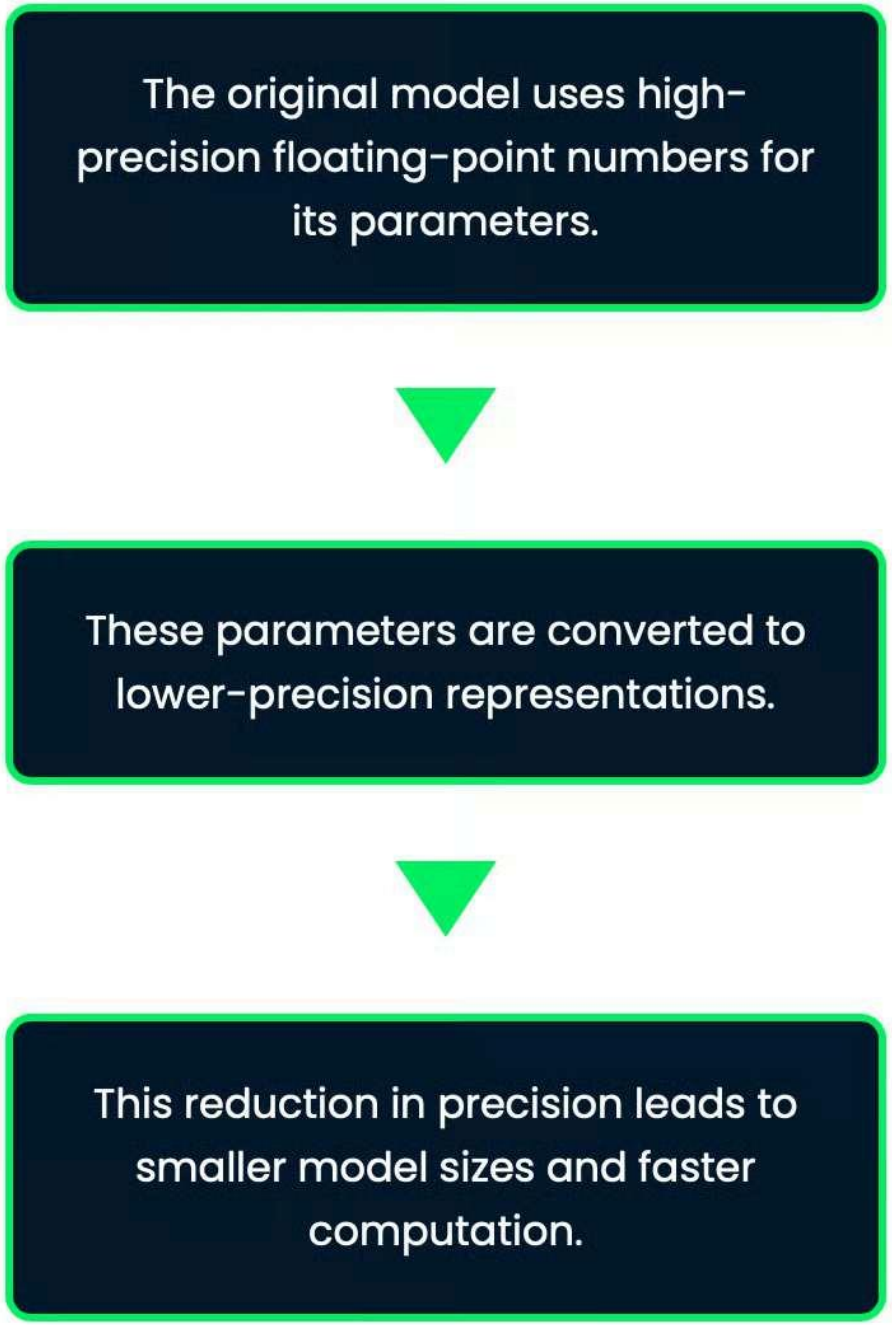
SLM Creation – via distillation



SLM Creation – via pruning

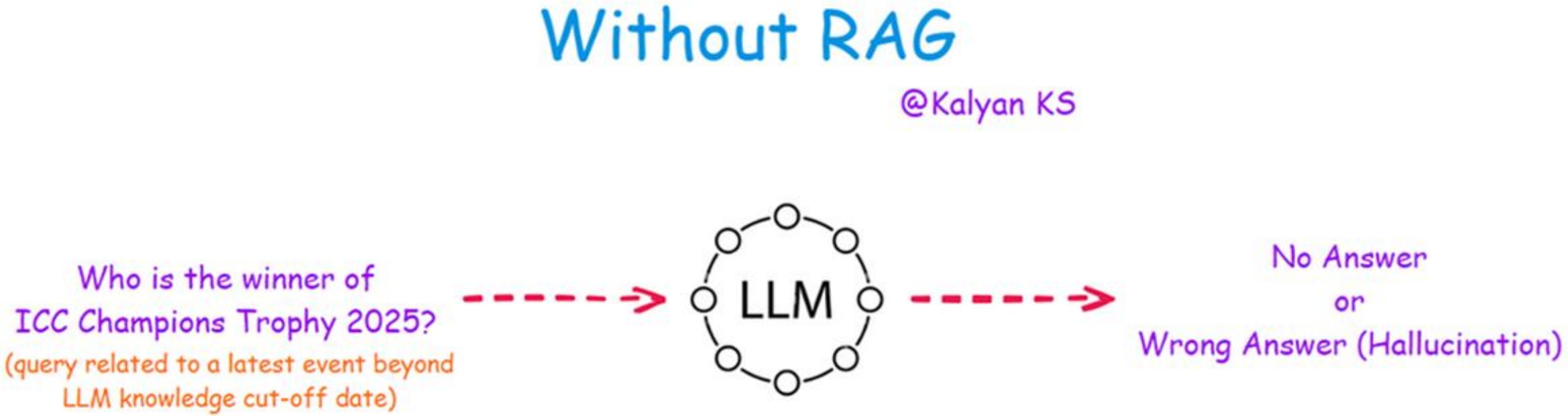


SLM Creation – via quantization



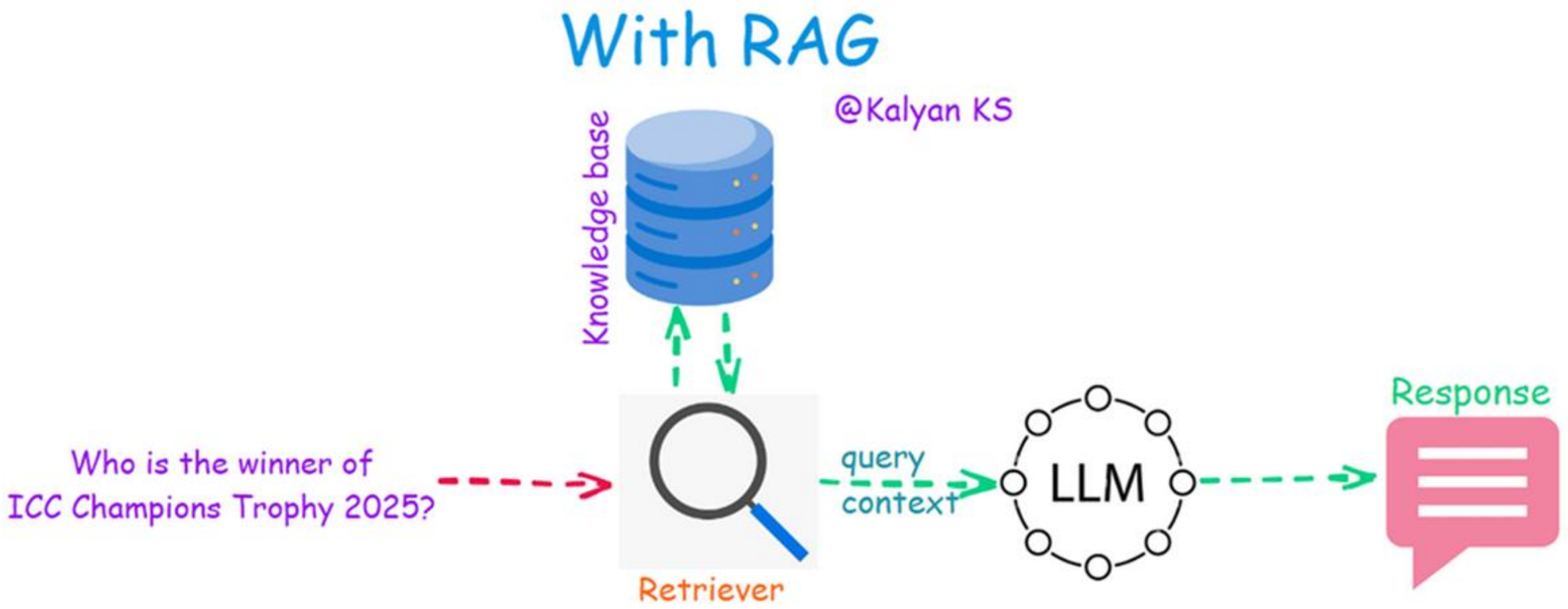
ASPECT	LARGE LANGUAGE MODELS	SPECIALIZED LANGUAGE MODELS
Model Size	Very large, often exceeding billions of parameters	Smaller, typically ranging from millions to hundreds of millions of parameters
Training	Trained on vast datasets using deep learning techniques	Can be trained on smaller, more specific datasets
Data Specificity	General-purpose, designed to handle a wide range of topics and contexts	Highly specialized, optimized for specific domains or tasks
Resource Requirements	High computational power, memory, and infrastructure needed for training and use	More efficient, requiring less computational power and memory
Performance	Excels in generating complex, nuanced text across diverse topics	Performs well on specific tasks but may struggle with broader generalization
Cost of Implementation	Expensive to build and maintain due to extensive resources and infrastructure	More cost-effective, requiring fewer resources for training and deployment
Risk of Hallucinations	Higher chance of generating incorrect or irrelevant information (hallucinations)	Lower risk of hallucinations due to domain-specific focus
Enterprise Usability	May face issues accessing data behind firewalls, especially in enterprise contexts	More easily integrated into enterprise environments, especially for specific tasks
Examples	OpenAI's GPT-4o, Google's Bard	Microsoft's phi-3-mini, OpenAI's GPT-4o mini

LLMs for IR - problems



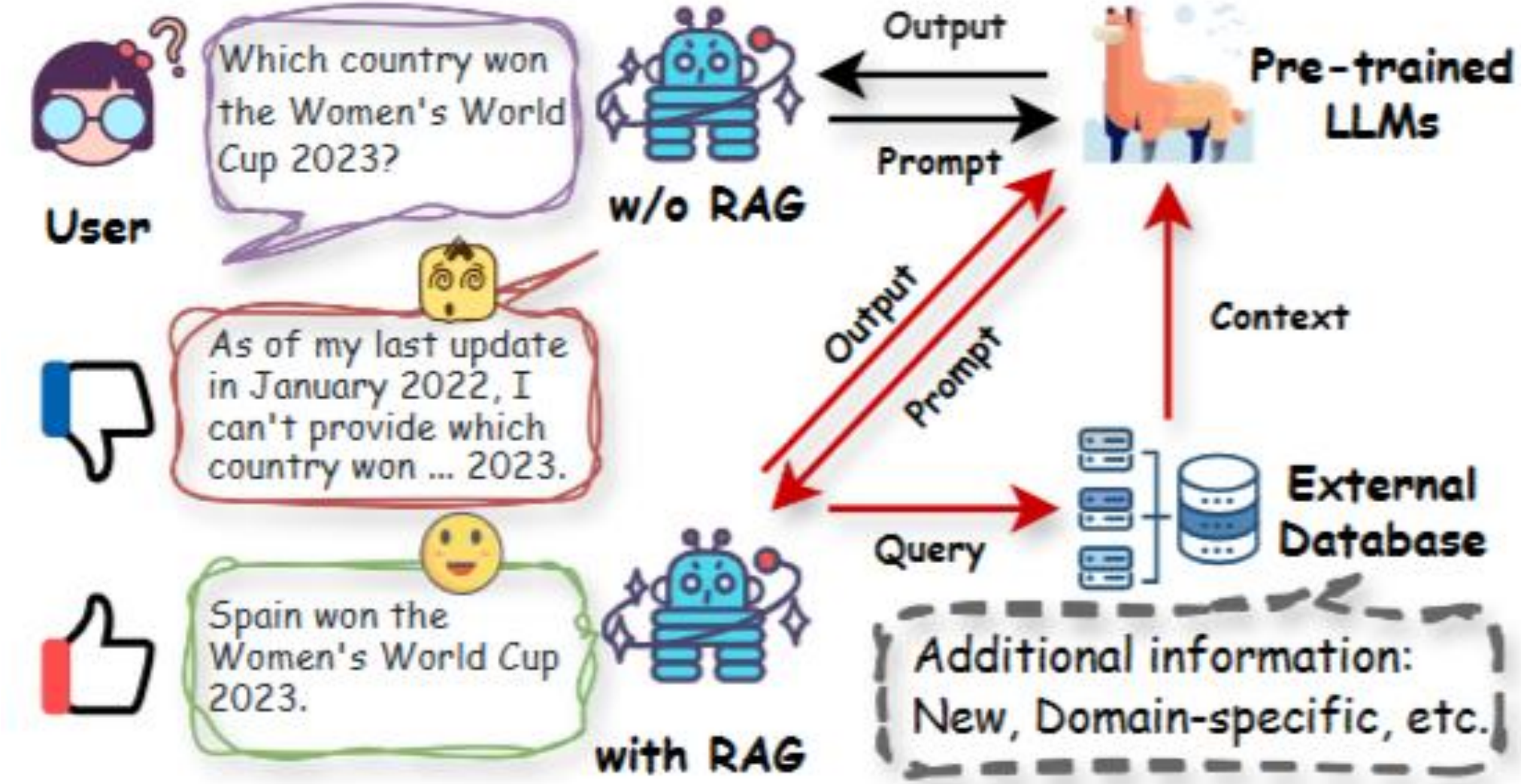
Credit: github.com/KalyanKS-NLP

RAG - Motivation



Credit: github.com/KalyanKS-NLP

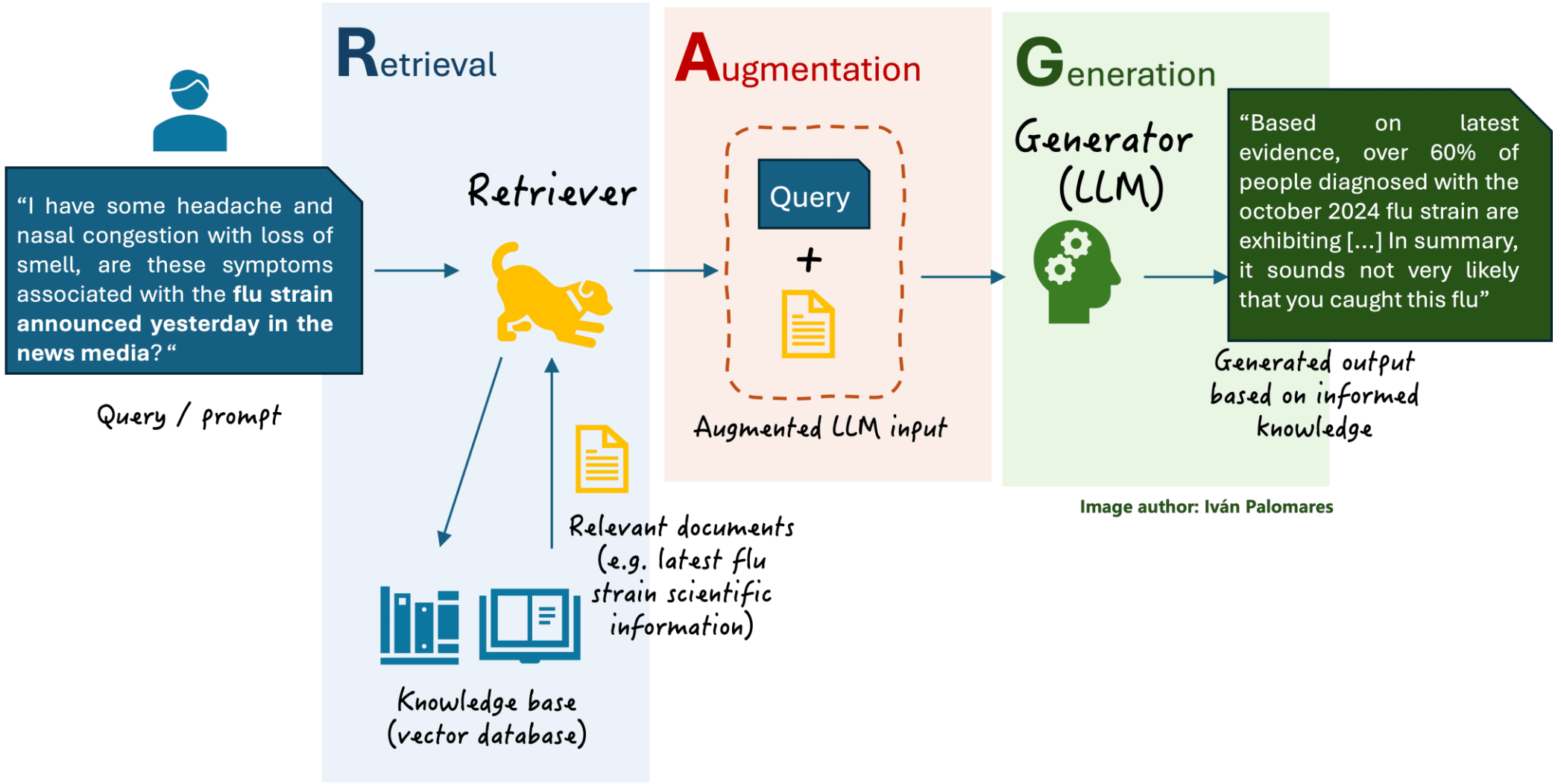
Adding a RAG


Which country won the Women's World Cup 2023?
User
w/o RAG
Output
Prompt
Pre-trained LLMs
As of my last update in January 2022, I can't provide which country won ... 2023.
Output
Prompt
Context
Query
External Database
Spain won the Women's World Cup 2023.
with RAG
Additional information:
New, Domain-specific, etc.

RAG – typical key data-related components

- **Retrieval**: a component called **retriever** accesses the vector database to find and retrieve relevant documents to the user query.
- **Augmentation**: the original user query is augmented by incorporating contextual knowledge from the retrieved documents.
- **Generation**: the LLM -also commonly referred to as **generator** from an RAG viewpoint- receives the user query augmented with relevant contextual information and generates a more precise and truthful text response.

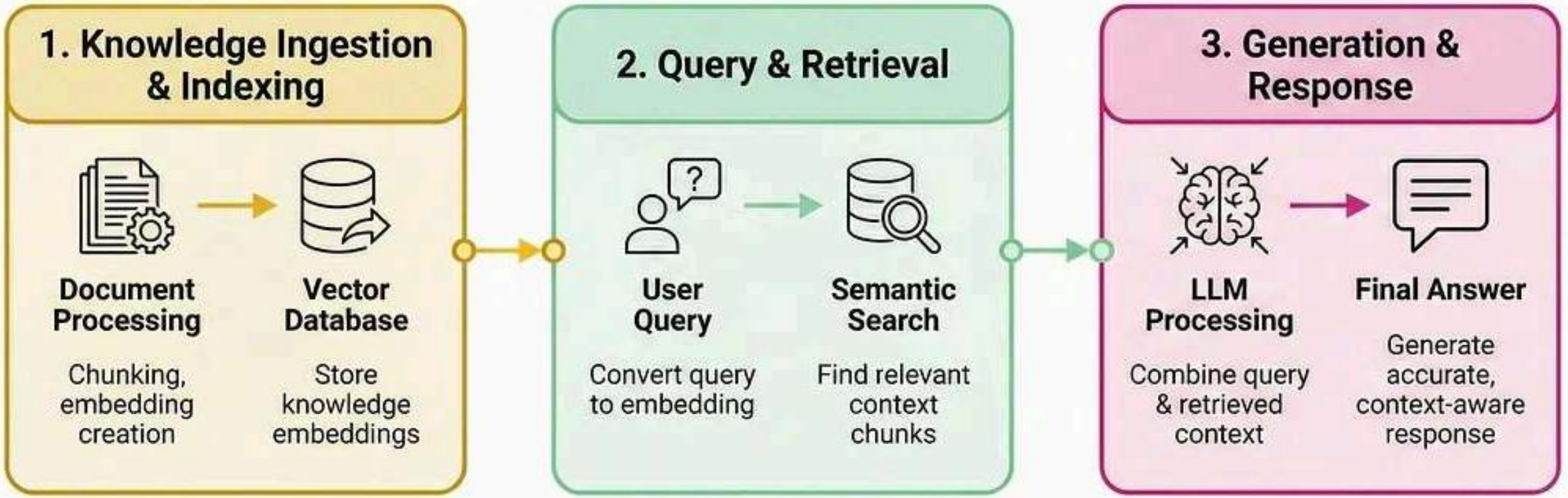
Typical RAG system



Prompting

- Prompting in LLMs refers to the technique of providing a specific instruction or context to guide the model's generation of text. The prompt serves as a conditioning signal and influences the language generation process of the model. Existing prompting strategies can be roughly categorized into three groups: zero-shot prompting, few-shot prompting, and chain-of-thought (CoT) prompting

How RAG (Retrieval-Augmented Generation) Works



Where to use Prompt Engineering?

1. Foundation & Analysis

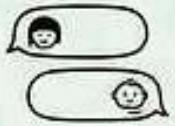
NLP Tasks

Language understanding, summarization

Data Analysis

Pattern extraction, insights

2. Interaction & Retrieval



Chatbots & Assistants

Virtual agents, support handling

FAQ Systems

Knowledge base querying, Q&A

3. Generation & Personalization

Content Generation

Writing, creative assets, code

Recommendation Systems

Tailored suggestions, ranking

Prompt Engineering vs RAG

Prompt Engineering (Optimization via Instruction)

Crafting Instructions

Optimizing prompts for desired output, creative writing, formatting

Reasoning & Logic Tasks

Structuring queries for complex problem-solving, logic puzzles, code generation

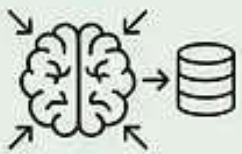
Focus: Maximizing model's internal knowledge & capability through precise communication.

Optimization Path in 2026

RAG (Retrieval-Augmented Generation) (Optimization via External Knowledge)

Accessing External Data

Retrieving up-to-date information, documents, proprietary data



Reducing Hallucinations

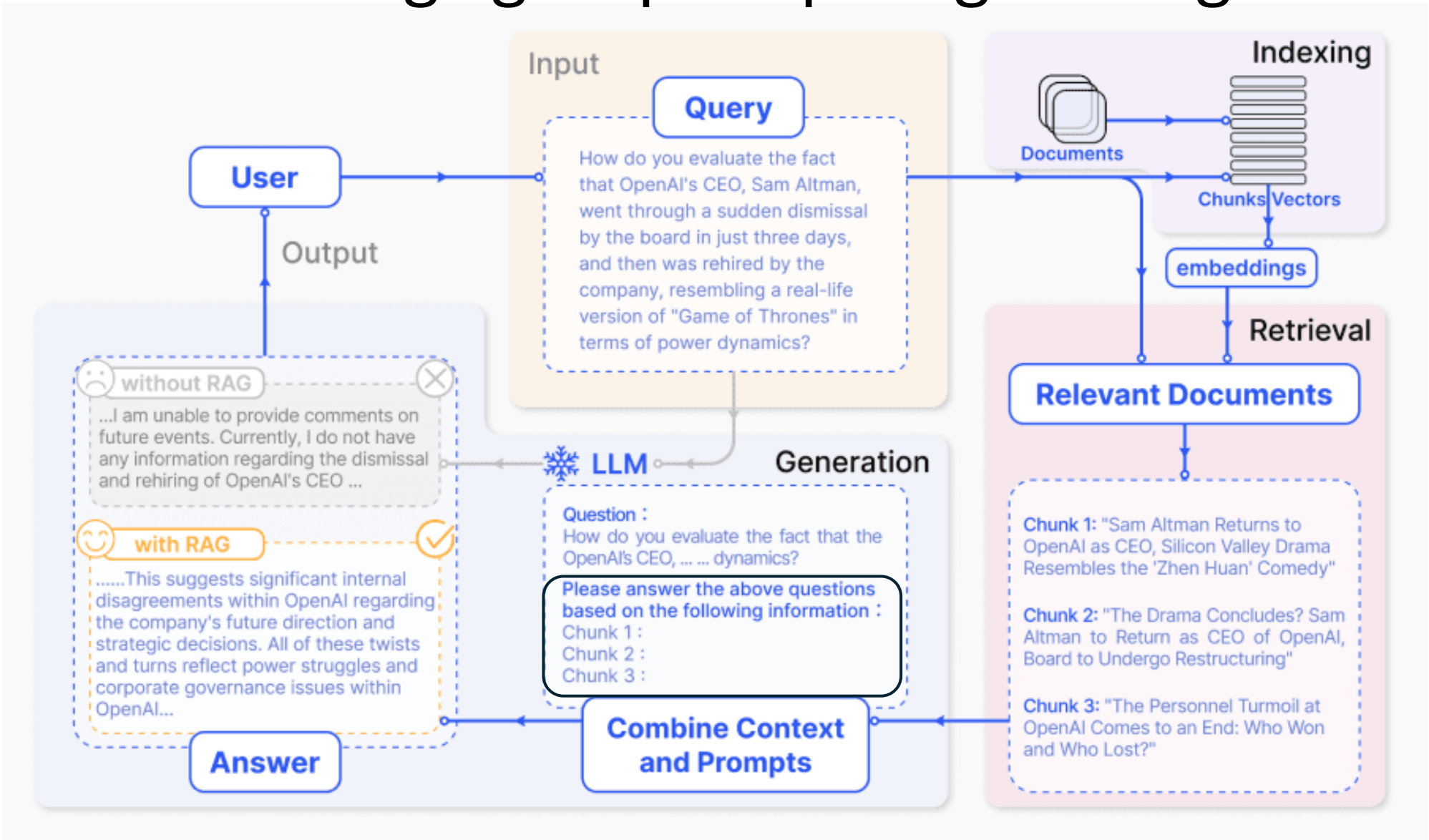
Providing factual context, citations, improving accuracy and reliability

Focus: Augmenting model's knowledge with retrieved, verifiable information.

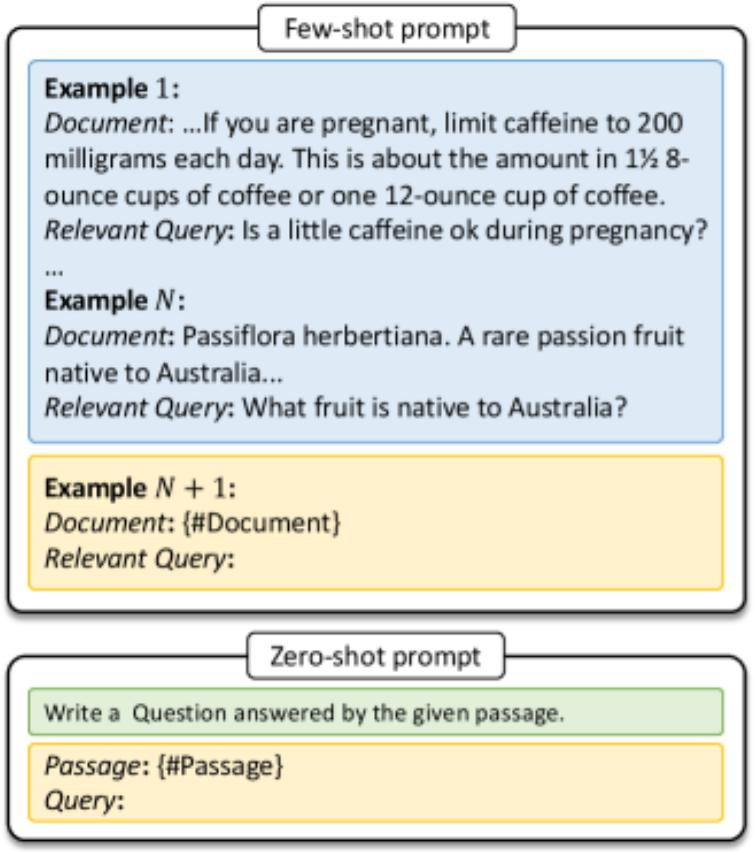
Choose Your Path Based on Data Needs, Accuracy Requirements, and Task Complexity.

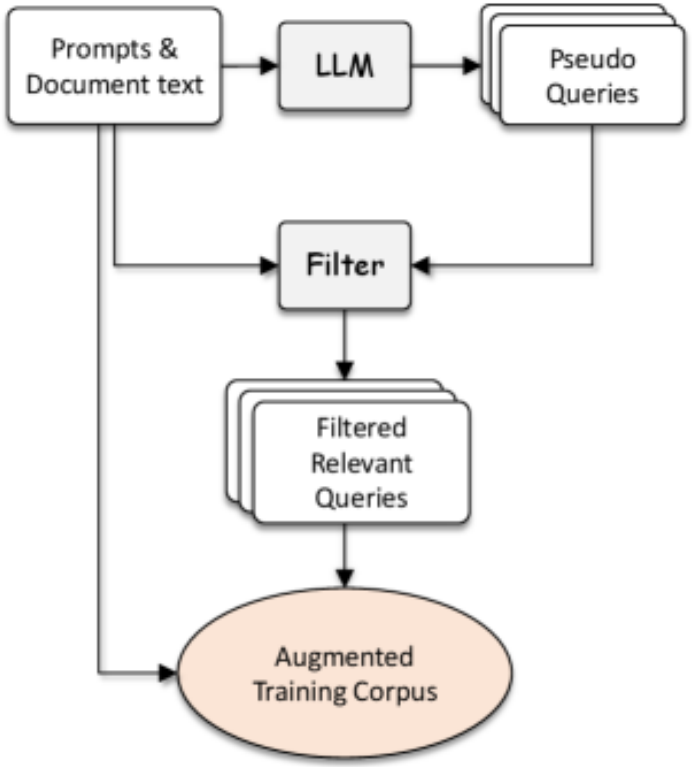
GenAI.works

AI Chat Bot (RAG system with large close LLM) – Leveraging for prompt engineering

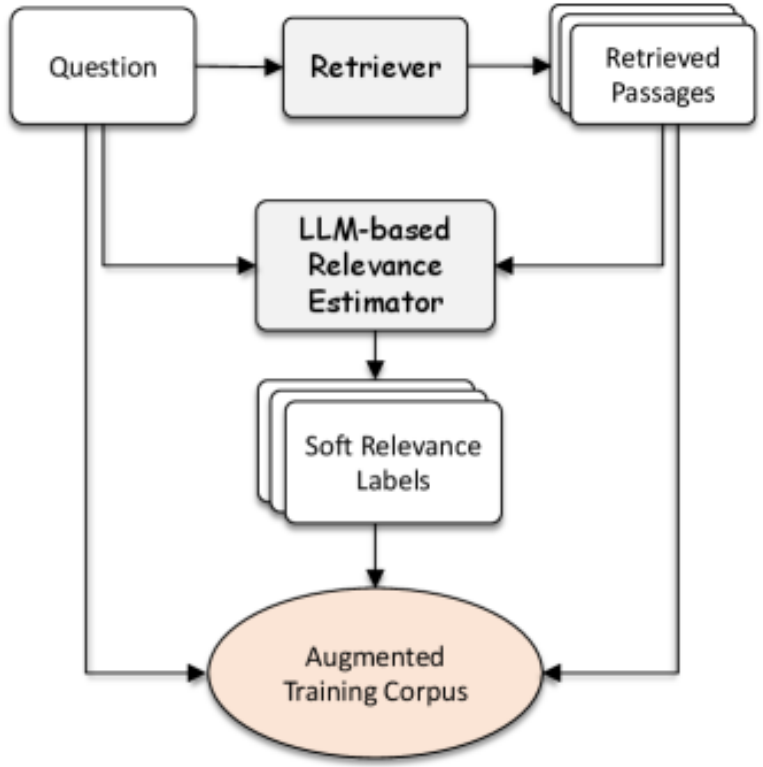


Leveraging LLMs to Generate Search Data on an AI Chat Bot





Framework of pseudo query generation

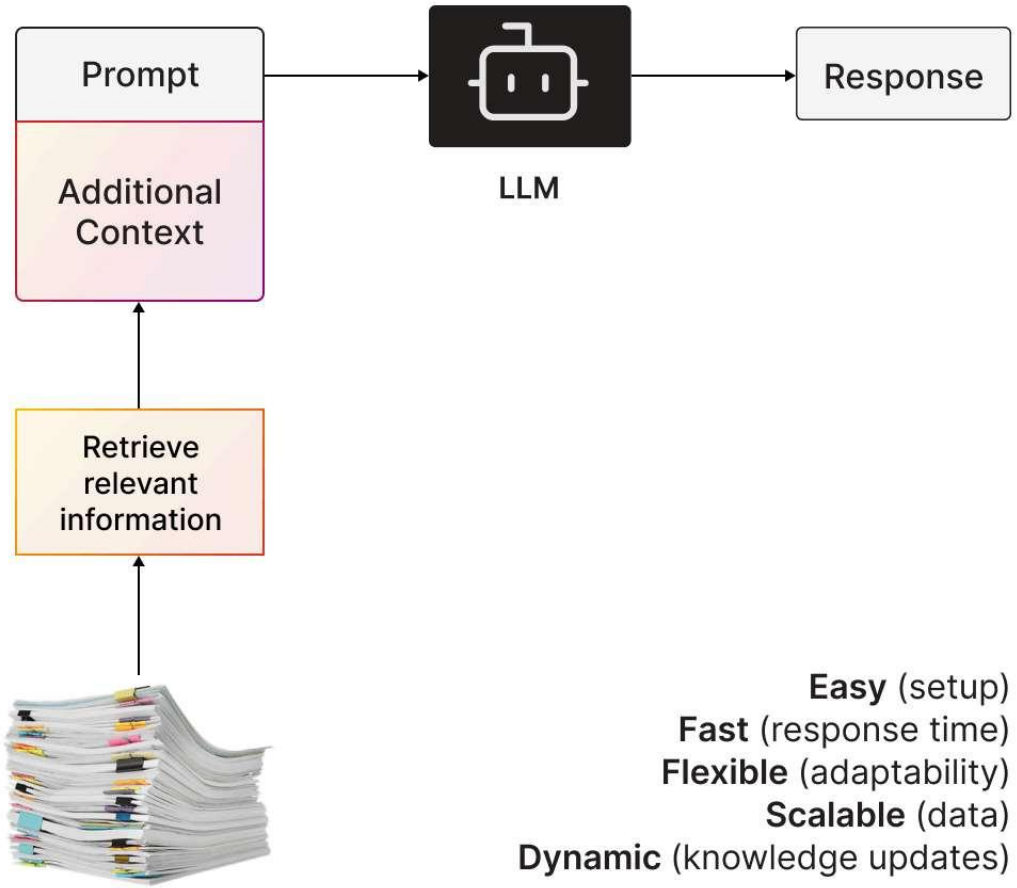


Framework of relevance label generation

Fine-Tuning vs. RAG

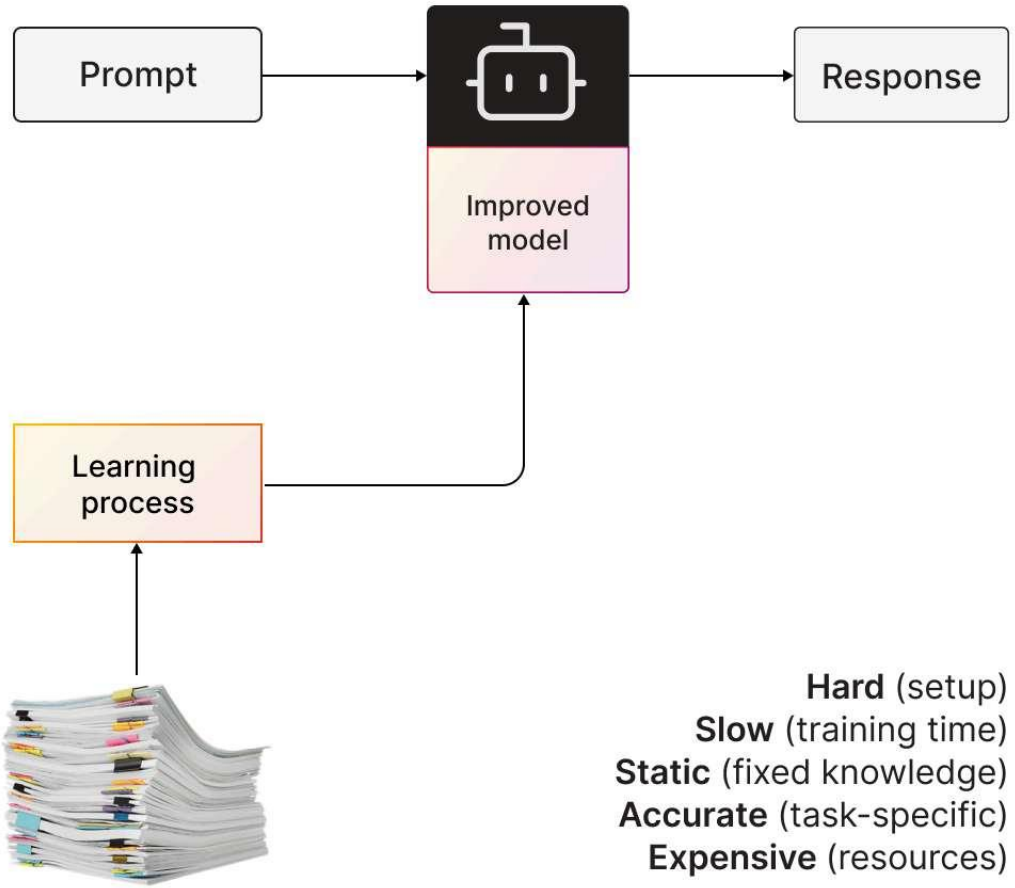
Feature	Fine-Tuning	RAG
Cost	High: Requires GPUs and training time.	Low: Minimal hardware requirements.
Setup Time	Long: Training takes weeks or months.	Short: RAG can be implemented quickly.
Maintenance	High: Frequent retraining required.	Low: Automatically updates with new data.
Flexibility	Low: Fixed to the trained model.	High: Can adapt to any LLM or dataset.
Use Cases	Repeated, predictable queries.	Dynamic, real-time information needs.

Retrieval-augmented generation (RAG)

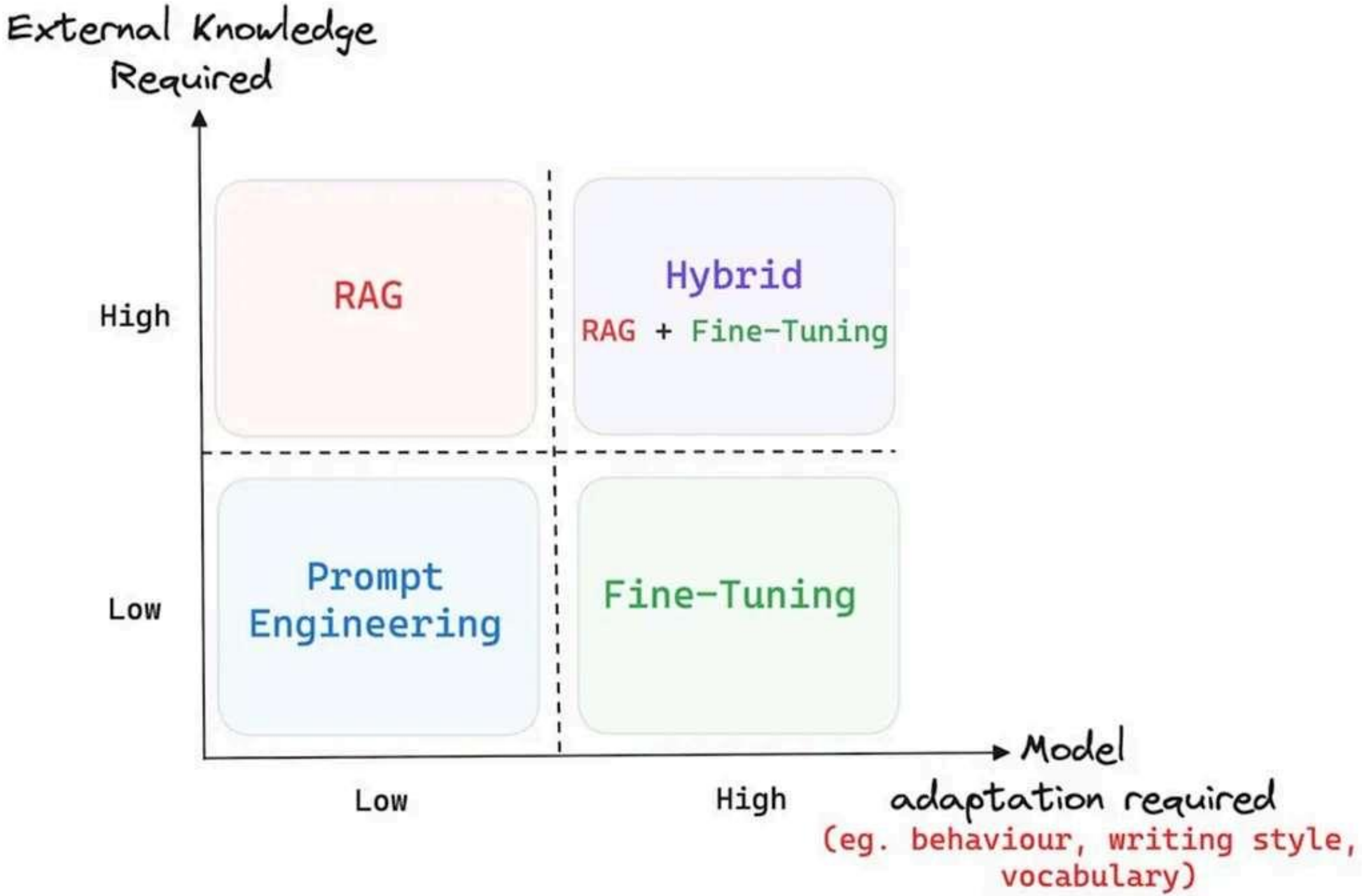


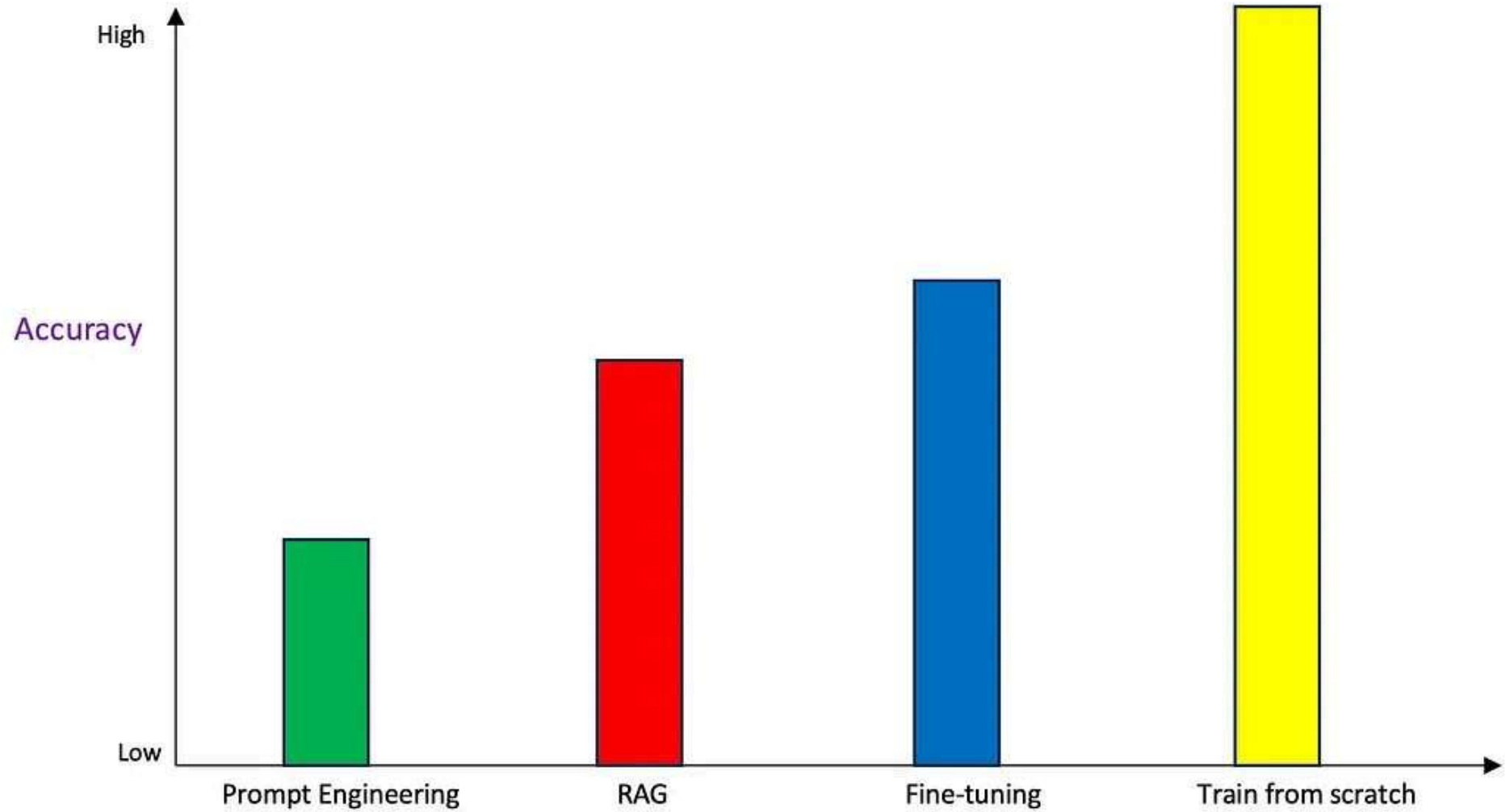
Easy (setup)
Fast (response time)
Flexible (adaptability)
Scalable (data)
Dynamic (knowledge updates)

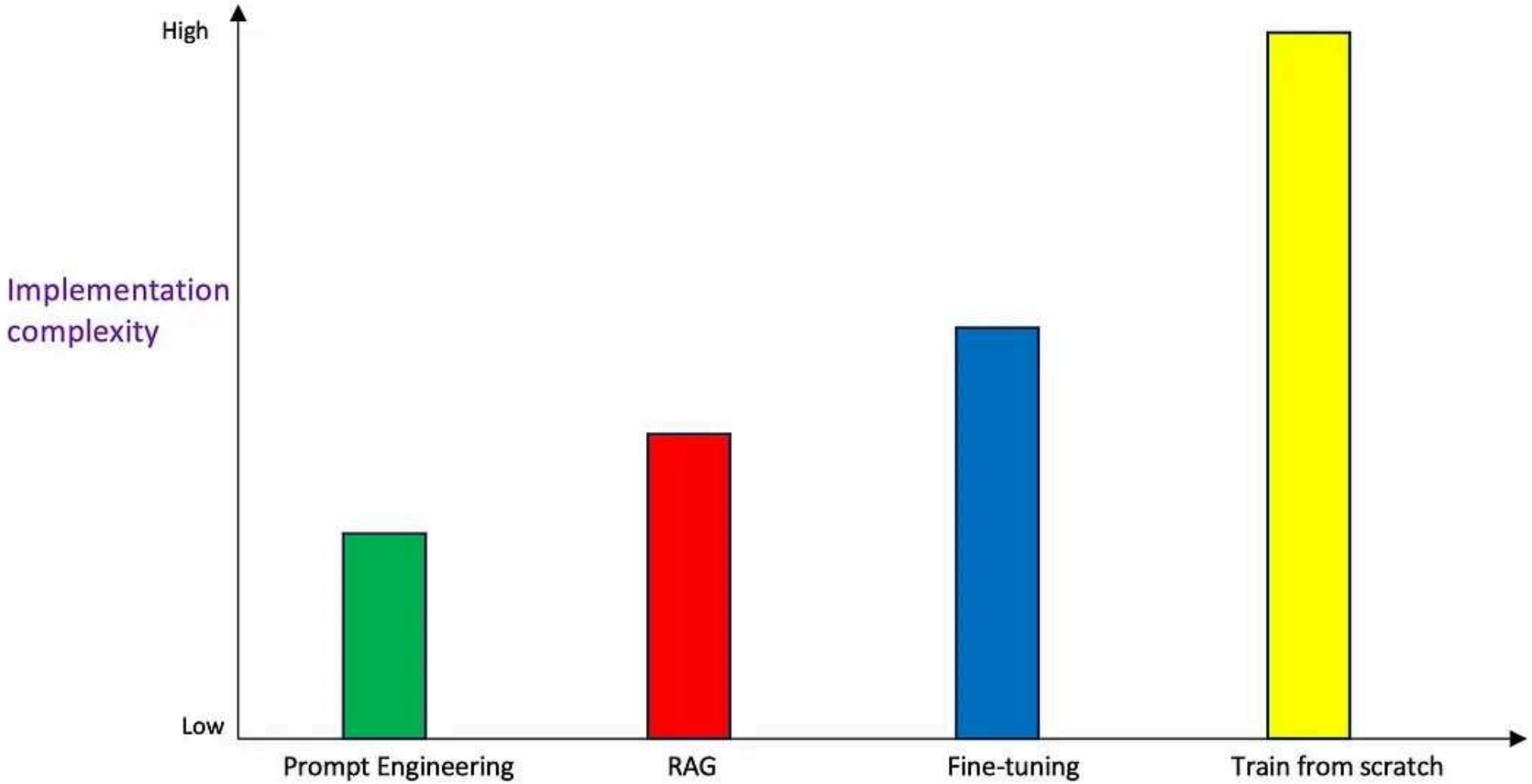
Fine-tuning the Model

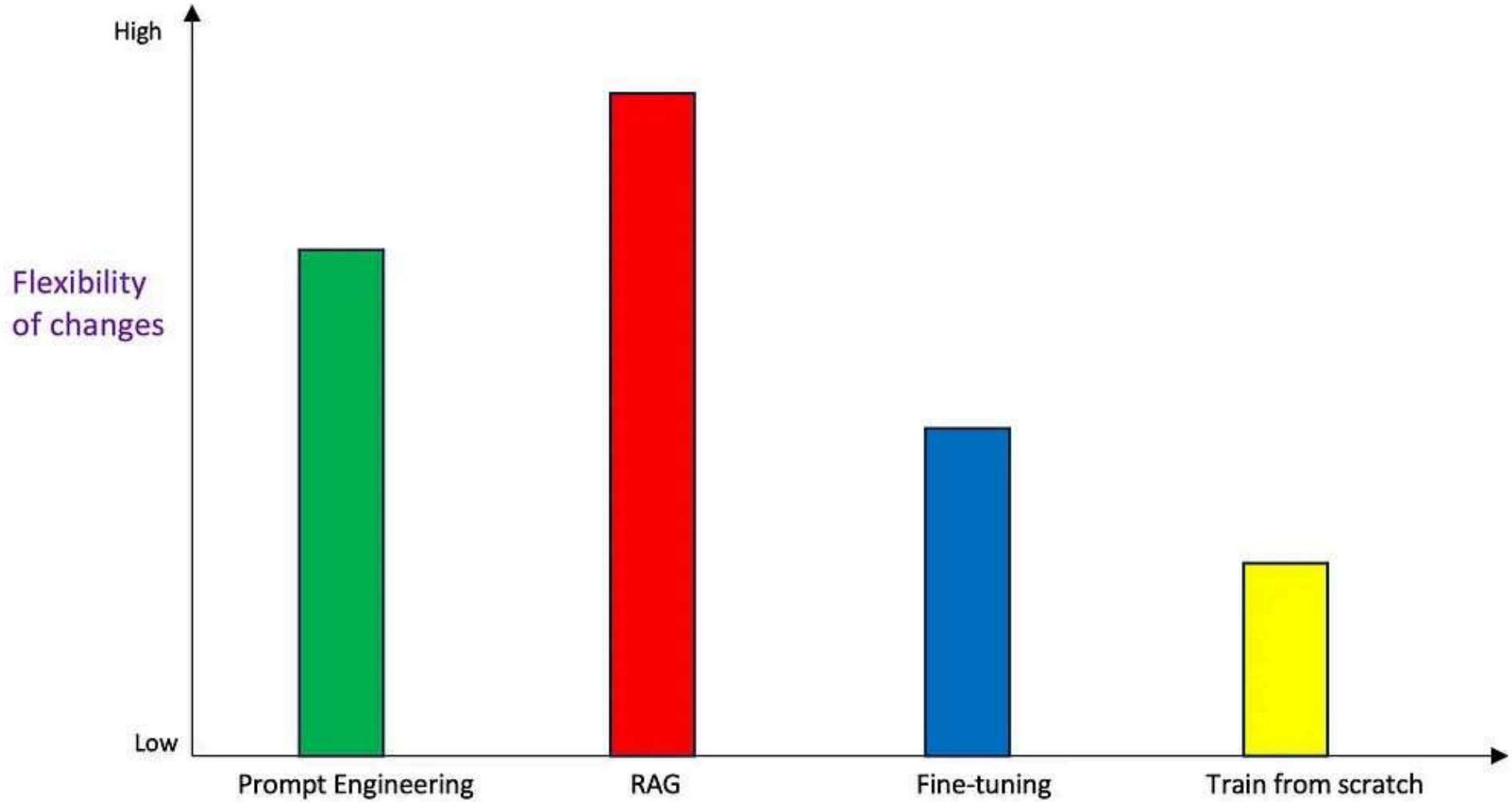


Hard (setup)
Slow (training time)
Static (fixed knowledge)
Accurate (task-specific)
Expensive (resources)


External Knowledge Required
High
Low
RAG
Hybrid
RAG + Fine-Tuning
Prompt Engineering
Fine-Tuning
Low
High
Model adaptation required
(eg. behaviour, writing style, vocabulary)


High
Accuracy
Low
Prompt Engineering
RAG
Fine-tuning
Train from scratch


High
Implementation complexity
Low
Prompt Engineering
RAG
Fine-tuning
Train from scratch


High
Flexibility
of changes
Low
Prompt Engineering
RAG
Fine-tuning
Train from scratch

The Classic RAG Workflow

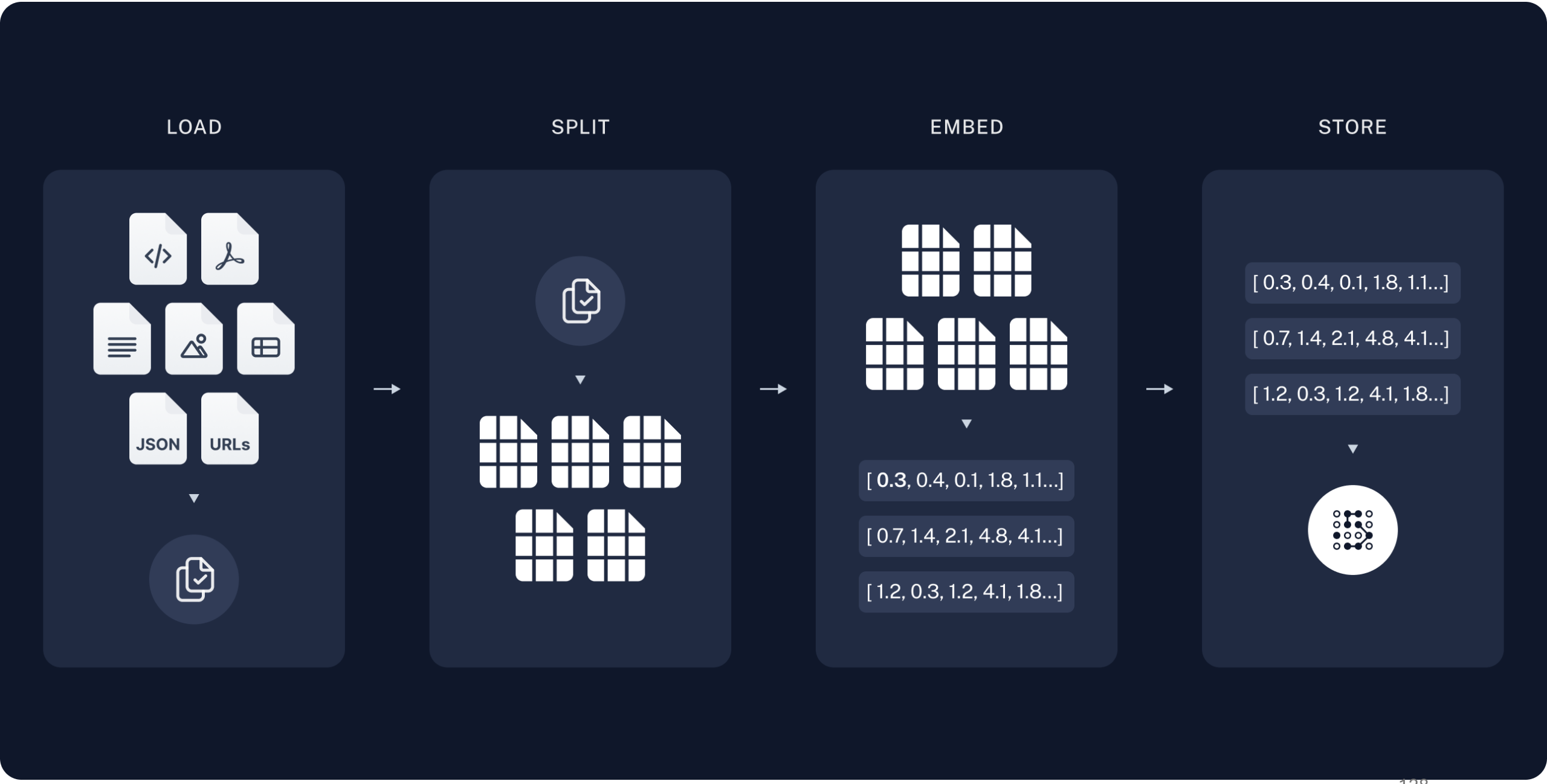
- A typical RAG system (depicted in the diagram below) handles three key data-related components:
- An **LLM** that has acquired knowledge from the data it has been trained with, typically millions to billions of text documents.
- A **vector database**, also called **knowledge base** storing text documents.
  - Vectors represent words, sentences, or entire documents, maintaining key properties of the original text such that two similar vectors are associated with words, sentences, or pieces of text with similar semantics.
  - Storing text as numerical vectors enhances the system's efficiency, such that relevant documents are quickly found and retrieved.
- **Queries or prompts** (for AI chat bots) formulated by the user in natural language.

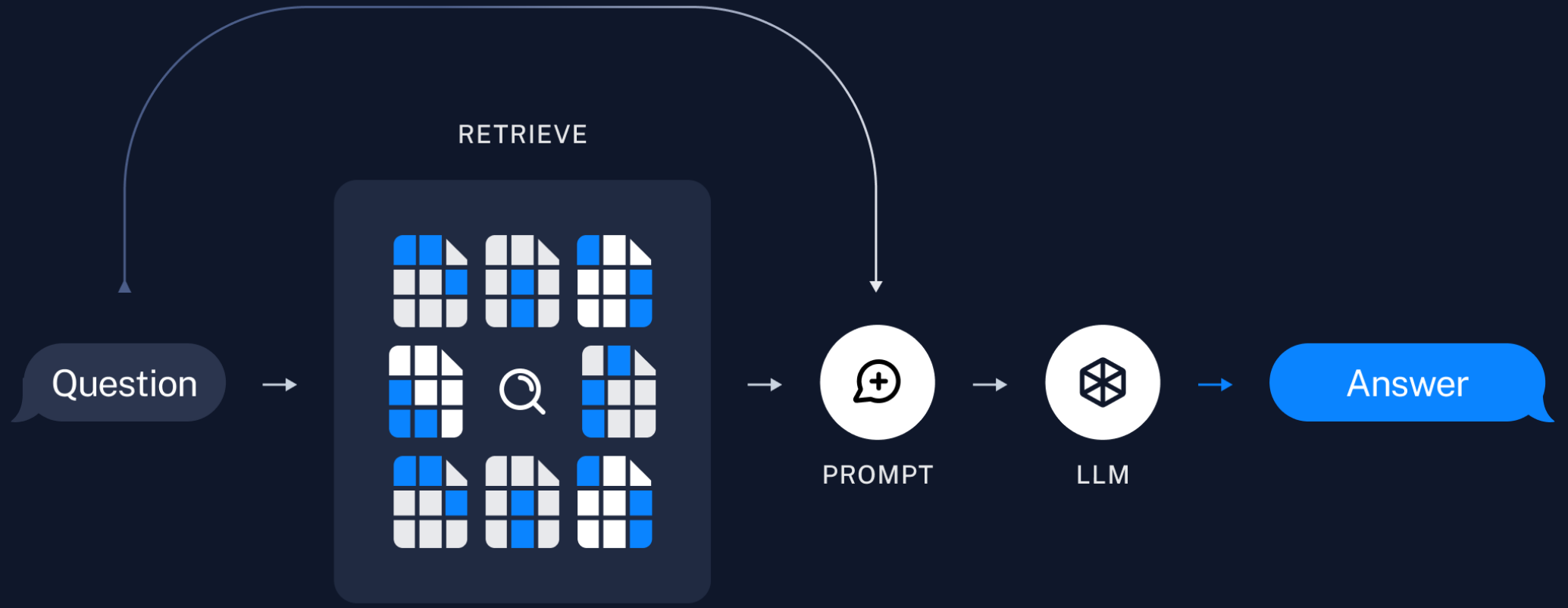
RAG - Ingestion

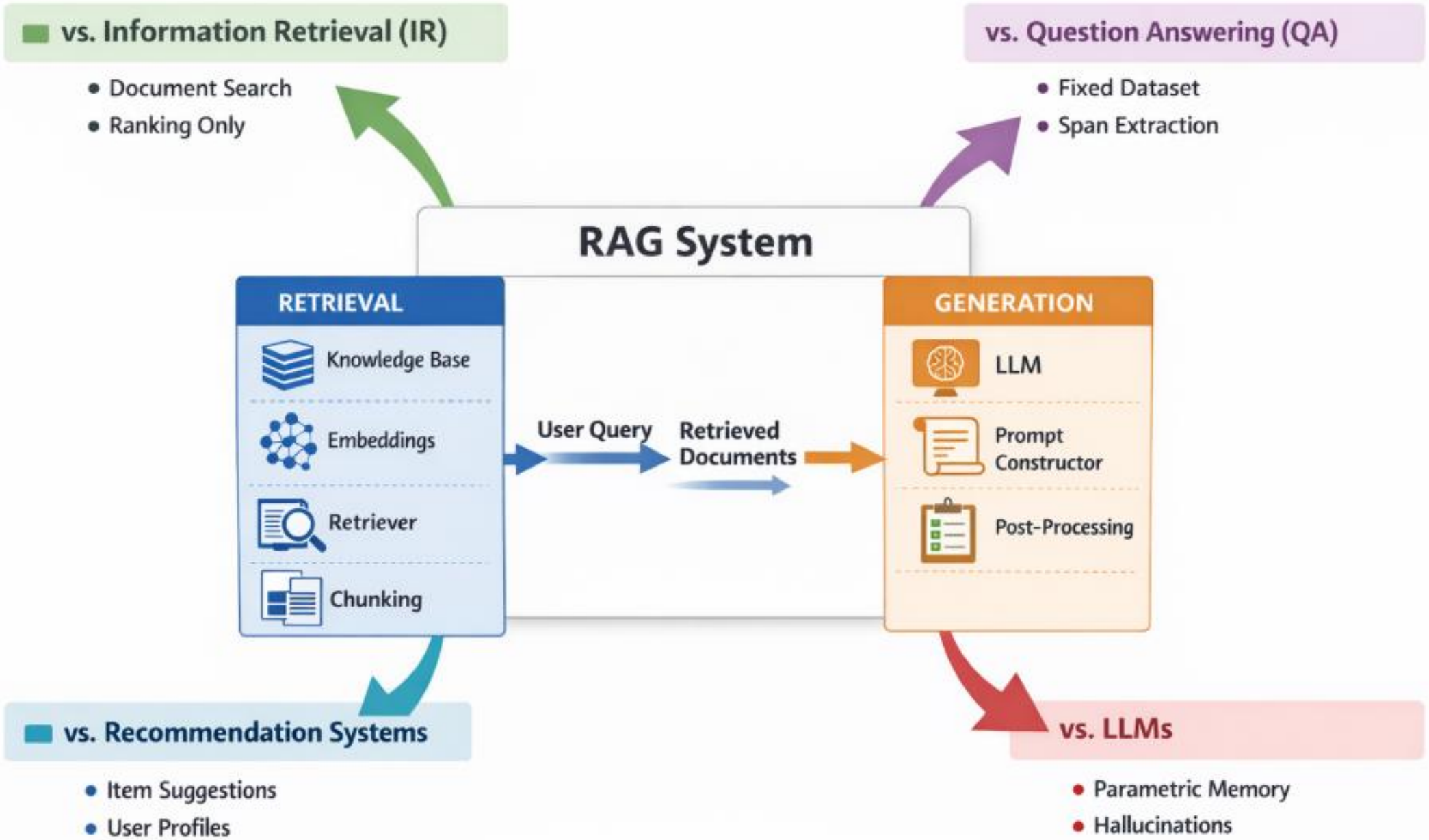
- In simple traditional RAG, you'll retrieve data from a vector database, using semantic search to find the true meaning of the user's query and retrieve relevant information instead of simply matching keywords in the query.

But before we can retrieve the data, you have to ingest the data. Here are steps to get data into your database:

- Chunk the data
- Create vector embeddings
- Load data into a vector database


LOAD
SPLIT
EMBED
STORE
JSON
URLs
[0.3, 0.4, 0.1, 1.8, 1.1...]
[0.7, 1.4, 2.1, 4.8, 4.1...]
[1.2, 0.3, 1.2, 4.1, 1.8...]
[0.3, 0.4, 0.1, 1.8, 1.1...]
[0.7, 1.4, 2.1, 4.8, 4.1...]
[1.2, 0.3, 1.2, 4.1, 1.8...]


RETRIEVE
Question
PROMPT
LLM
Answer


vs. Information Retrieval (IR)
Document Search
Ranking Only
vs. Question Answering (QA)
Fixed Dataset
Span Extraction
RAG System
RETRIEVAL
Knowledge Base
Embeddings
Retriever
Chunking
User Query
Retrieved Documents
GENERATION
LLM
Prompt Constructor
Post-Processing
vs. Recommendation Systems
Item Suggestions
User Profiles
vs. LLMs
Parametric Memory
Hallucinations

RAG – Ingestion – Chunking - considerations

- Vector count: How many chunks you will create directly affects the vector count, which in turn affects search time.
- Vector quality: How you chunk your data also determines the semantic coherence and search precision of each vector.
- Index structure: Chunk characteristics affect clustering efficiency and index transversal speed.

RAG – Ingestion - Chunking

- Several variables play a role in determining the best chunking strategy, and these variables vary depending on the use case. Here are some key aspects to keep in mind:
- **What kind of data is being chunked?** Are you working with long documents, such as articles or books, or shorter content, like tweets, product descriptions, or chat messages? Small documents may not need to be chunked at all, while larger ones may exhibit certain structure that will inform chunking strategy, such as sub-headers or chapters.
- **Which embedding model are you using?** Different embedding models have differing capacities for information, especially on specialized domains like code, finance, medical, or legal information. And, the way these models are trained can strongly affect how they perform in practice. After choosing an appropriate model for your domain, be sure to adapt your chunking strategy to align with expected document types the model has been trained on.
- **What are your expectations for the length and complexity of user queries?** Will they be short and specific or long and complex? This may inform the way you choose to chunk your content as well so that there's a closer correlation between the embedded query and embedded chunks.
- **How will the retrieved results be utilized within your specific application?** For example, will they be used for semantic search, question answering, retrieval augmented generation, or even an agentic workflow? For example, the amount of information a human may review from a search result may be smaller or larger than what an LLM may need to generate a response. These users determine how your data should be represented within the vector database.

RAG – Ingestion – Chunking – Fixed-size chunking

- This is the most common and straightforward approach to chunking: we simply decide the number of tokens in our chunk, and use this number to break up our documents into fixed size chunks.
- Usually, this number is the max context window size of the embedding model (such as 1024 for llama-text-embed-v2, or 8196 for text-embedding-3-small).
- Keep in mind that different embedding models may tokenize text differently, so you will need to estimate token counts accurately.
- Fixed-sized chunking will be the best path in most cases, and we recommend starting here and iterating only after determining it insufficient.

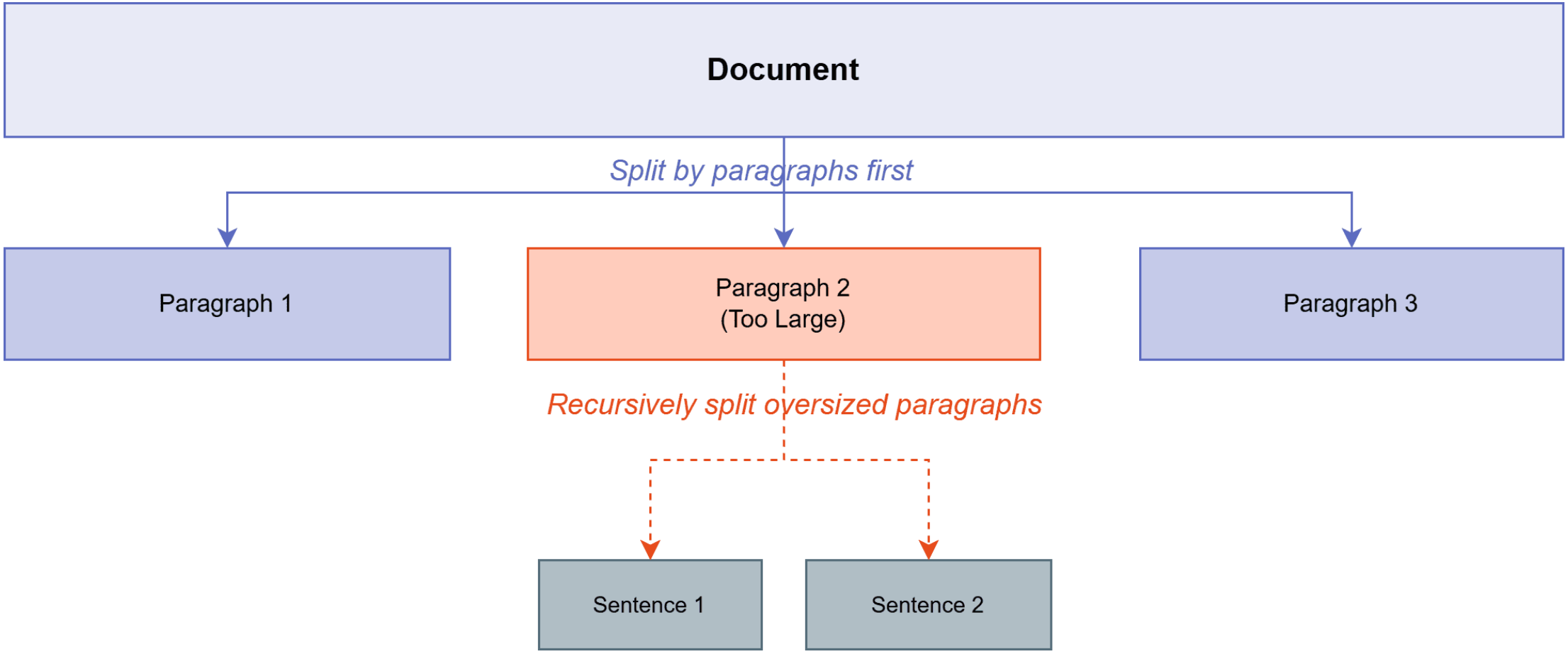
RAG – Ingestion – Chunking – Simple Sentence and Paragraph splitting

- As we mentioned before, some embedding models are optimized for embedding sentence-level content. But sometimes, sentences need to be mined from larger text datasets that aren't preprocessed. In these cases, it's necessary to use sentence chunking, and there are several approaches and tools available to do this:
- **Naive splitting:** The most naive approach would be to split sentences by periods (“.”), new lines, or white space.
- **NLTK**: The Natural Language Toolkit (NLTK) is a popular Python library for working with human language data. It provides a trained sentence tokenizer that can split the text into sentences, helping to create more meaningful chunks.
- **spaCy**: spaCy is another powerful Python library for NLP tasks. It offers a sophisticated sentence segmentation feature that can efficiently divide the text into separate sentences, enabling better context preservation in the resulting chunks.

RAG – Ingestion – Chunking – Recursive Character Level Chunking

- LangChain implements a RecursiveCharacterTextSplitter that tries to split text using separators in a given order.
- The default behavior of the splitter uses the ["\n\n", "\n", " ", ""] separators to break paragraphs, sentences and words depending on a given chunk size.
- This is a great middle ground between always splitting on a specific character and using a more semantic splitter, while also ensuring fixed chunk sizes when possible.

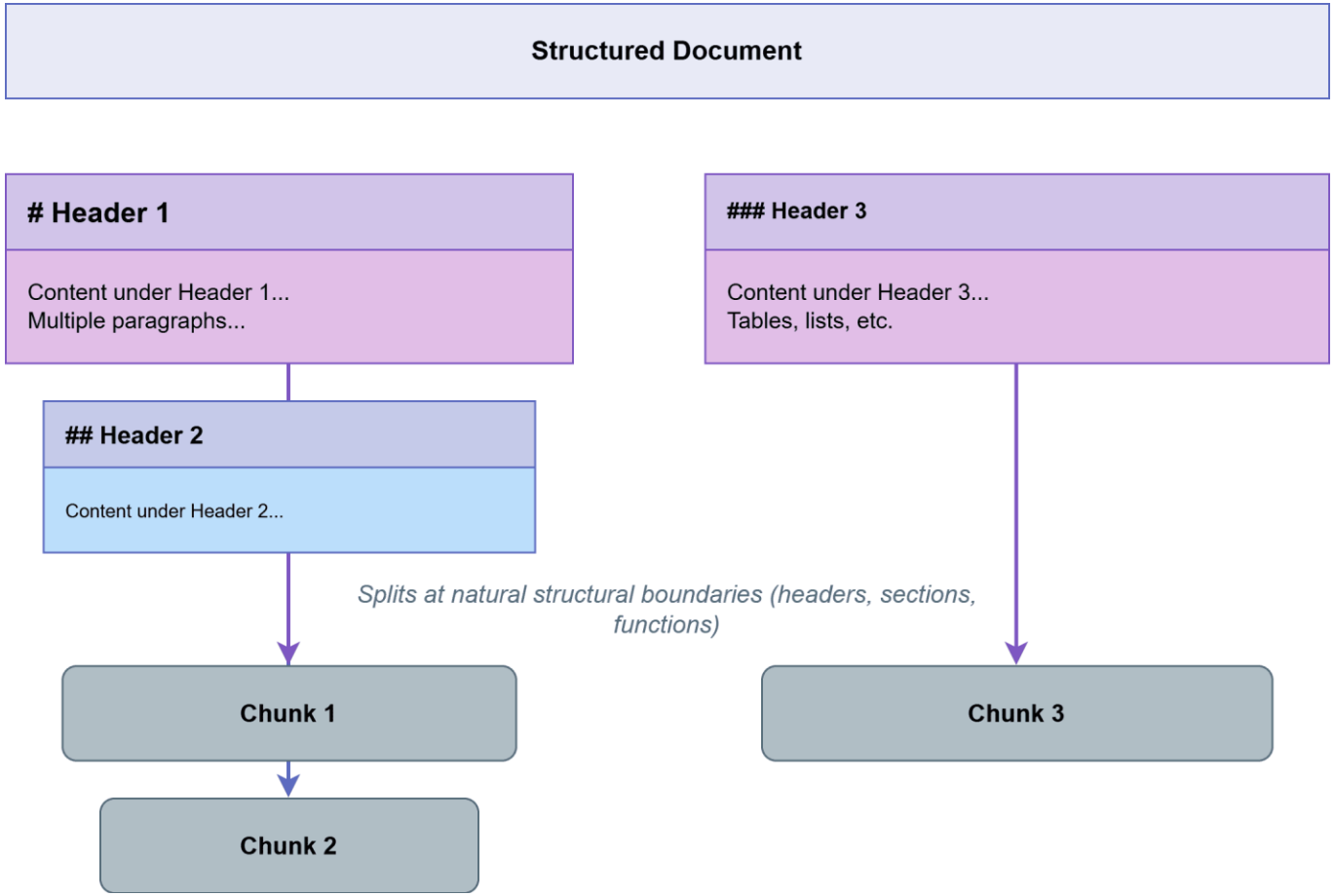
RAG – Ingestion – Chunking – Recursive Character Level Chunking



RAG – Ingestion – Chunking – Document structure-based chunking

- When chunking large documents such as PDFs, DOCX, HTML, code snippets, Markdown files and LaTex, specialized chunking methods can help preserve the original structure of the content during chunk creation.
- PDF documents contain loads of headers, text, tables, and other bits and pieces that require preprocessing to chunk. LangChain has some handy utilities to help process these documents, while Pinecone Assistant can chunk and processes these for you
- HTML, from scraped web pages can contain tags (<p> for paragraphs, or <title> for titles) that can inform text to be broken up or identified, like on product pages or blog posts. Roll your own parser, or use LangChain splitters here to process these for chunking
- Markdown: Markdown is a lightweight markup language commonly used for formatting text. By recognizing the Markdown syntax (e.g., headings, lists, and code blocks), you can intelligently divide the content based on its structure and hierarchy, resulting in more semantically coherent chunks.
- LaTex: LaTeX is a document preparation system and markup language often used for academic papers and technical documents. By parsing the LaTeX commands and environments, you can create chunks that respect the logical organization of the content (e.g., sections, subsections, and equations), leading to more accurate and contextually relevant results.

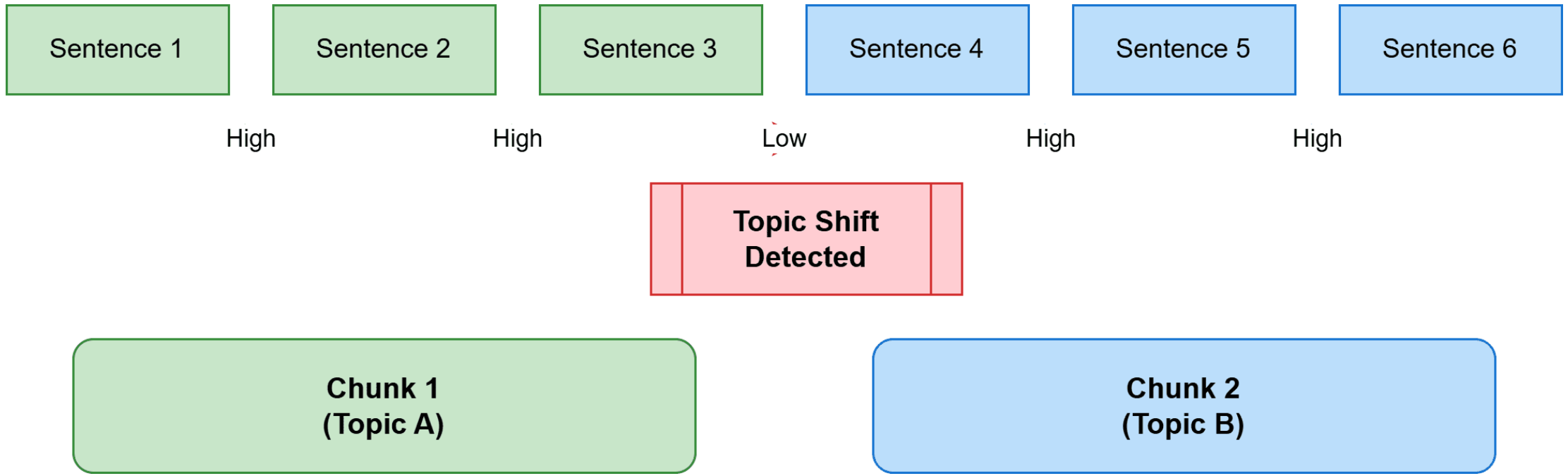
RAG – Ingestion – Chunking – Document structure-based chunking



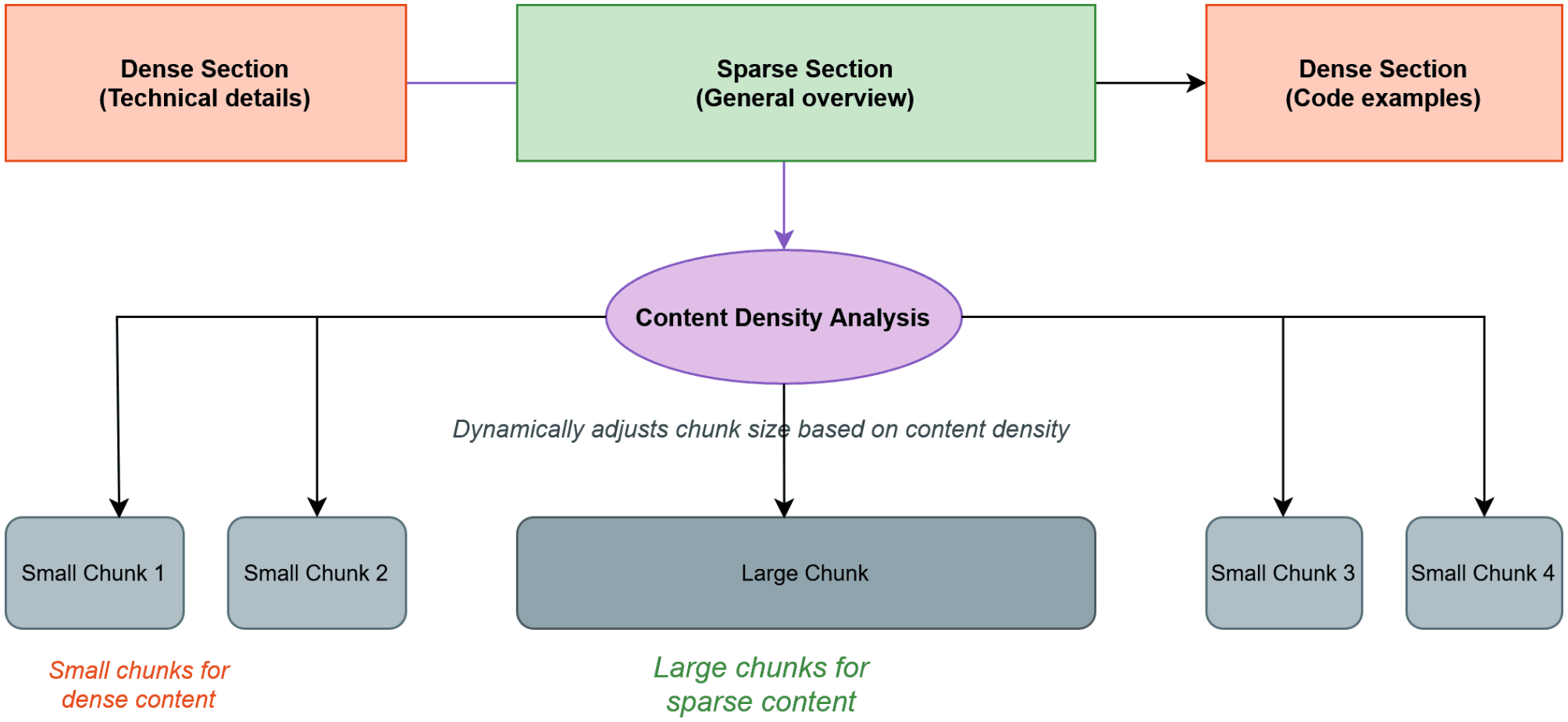
RAG – Ingestion – Chunking – Semantic Chunking

- A new experimental technique for approaching chunking was first introduced by Greg Kamradt. In his notebook, Kamradt rightfully points to the fact that a global chunking size may be too trivial of a mechanism to consider the meaning of segments within the document. If we use this type of mechanism, we can't know if we're combining segments that have anything to do with one another.
- Luckily, if you're building an application with LLMs, you most likely already can create embeddings - and embeddings can be used to extract the semantic meaning present in your data. This semantic analysis can be used to create chunks that are made up of sentences that talk about the same theme or topic.
- Semantic chunking involves breaking a document into sentences, grouping each sentence with its surrounding sentences, and generating embeddings for these groups. By comparing the semantic distance between each group and its predecessor, you can identify where the topic or theme shifts, which defines the chunk boundaries. You can learn more about applying semantic chunking with Pinecone here.

RAG – Ingestion – Chunking – Semantic Chunking



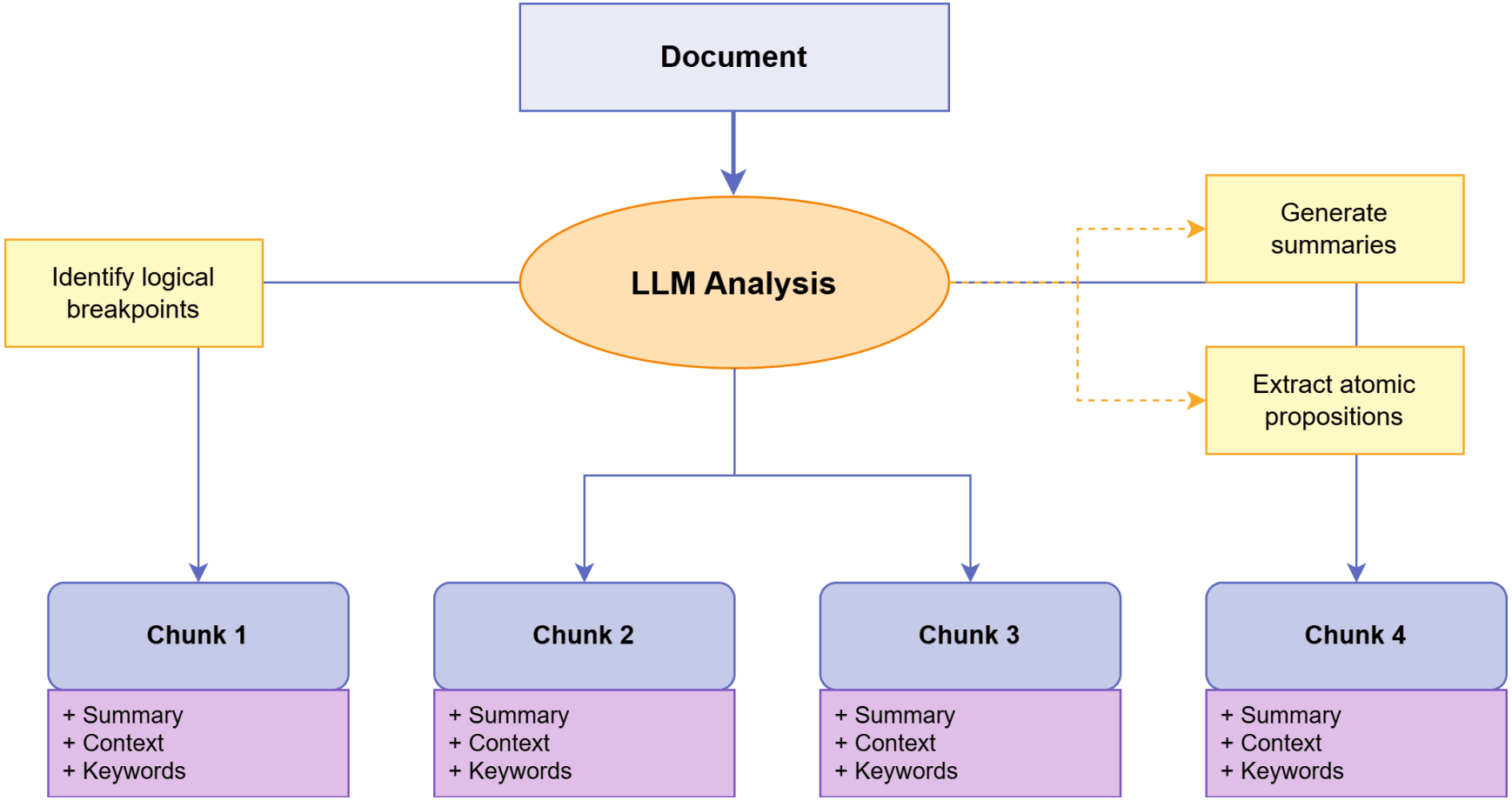
RAG – Ingestion – Chunking – Adaptive Chunking



RAG – Ingestion – Chunking – Contextual Chunking with LLMs

- Sometimes, it's not possible to chunk information from a larger complex document without losing the context entirely. This can happen when the documents are many hundreds of pages, change topics frequently, or require understanding from many related portions of the document. Anthropic introduced contextual retrieval in 2024 to help address this problem.
- Anthropic prompted a Claude instance with an entire document and it's chunk, in order to generate a contextualized description, which is appended to the chunk and then embedded. The description helps retain the high-level summary meaning of the document to the chunk, which exposes this information to incoming queries. To avoid processing the document each time, it's cached within the prompt for all necessary chunks. You can learn more about contextual retrieval in our video here and our code example here.

RAG – Ingestion – Chunking – Contextual Chunking with LLMs



RAG – Ingestion – Create vector embeddings

- Then, using an embedding model, you'll embed each chunk and load it into the vector database. The embedding model is a special type of LLM that converts the data chunk into a vector embedding, a numerical representation of the data's meaning. This allows computers to search for similar items based on the vector representation of the stored data.

RAG - Vector Databases

- **Vector DB** - a specialized for the high-dimensional vectors

- Enable efficient similarity-based searches
  - For instance, vectors associated with two Mediterranean restaurant reviews would be relatively close

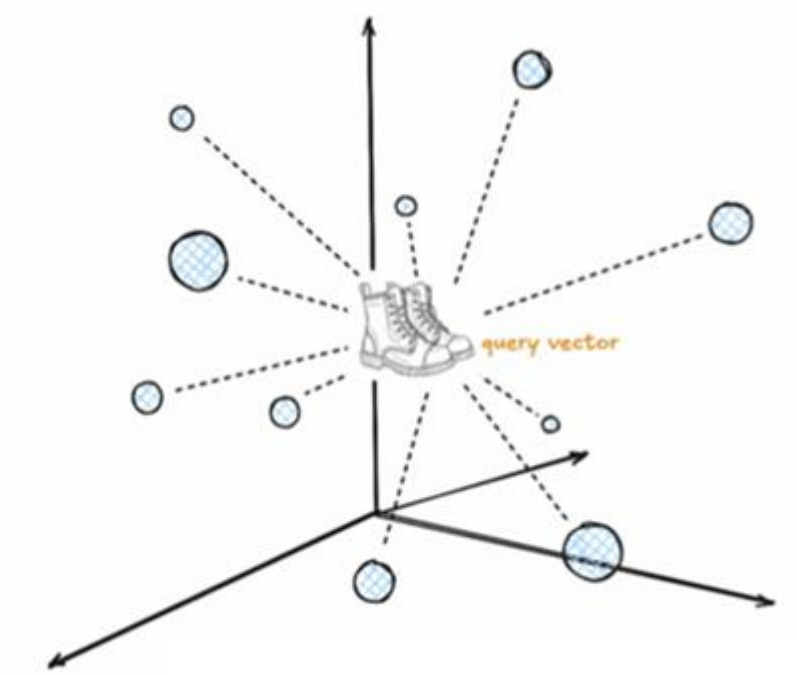
- Vector DB in IR - documents relevant to user's query will be retrieved efficiently

RAG – Ingestion – Load data into a vector database

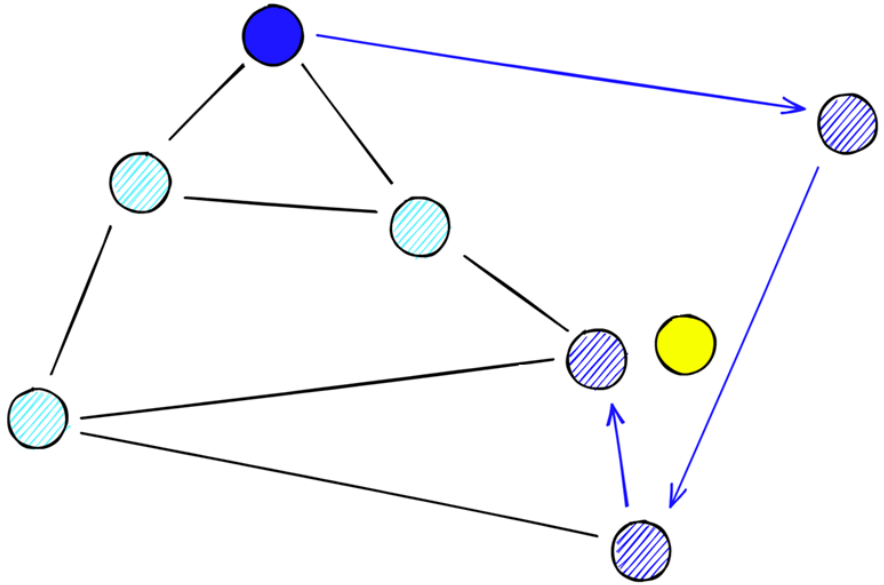
- Then, using an embedding model, you'll embed each chunk and load it into the vector database. The embedding model is a special type of LLM that converts the data chunk into a vector embedding, a numerical representation of the data's meaning. This allows computers to search for similar items based on the vector representation of the stored data.

RAG - Vector DB Indexing Strategies

Flat indexing

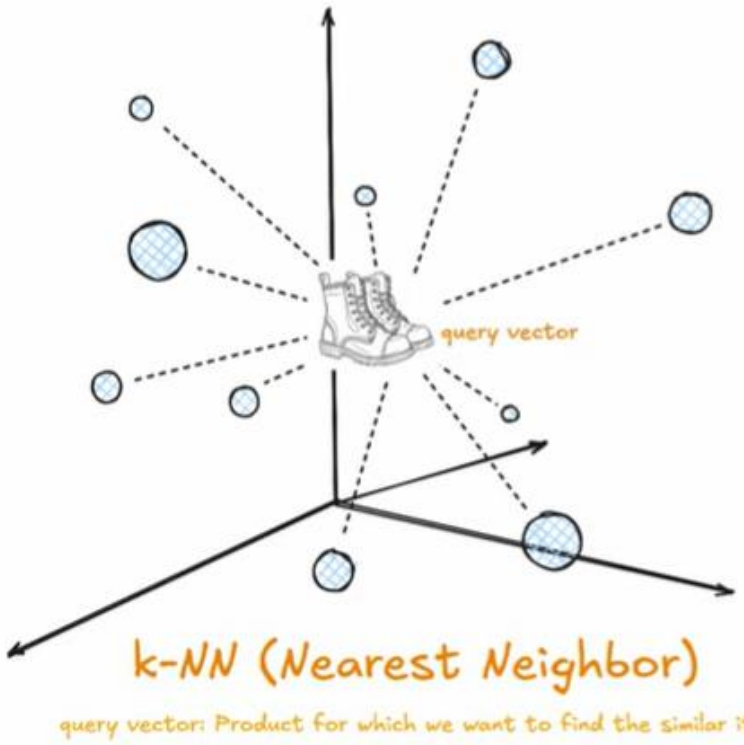


Approximating



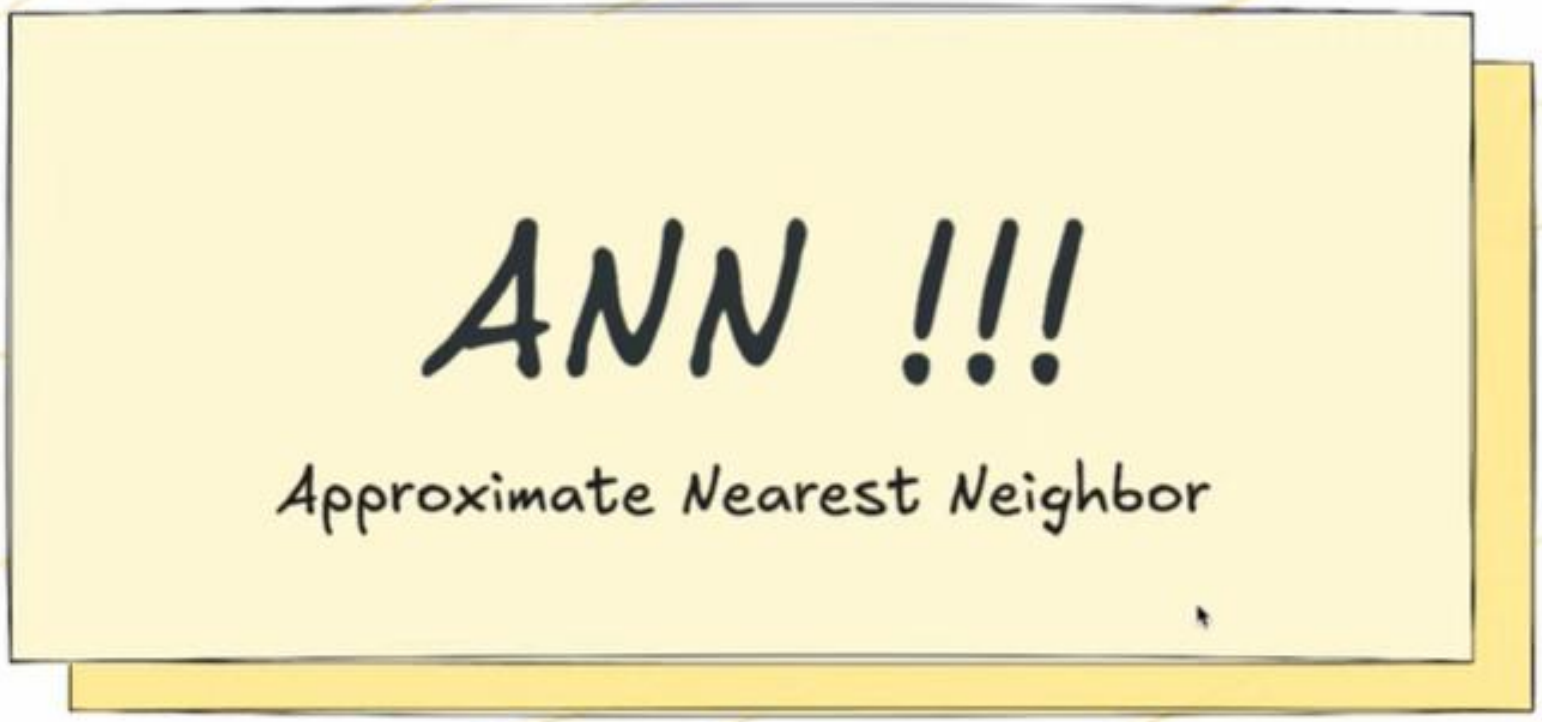
RAG - Indexing Strategies – Flat Indexing

Flat Indexing

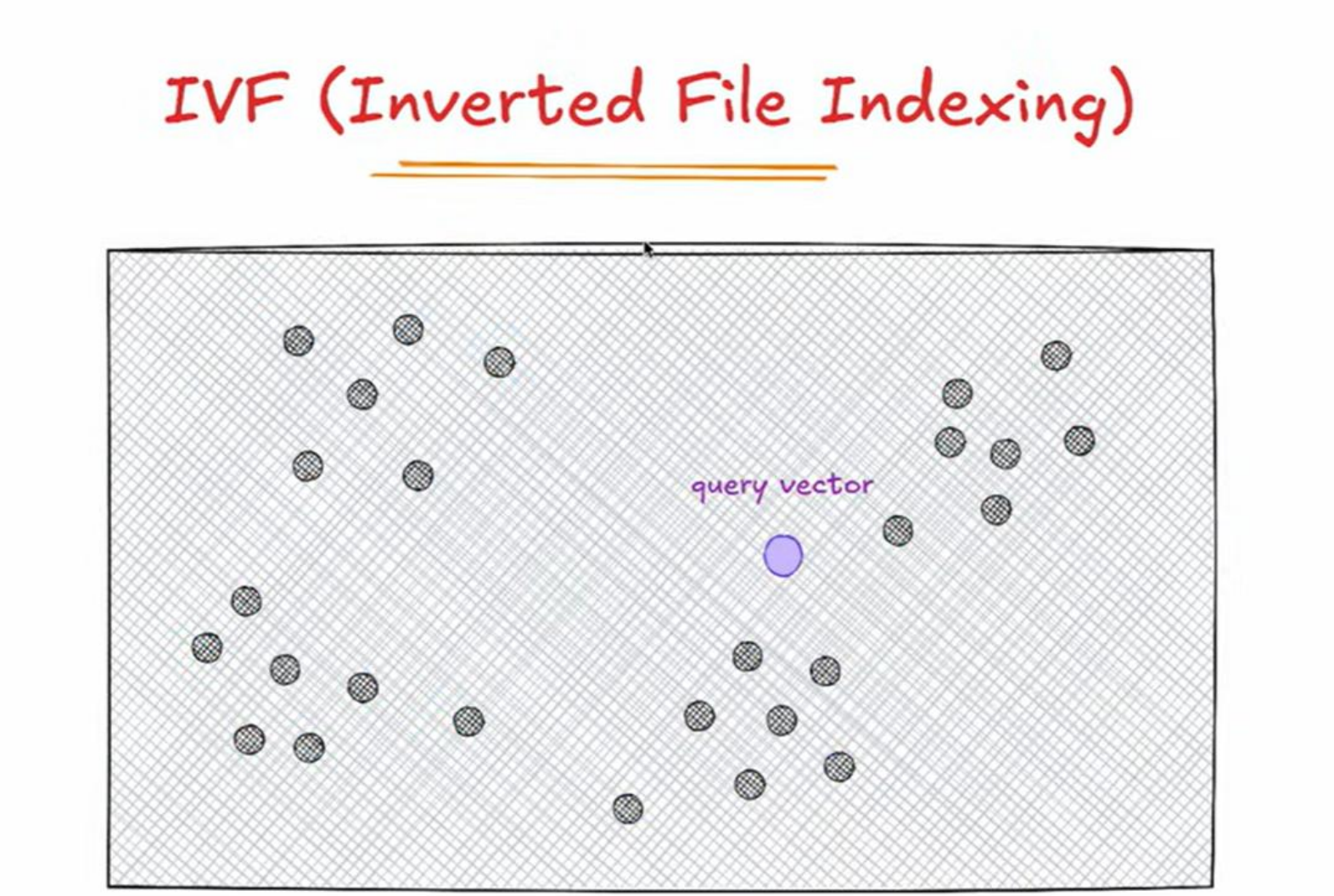


- Simplest form of indexing
- Finds the nearest neighbors betweenthe query embedding and every other embedding.
- Accurate but very slow for large datasets (billions or trillions)

RAG - Indexing Strategies - ANN

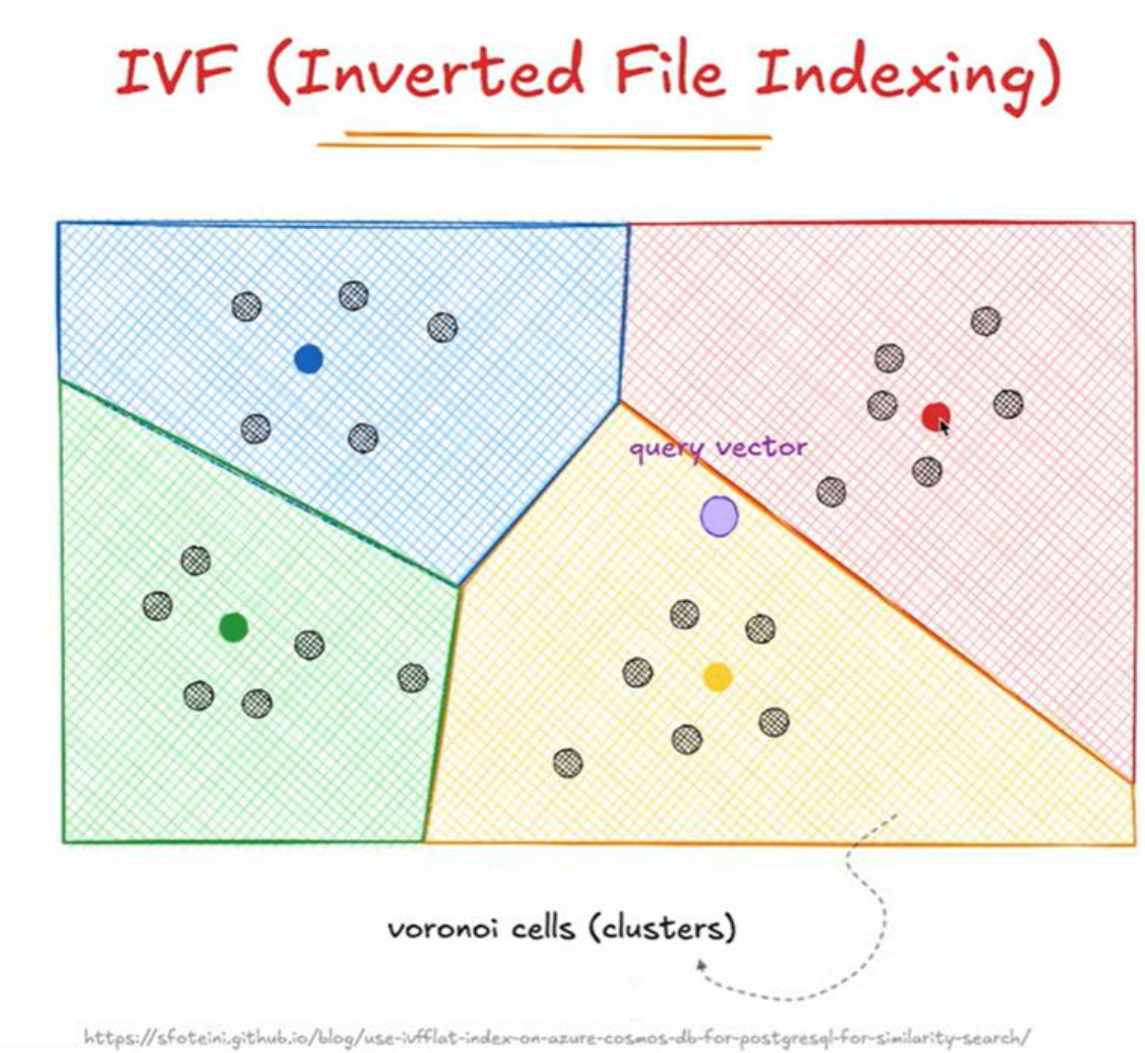


RAG - Indexing Strategies - IVF

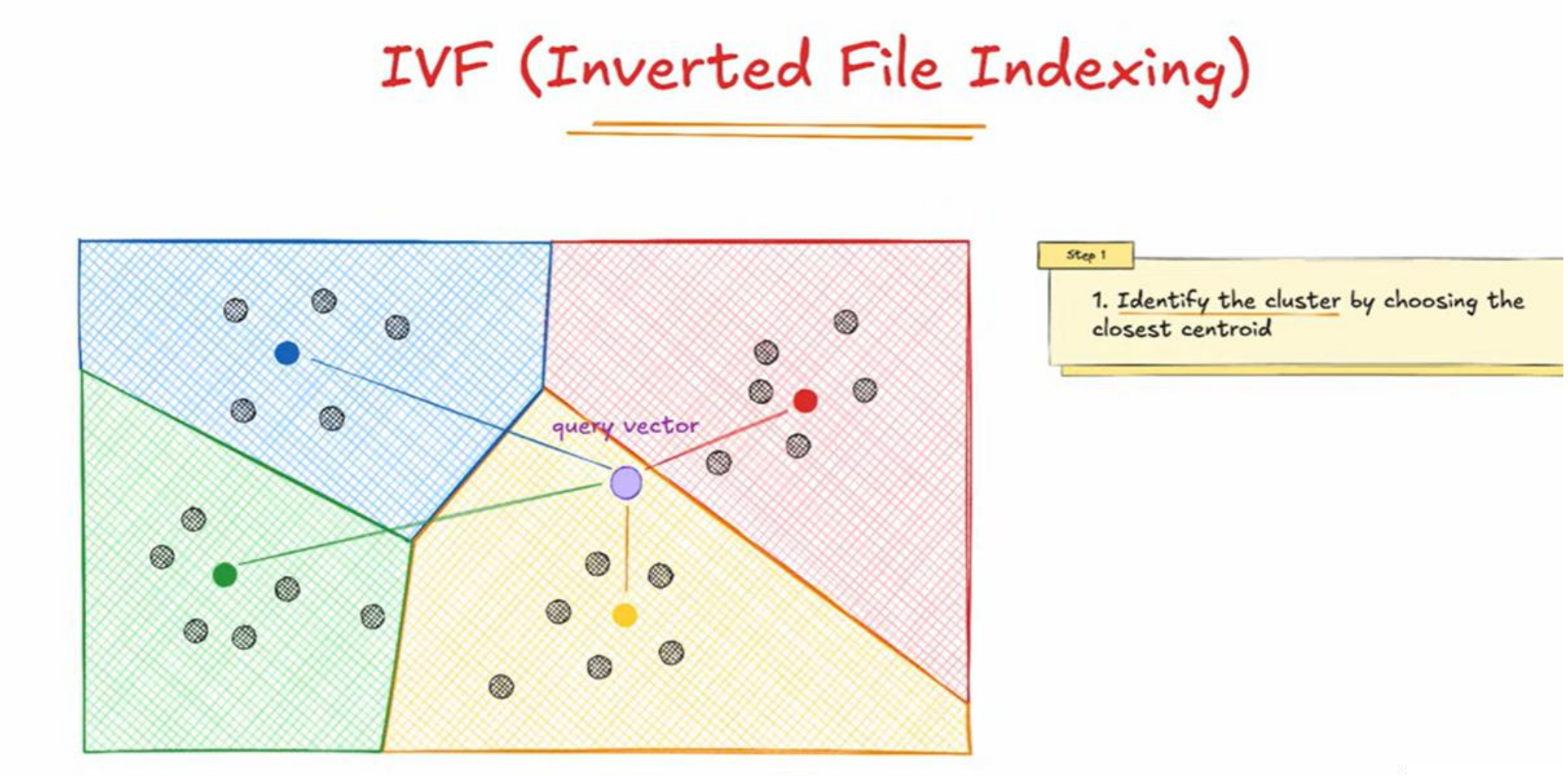


Credit: sfoteinin.github.io blog

RAG - Indexing Strategies - IVF

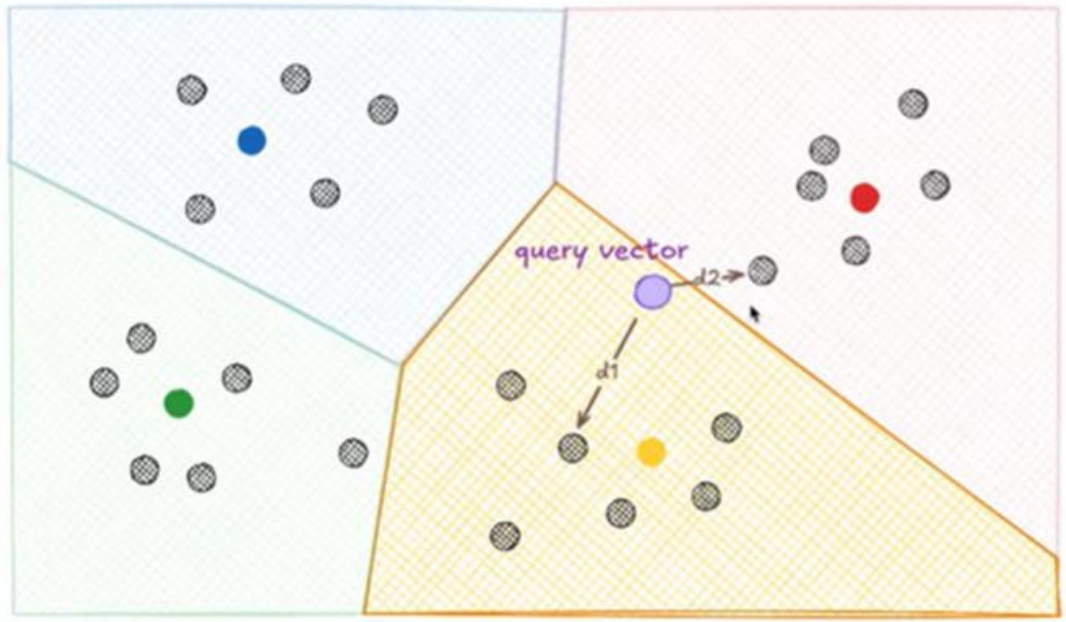


RAG - Indexing Strategies - IVF



RAG - Indexing Strategies - IVF

IVF (Inverted File Indexing)



Step 1

1. Identify the cluster by choosing the closest centroid

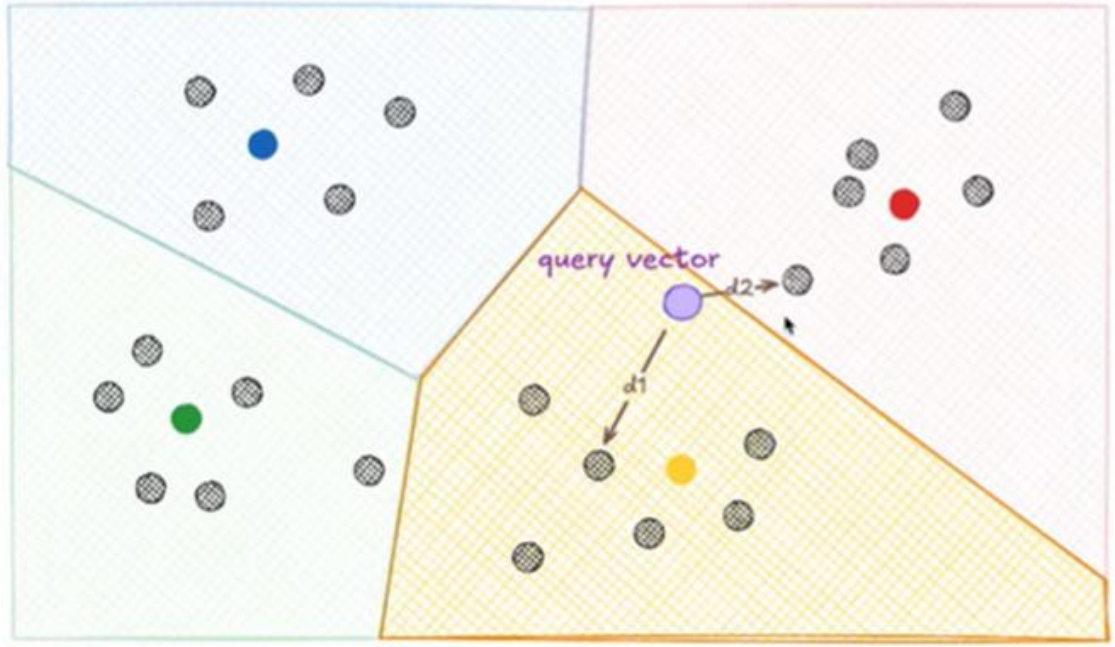
Step 2

2. From the chosen cluster calculate the similarity between all the points

https://sfoteini.github.io/blog/use-ivfflat-index-on-azure-cosmos-db-for-postgresql-for-similarity-search/

RAG - Vector DB Indexing Strategies

IVF (Inverted File Indexing)



Step 1

1. Identify the cluster by choosing the closest centroid

Step 2

2. From the chosen cluster calculate the similarity between all the points

Step 3

3. Return the top 10 similar items

https://sfoteini.github.io/blog/use-ivfflat-index-on-azure-cosmos-db-for-postgresql-for-similarity-search/

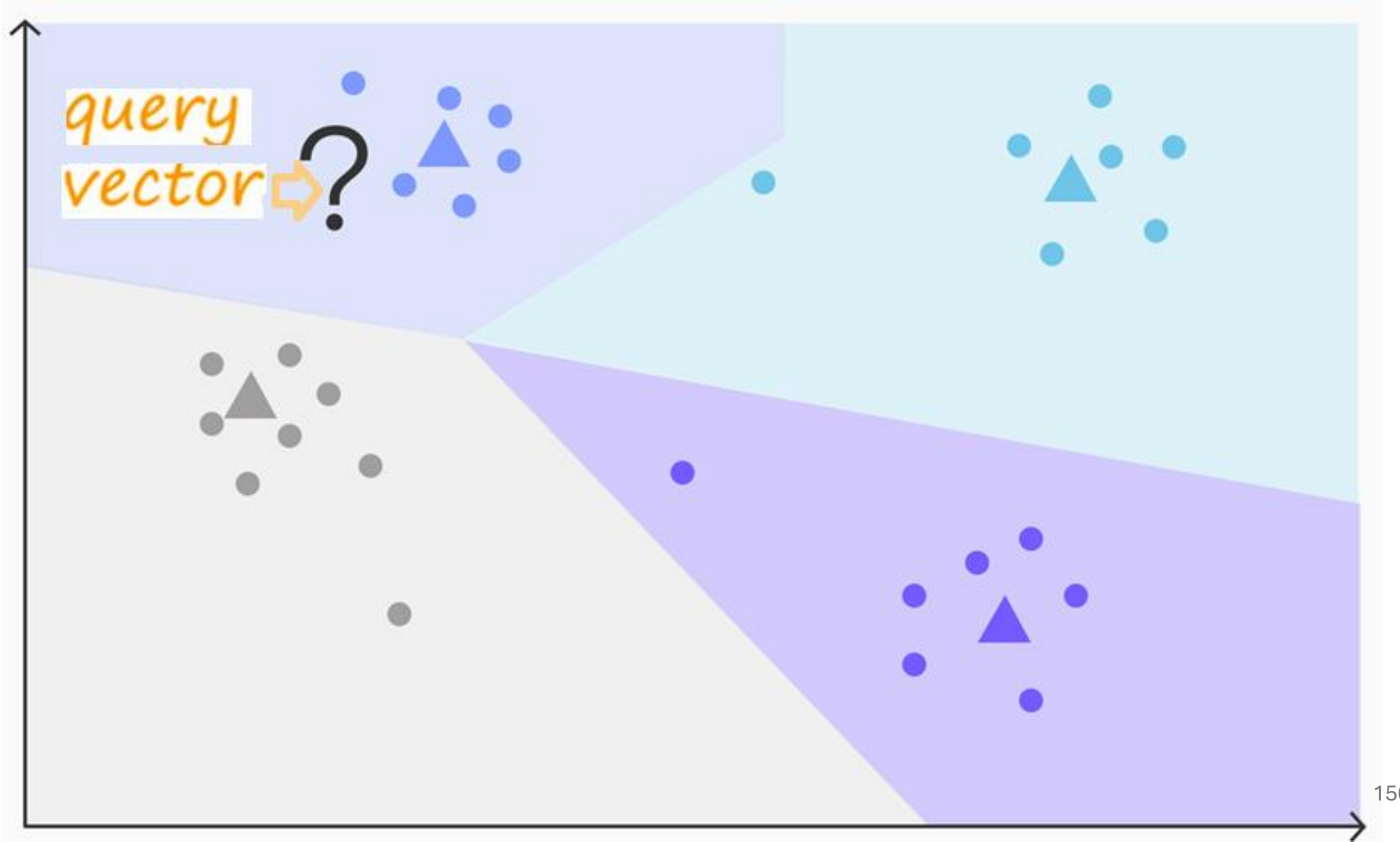
RAG - Indexing Strategies - IVF – The nList Parameter

IVF (Inverted File Index)

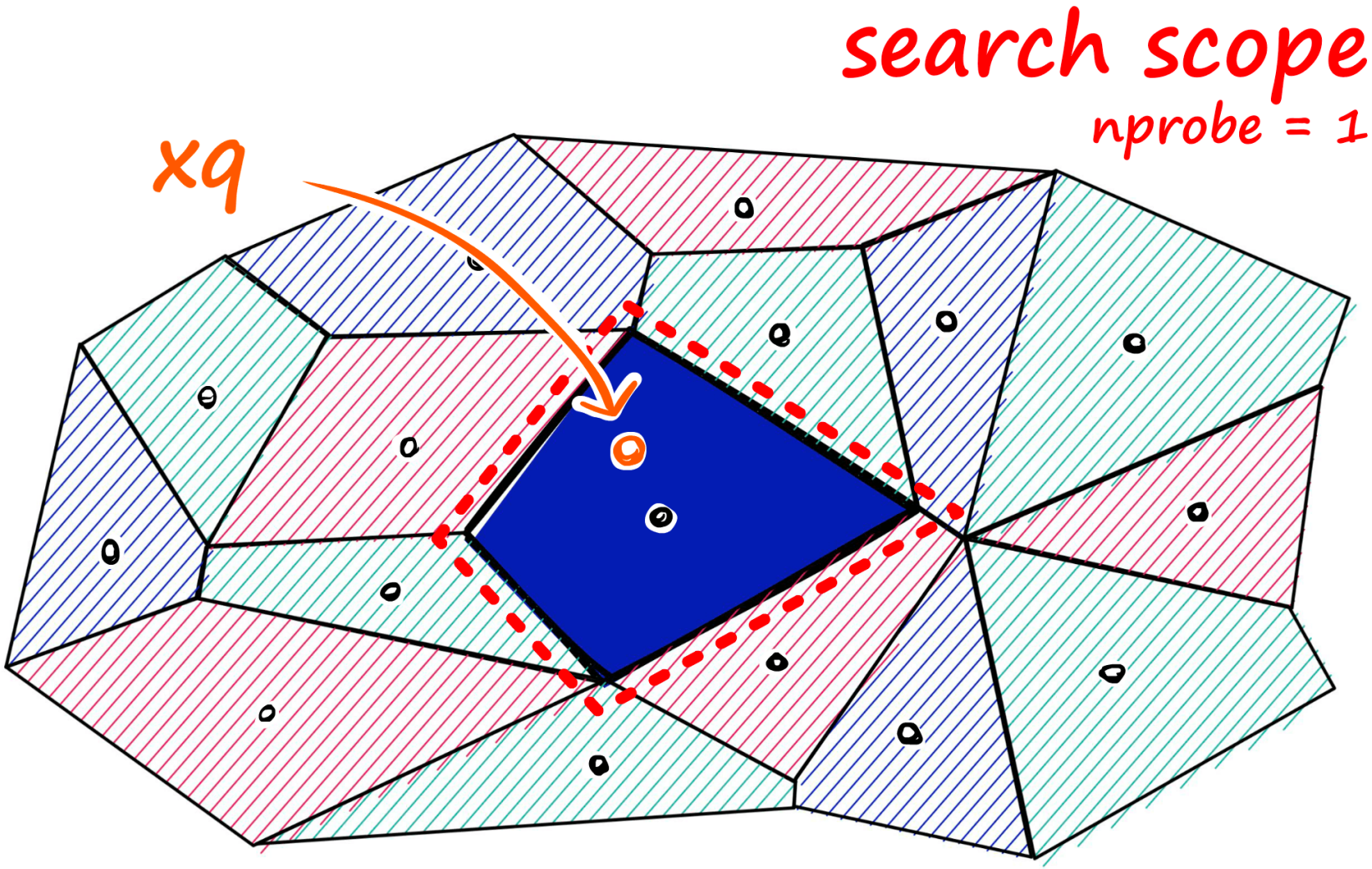
It partitions your vector space into clusters, searching only the relevant clusters. Great for large datasets, with a little less accuracy.

- ❑ nList parameter is the number of clusters
- ❑ Best practice: square root of dataset size
- ❑ Ideal for billion-scale datasets

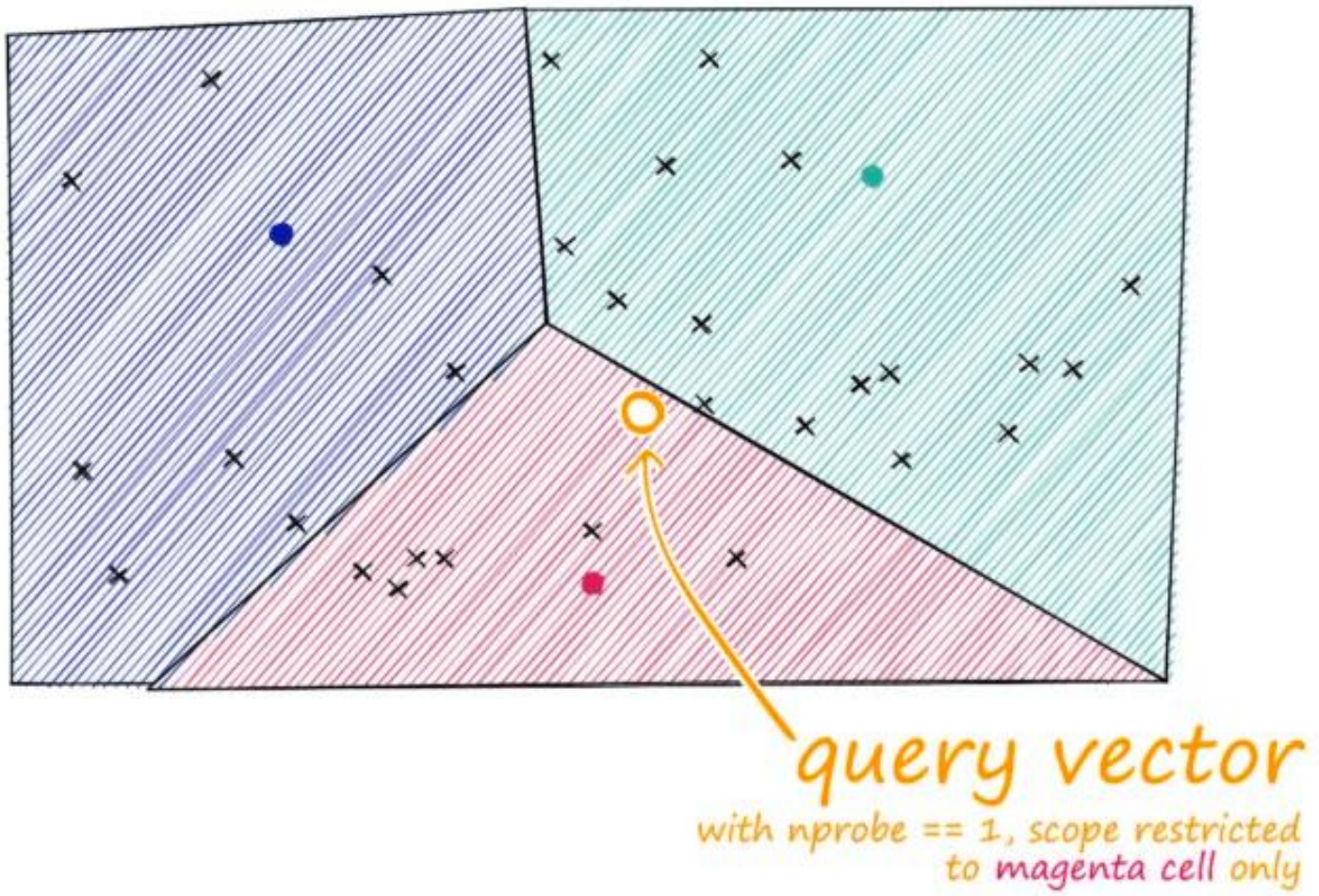
RAG - Indexing Strategies - IVF



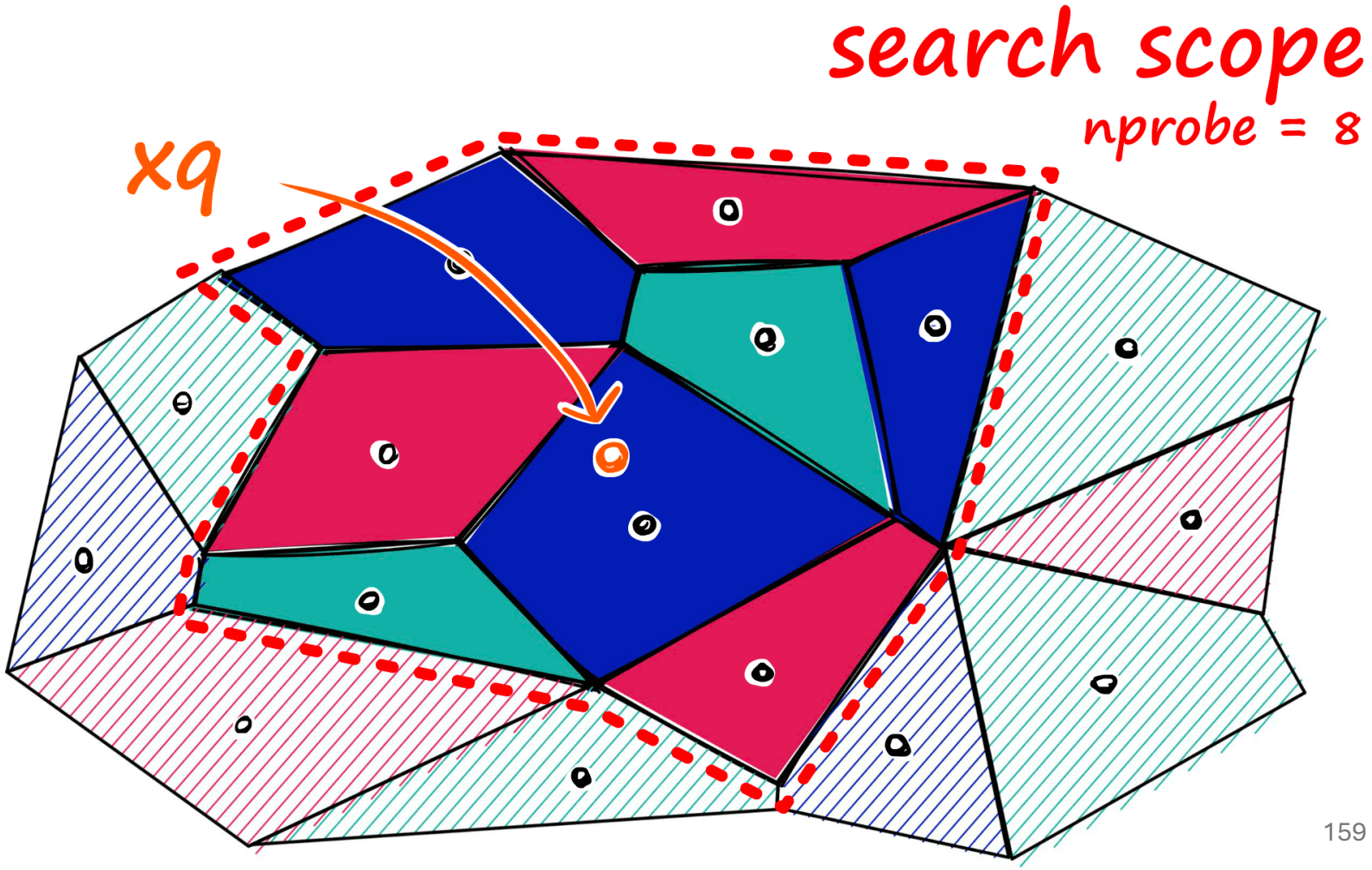
RAG - Indexing Strategies - IVF – search in closest cluster



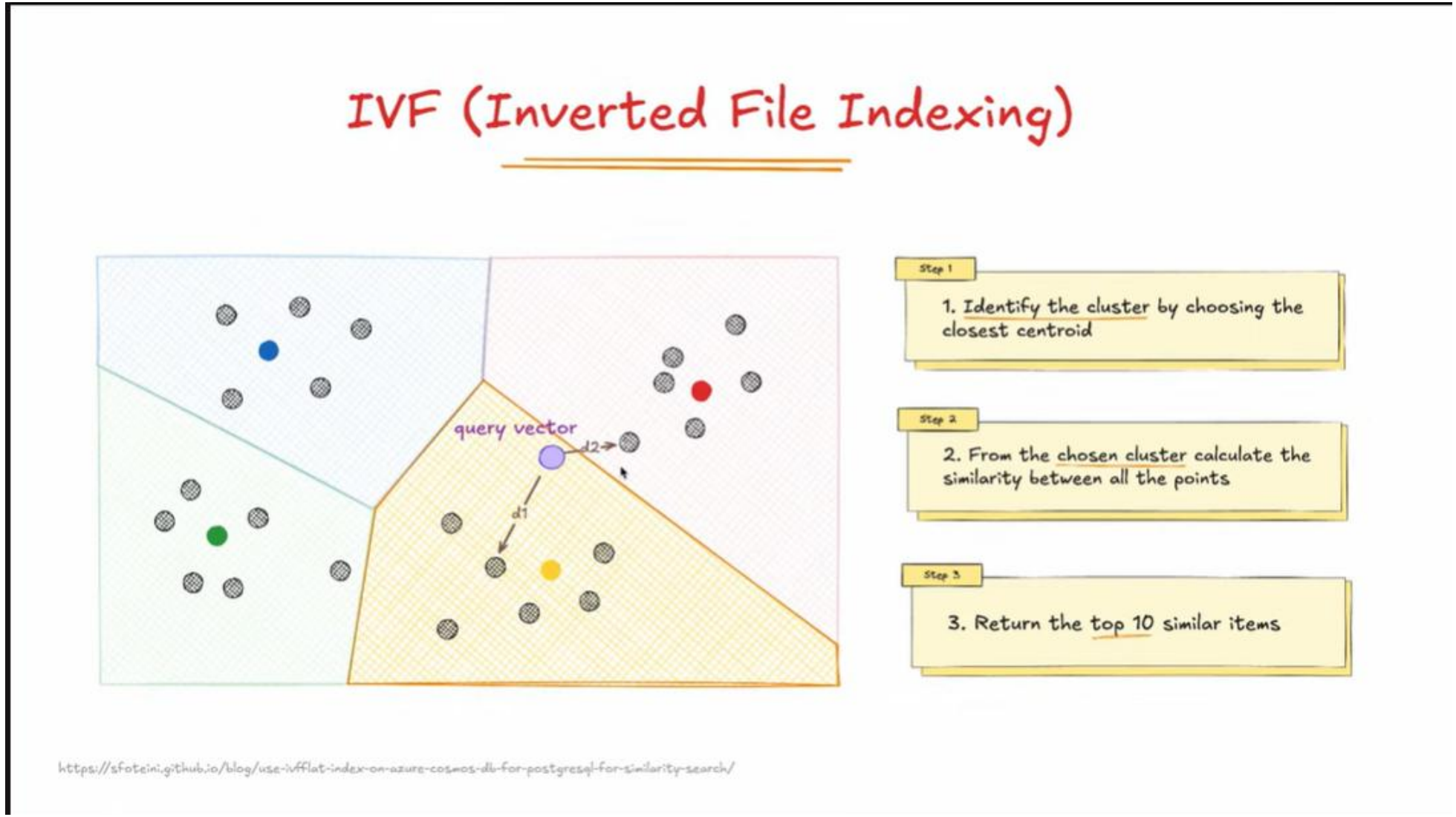
RAG - Indexing Strategies - IVF – search in closest cluster – insufficient



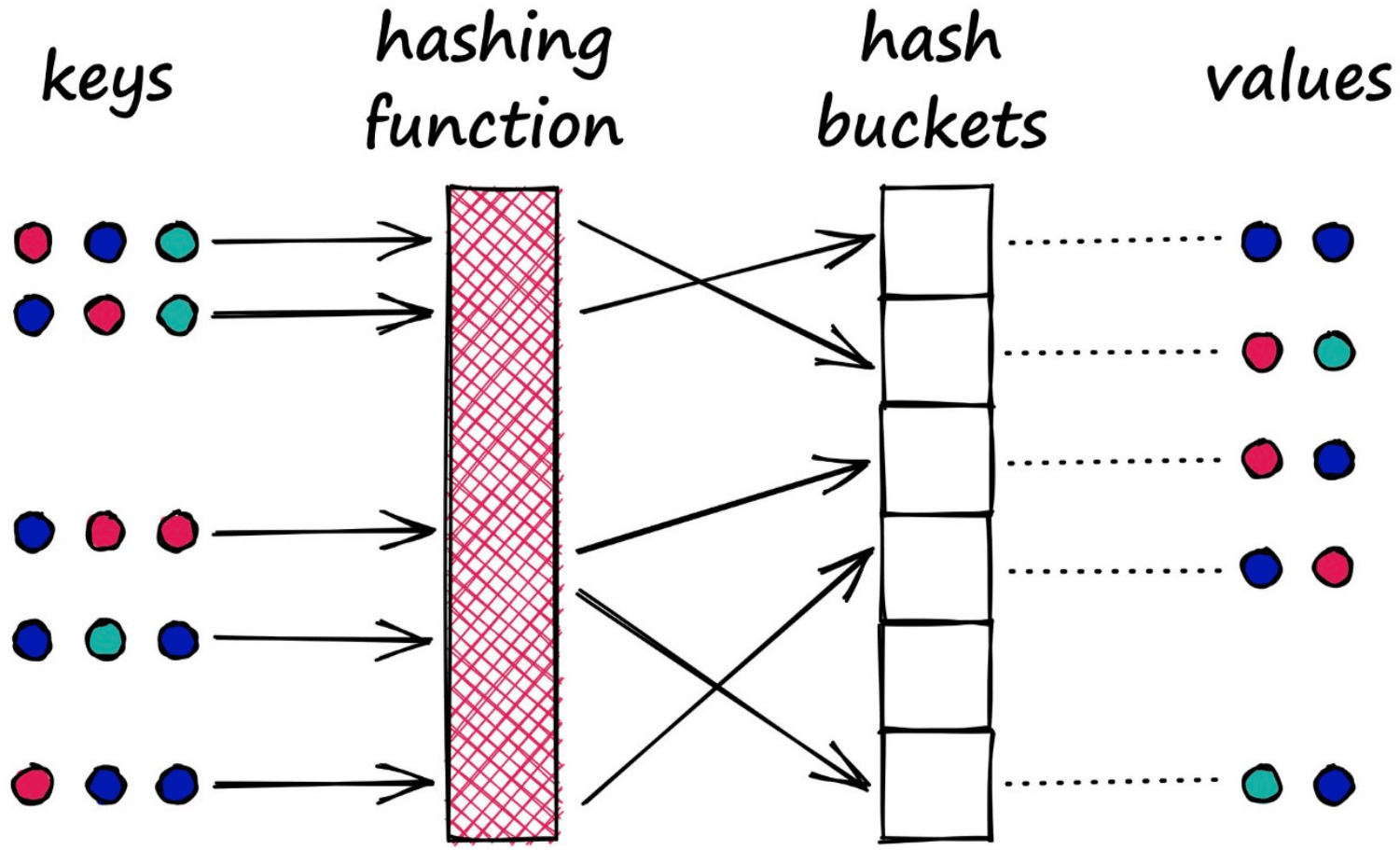
RAG - Indexing Strategies - IVF – Increase nprobe - Search in neighboring clusters



RAG - Indexing Strategies - IVF – Increase nprobe - Search in neighboring clusters

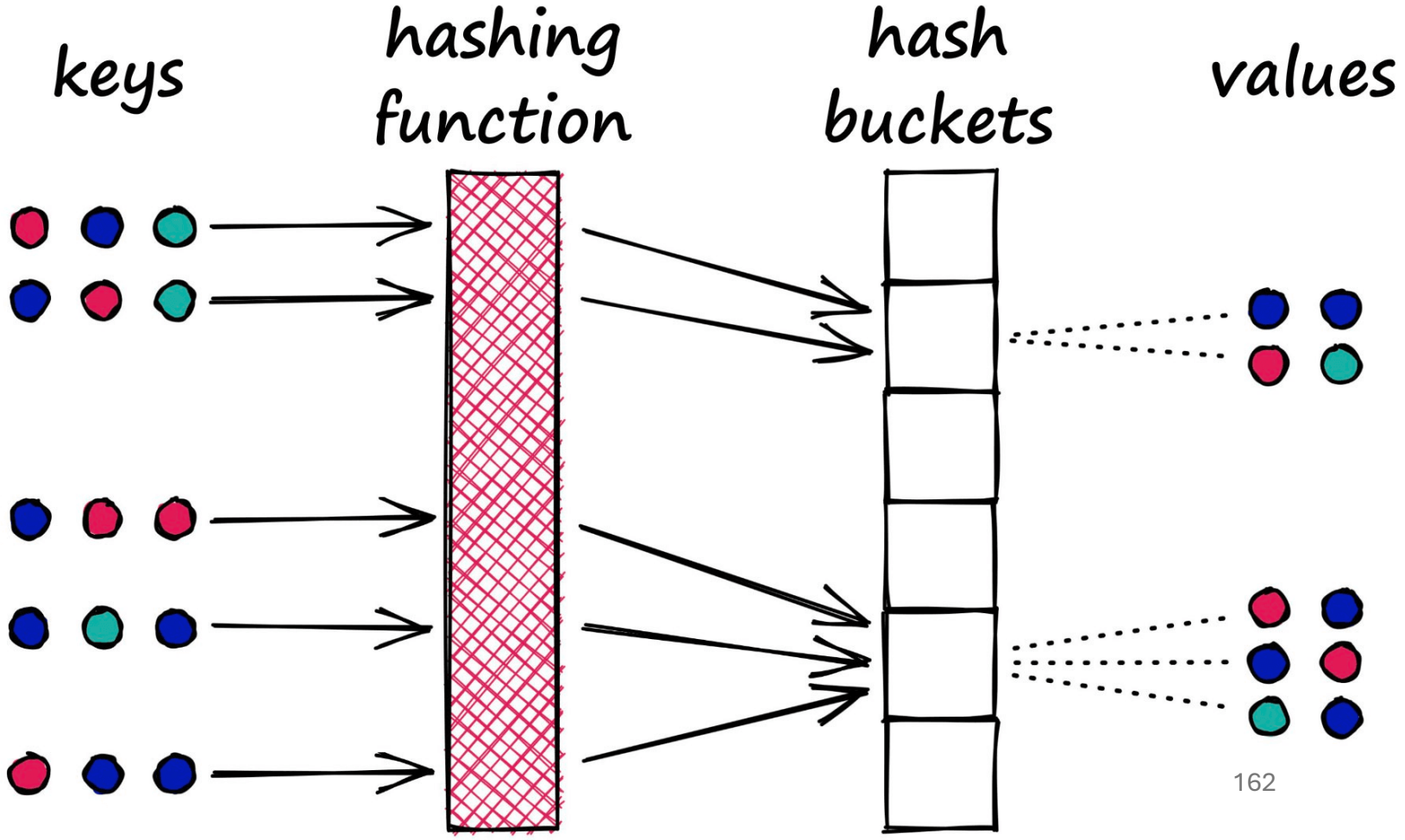


RAG - Indexing Strategies – LSH – A typical hash function



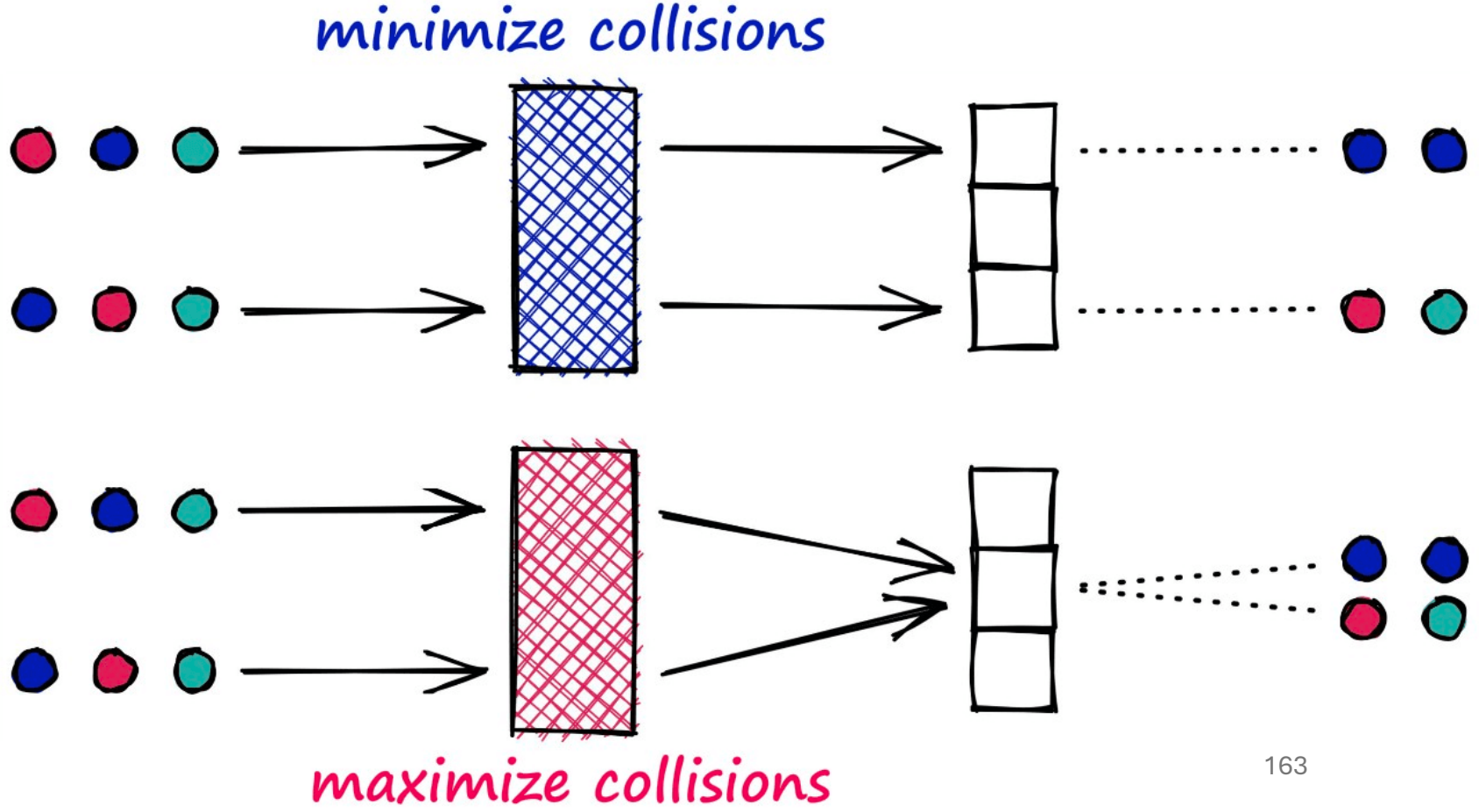
RAG - Indexing Strategies – LSH – Locality sensitive hashing (LSH)

An LSH function aims to place similar values into the same buckets.



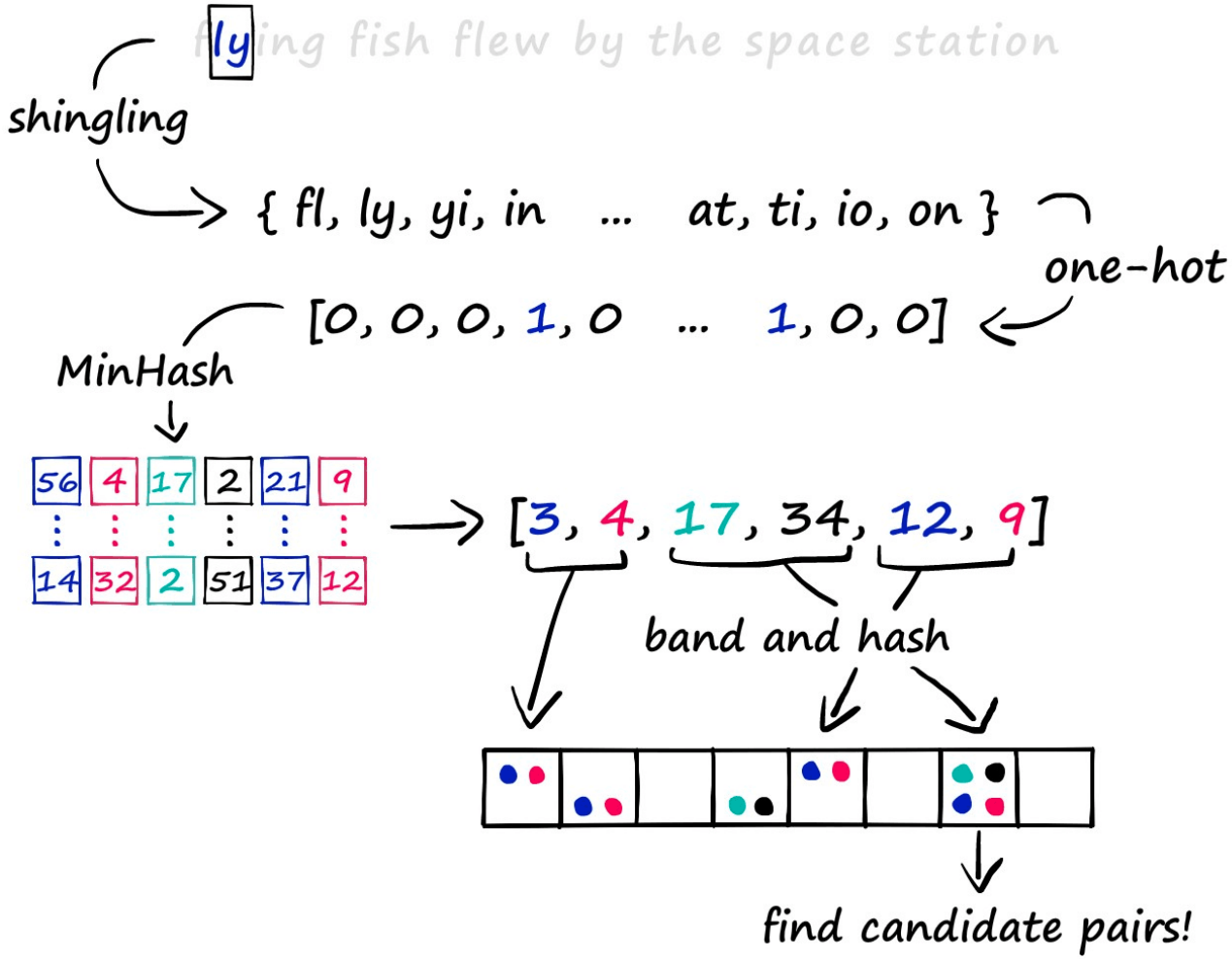
RAG - Indexing Strategies – LSH – Locality sensitive hashing (LSH)

- Typical Hashing - the top (blue) minimizes hashing collisions.
- LSH - The bottom (magenta) maximizes hashing collisions
  - aims to maximize collisions between similar items.



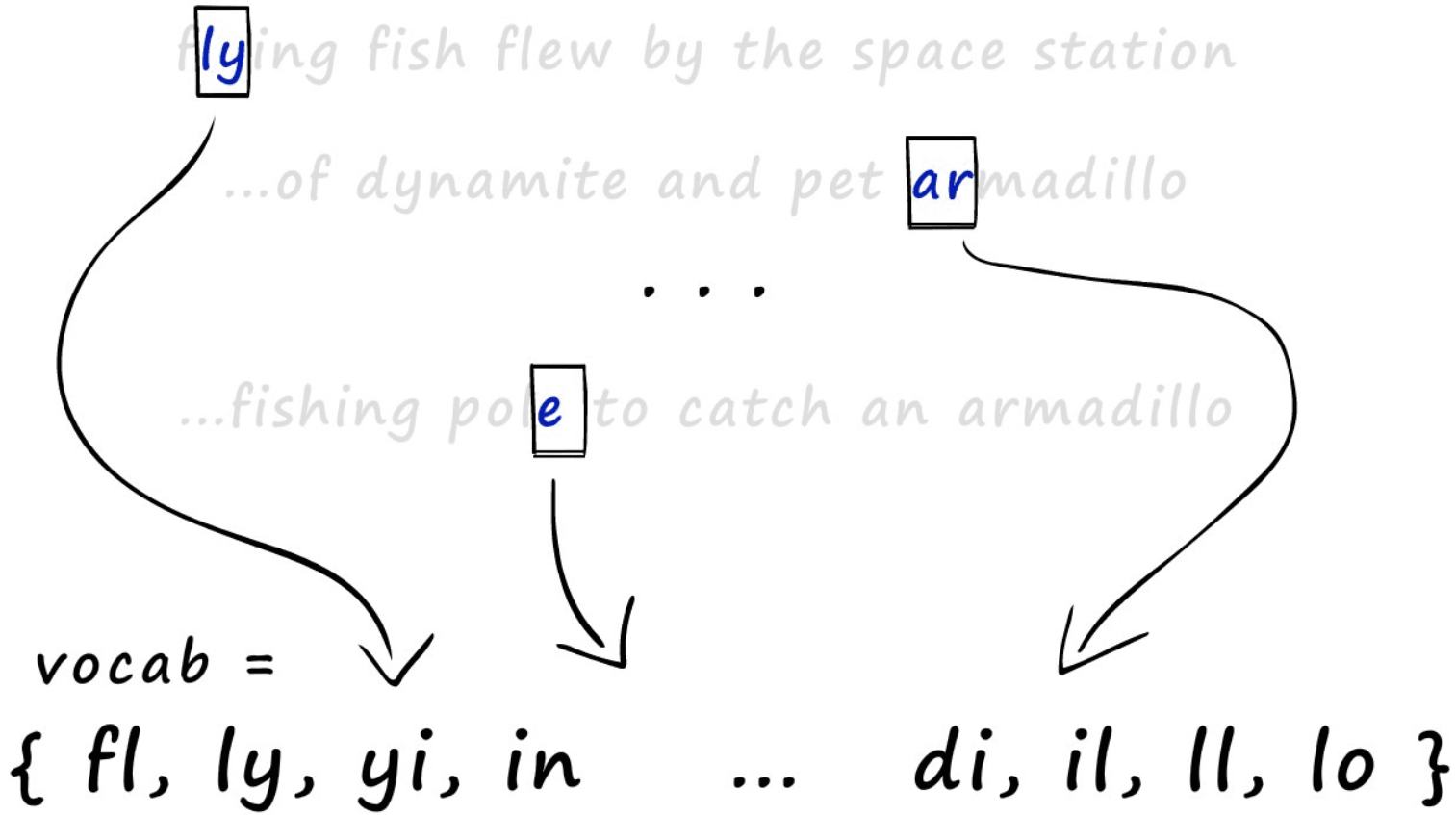
RAG - Indexing Strategies – LSH – LHS Process: Shingling, MinHashing, and LSH

A high-level view of the LSH process



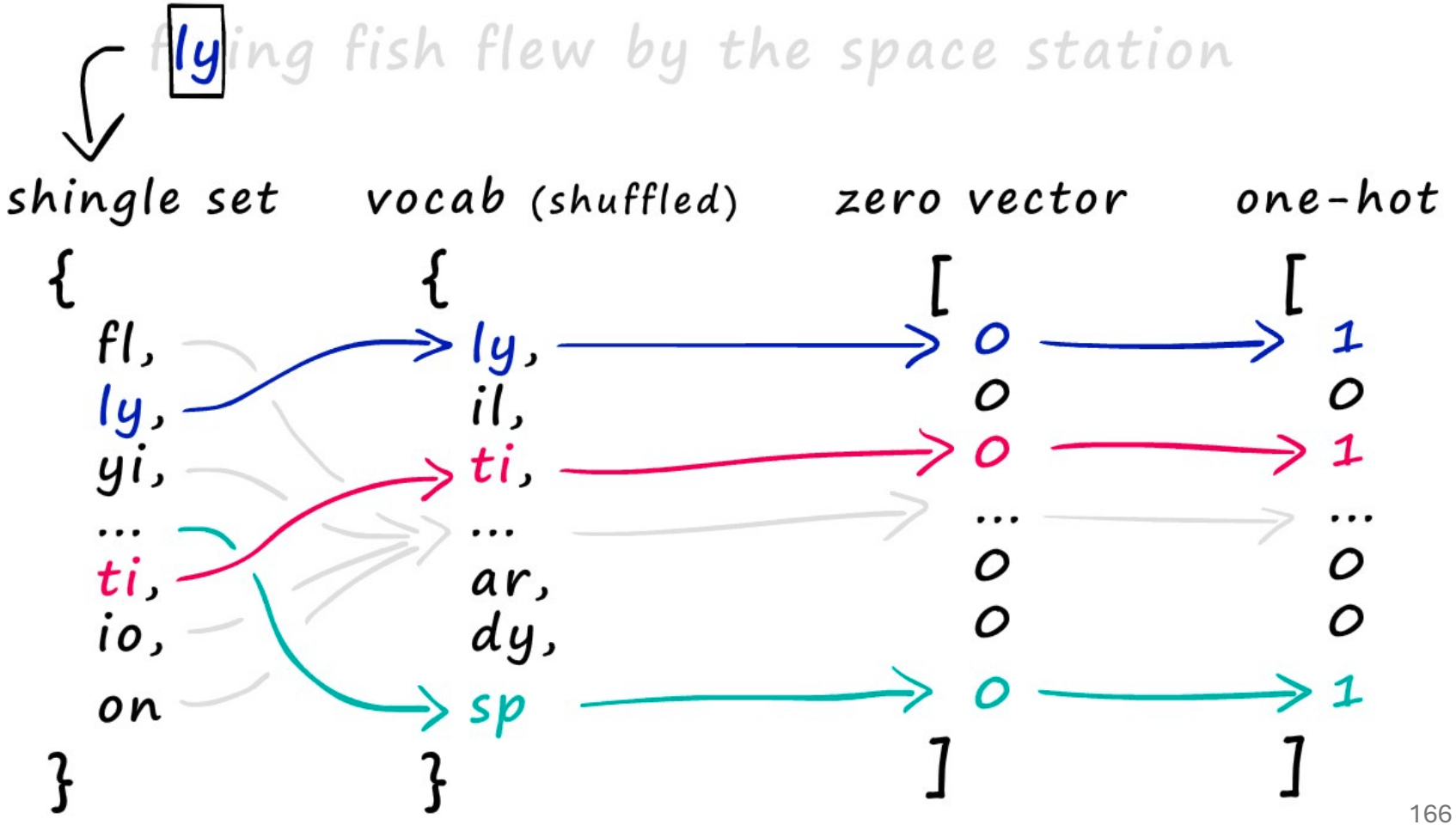
RAG - Indexing Strategies – LSH – LHS Process: (k-)Shingling

- **k-Shingling** - the process of converting a string of text into a set of ‘shingles’.
- Similar to a moving window of length k.



RAG - Indexing Strategies – LSH – LHS Process: (k-)Shingling

- Create one hot encodings (determined by vocabulary)



RAG - Indexing Strategies – LSH – LHS Process: Minhashing

- Generate (randomly) one Minhash function for every position in our signature
- For dense vectors - if we wanted to create a dense vector/signature of 20 numbers - we would use 20 minhash functions

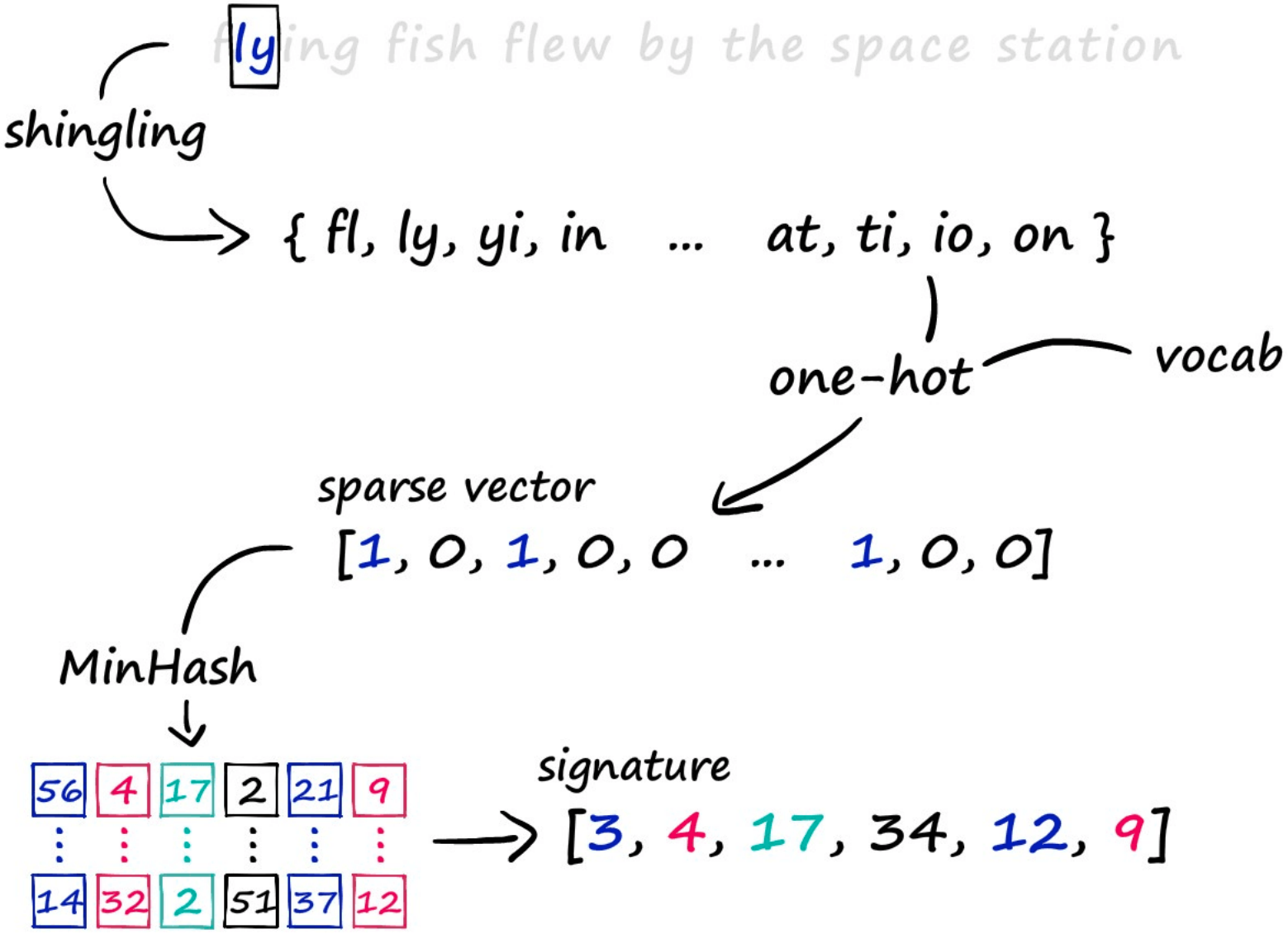
- MinHash functions are simply a randomized order of numbers - and we count from *1* to the final number (which is len(vocab)).

RAG - Indexing Strategies – LSH – LHS Process: Minhashing

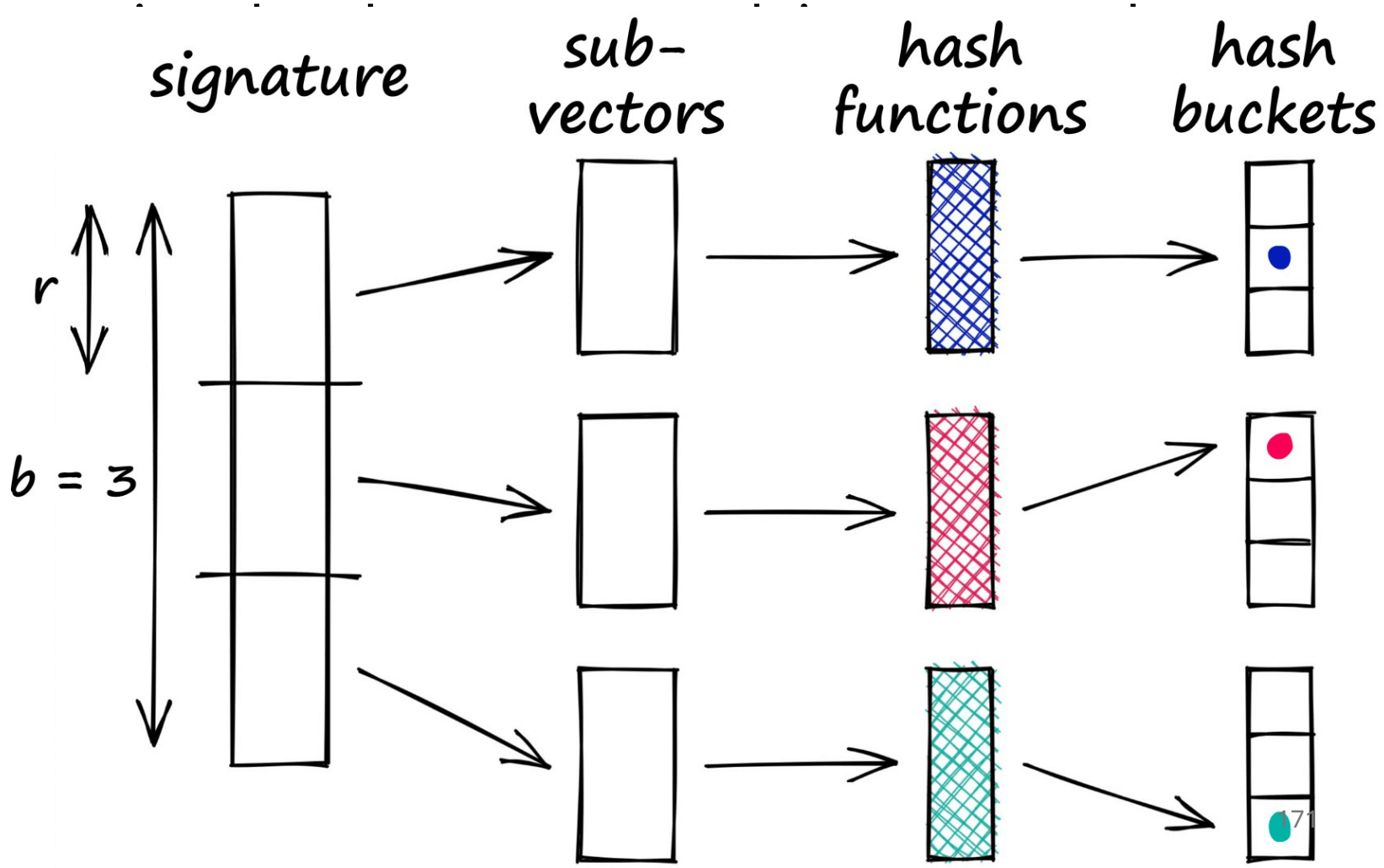
- We look at our sparse vector and say, “did this shingle at vocab[1] exist in our set?”.
- If it did — the sparse vector value will be 1 (0 otherwise).
- We return the sum of resulting 1s.

- We need to produce many (20 or more) of these values. So, we assign a different minhash function to each signature position

- A high-level v
  text, build a s
  process it thr



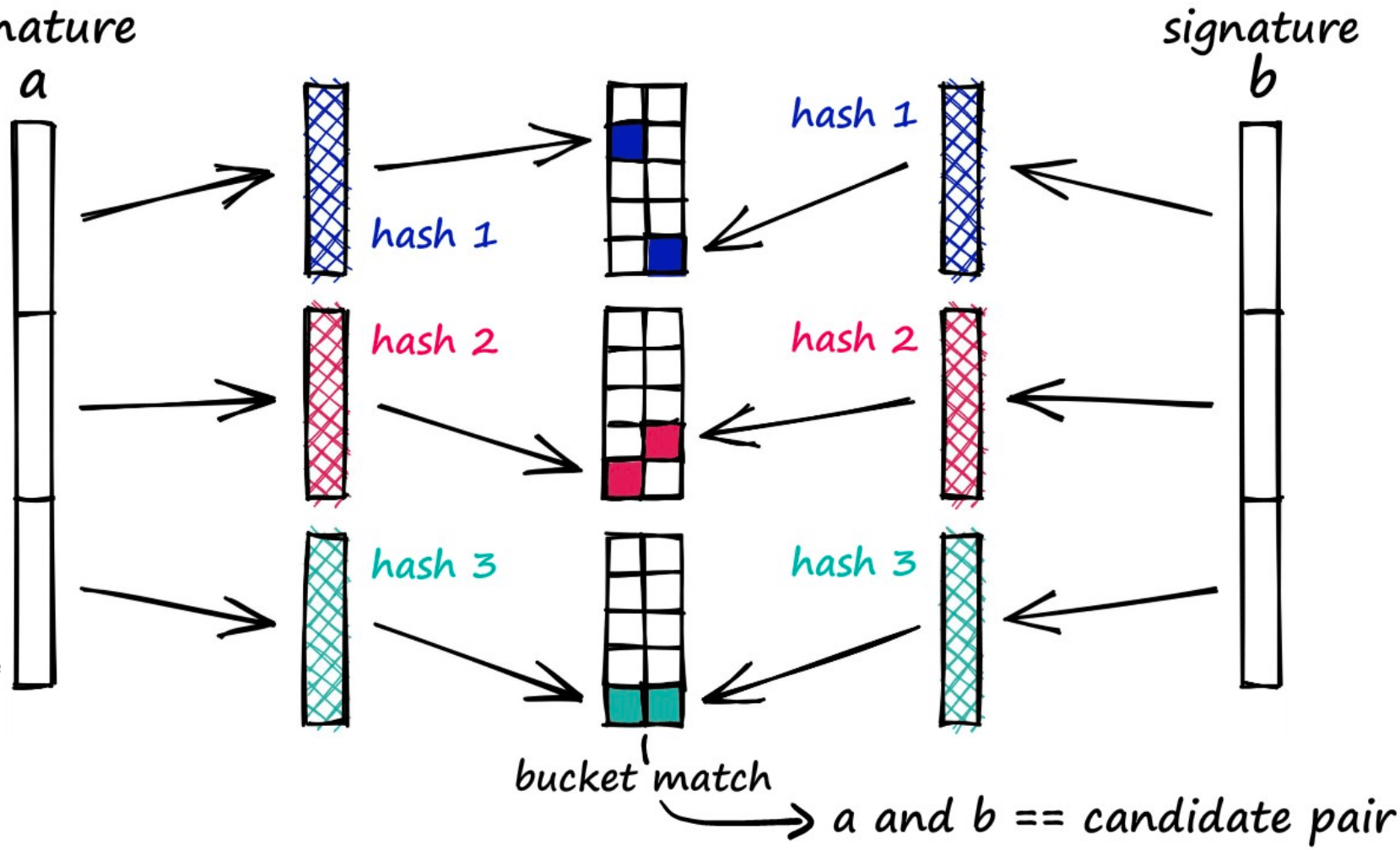
- We split our sig
through a hash
hash functions



Performing search with LSH consists of three steps:

1. Index all of our vectors into their hashed vectors.
2. Introduce our query vector (search term). It is hashed using the same LSH function.
3. Compare our hashed query vector to all other hash buckets via Hamming distance — identifying the nearest

\We split the signatures into subvectors. Each equivalent subvector across all signatures must be processed through the same hash function. However, it is not necessary to use different hash functions for each subvector (we can use just one hash function for them all).



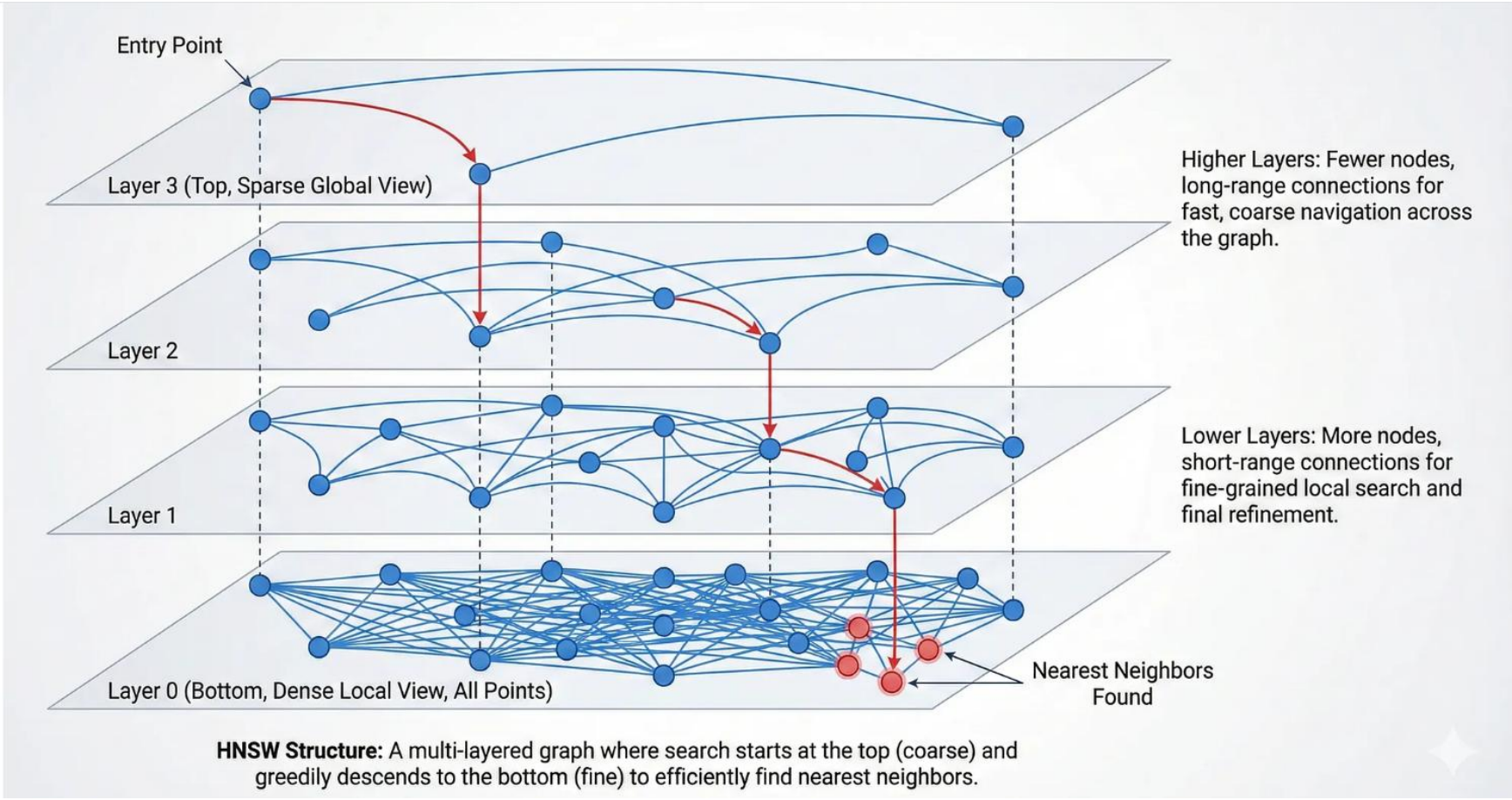
RAG – Ingestion – Chunking

HNSW (Hierarchical Navigable Small World)

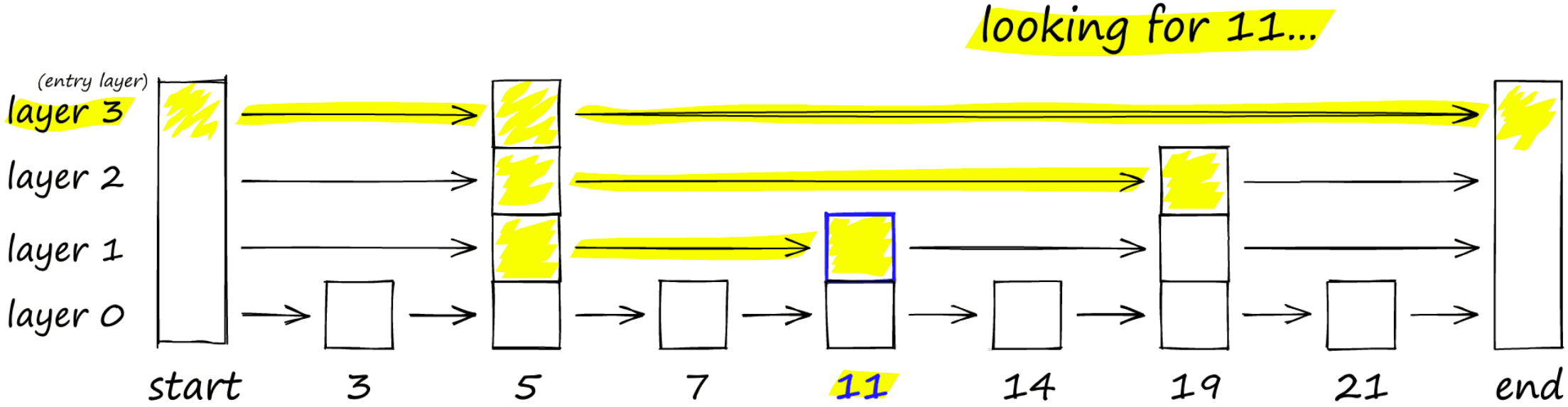
Multi-layered graph structure with great query speed, but comes with significant memory consumption.

- ❑ Fast query with 95%+ recall accuracy
- ❑ Memory: 50-100 bytes per vector
- ❑ Query time: 1-5ms for +1M datasets

Hierarchical Navigable Small World (HNSW)

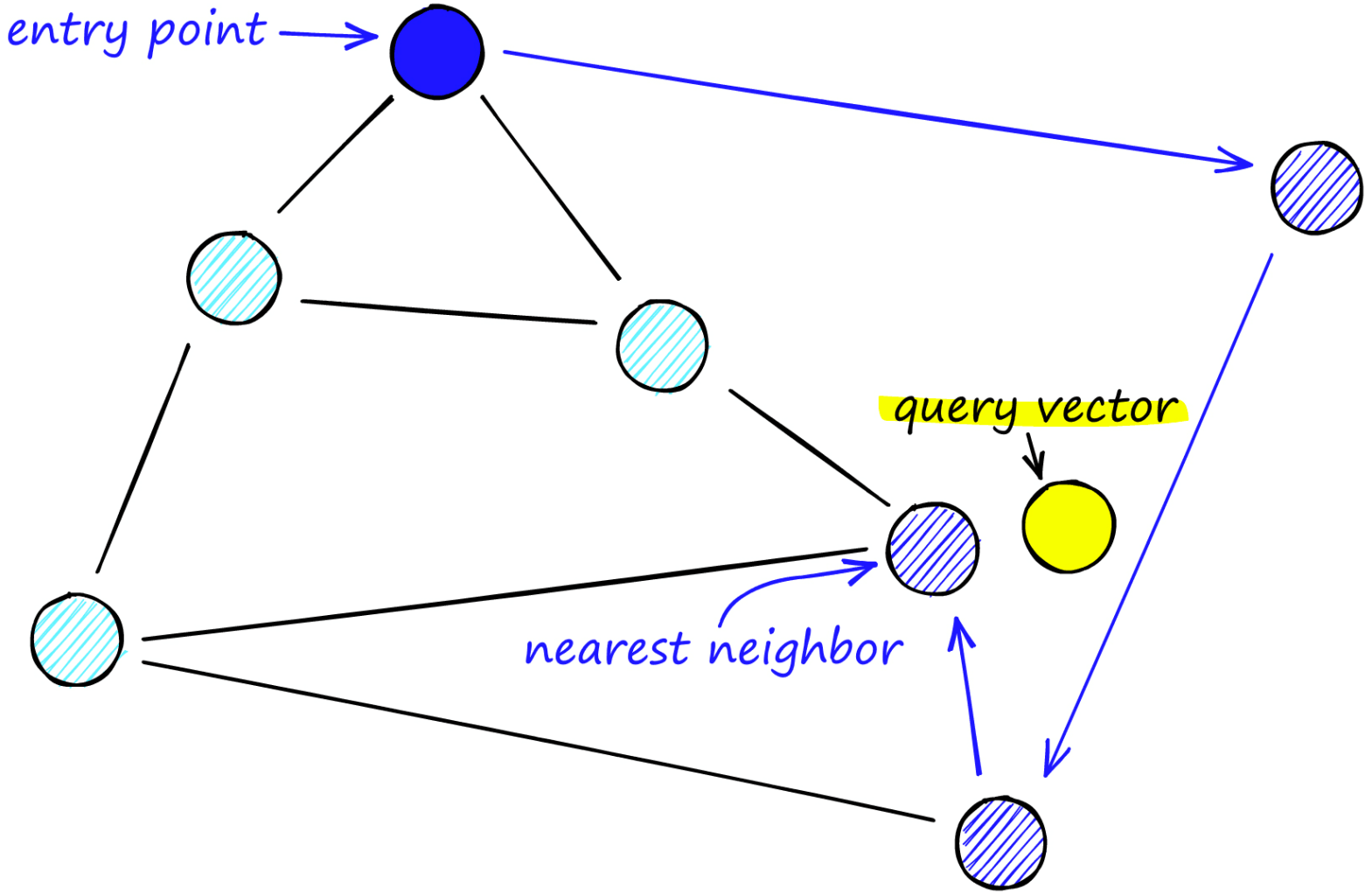


Insperation: Probabilistic Skip List



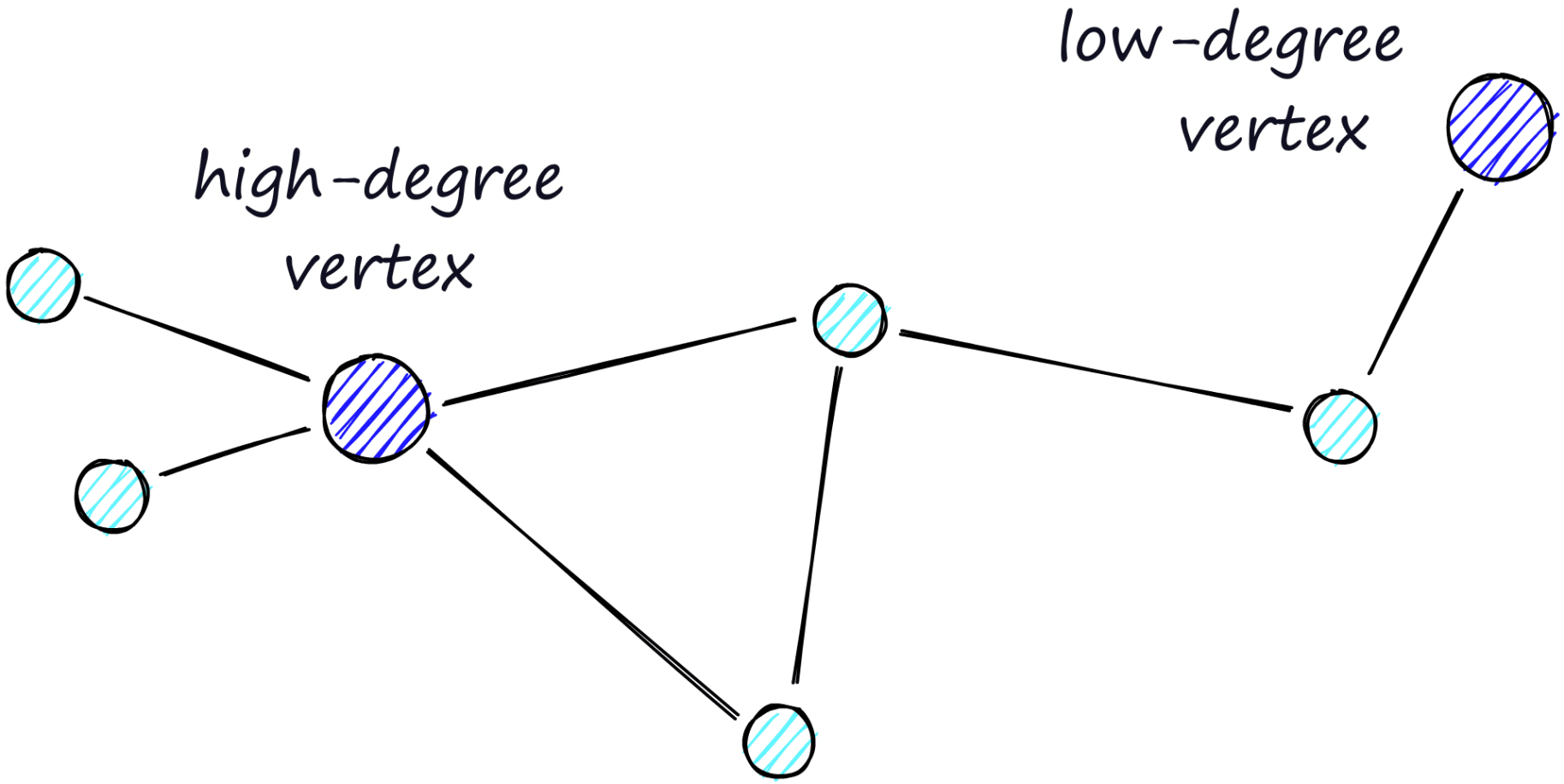
Navigable Small World Graphs

The search process through a NSW graph. Starting at a pre-defined entry point, the algorithm greedily traverses to connected vertices that are nearer to the query vector.

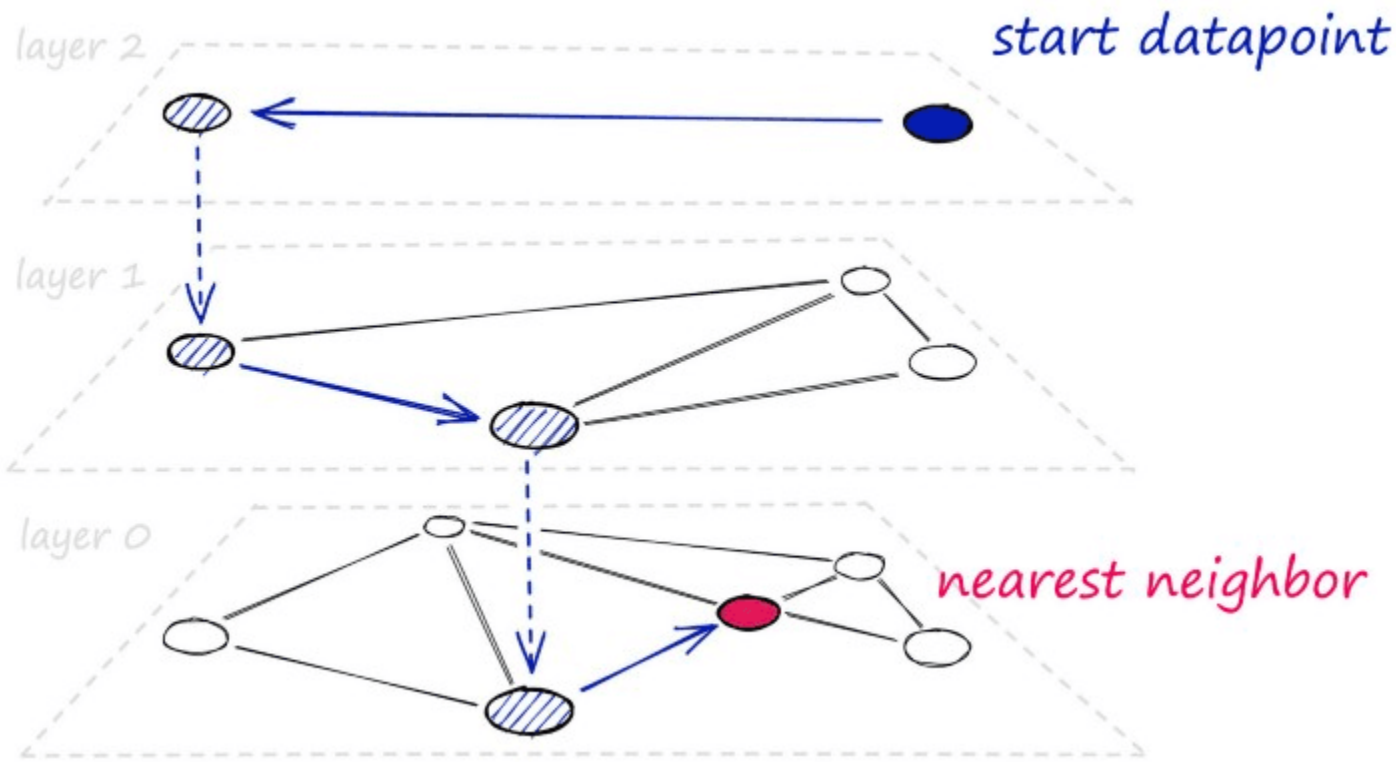


The routing

High-degree vertices have many links, whereas low-degree vertices have very few links.

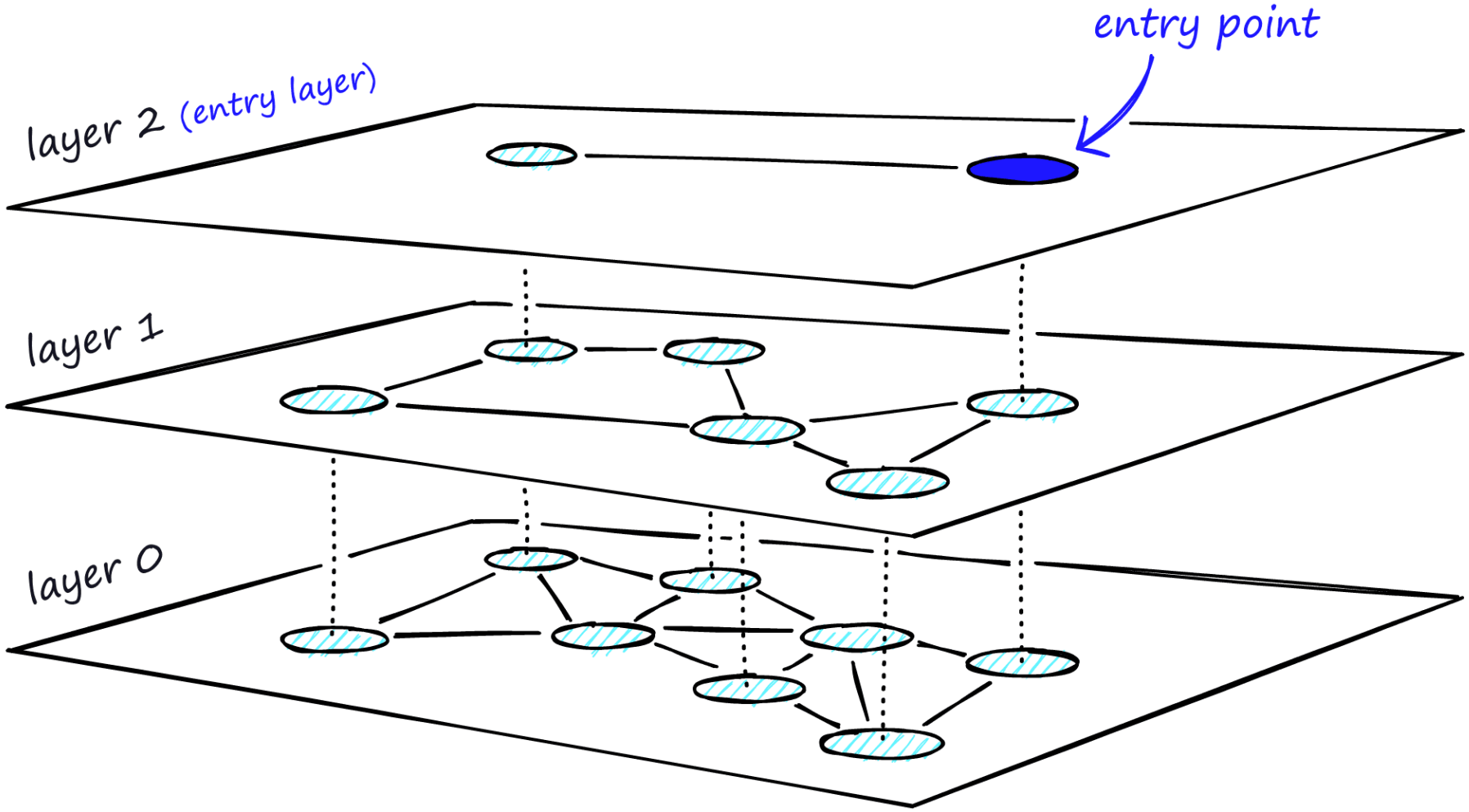


HNSW - Searching Nearest Neighbors



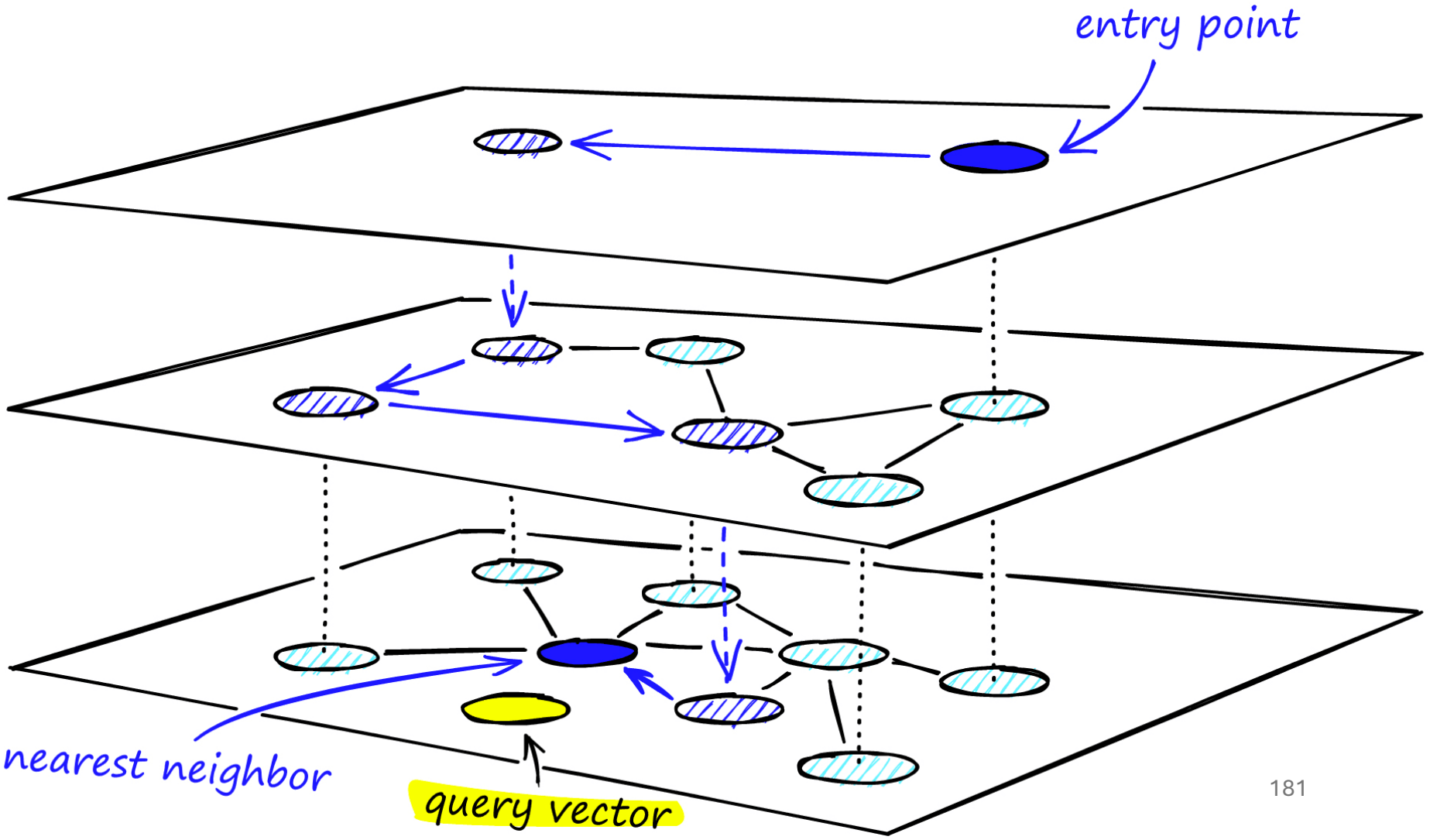
Creating HNSW

Layered graph of HNSW, the top layer is our entry point and contains only the longest links, as we move down the layers, the link lengths become shorter and more numerous.



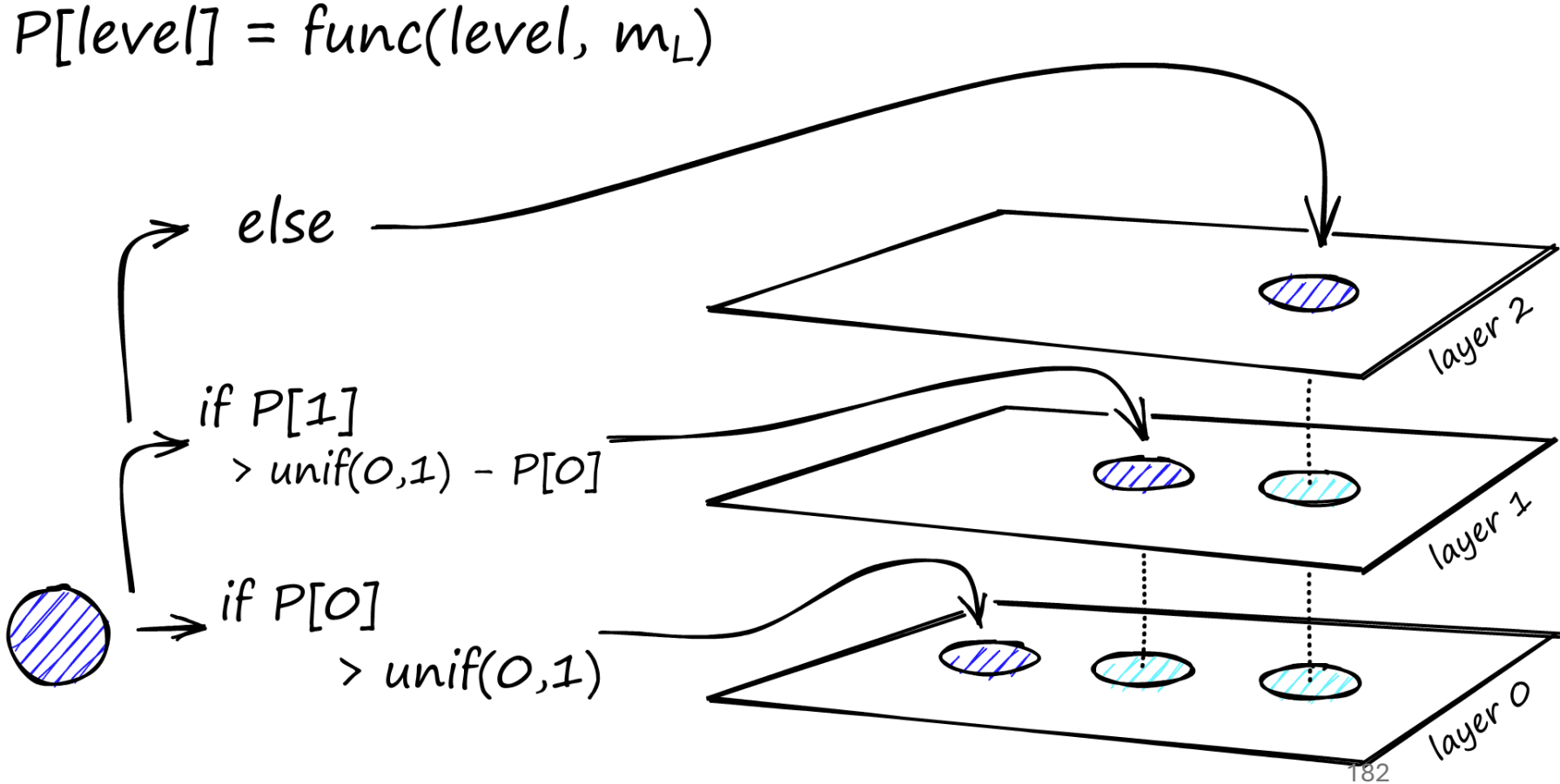
Creating HNSW

The search process through the multi-layer structure of an HNSW graph.



Graph Construction

The probability function is repeated for each layer (other than layer 0). The vector is added to its insertion layer and every layer below it.



Graph Construction

Explanation of the number of links assigned to each vertex and the effect of M, M_max, and M_max0.

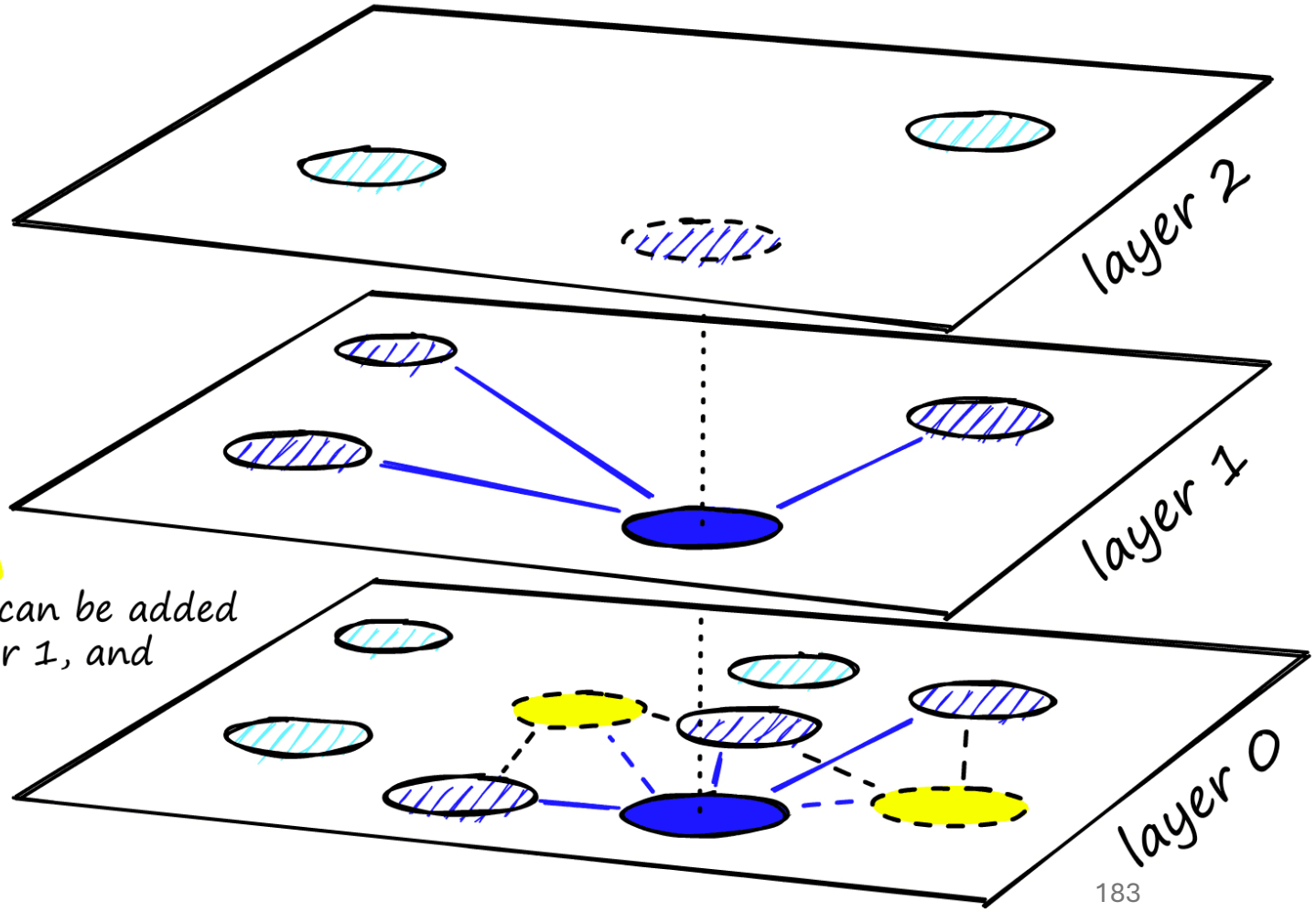
insert vector at layer 1

with M = 3 layer 1 and 0 find 3 links

as more vertices are inserted, more links can be added - up to M_{max} for layer 1, and M_{max0} for layer 0

$M_{max} = 3$

$M_{max0} = 5$



RAG – Augmentation

- create an augmented prompt with both the search results and the user's query to send to the LLM. This is where the magic happens.
- An augmented prompt might look like this:

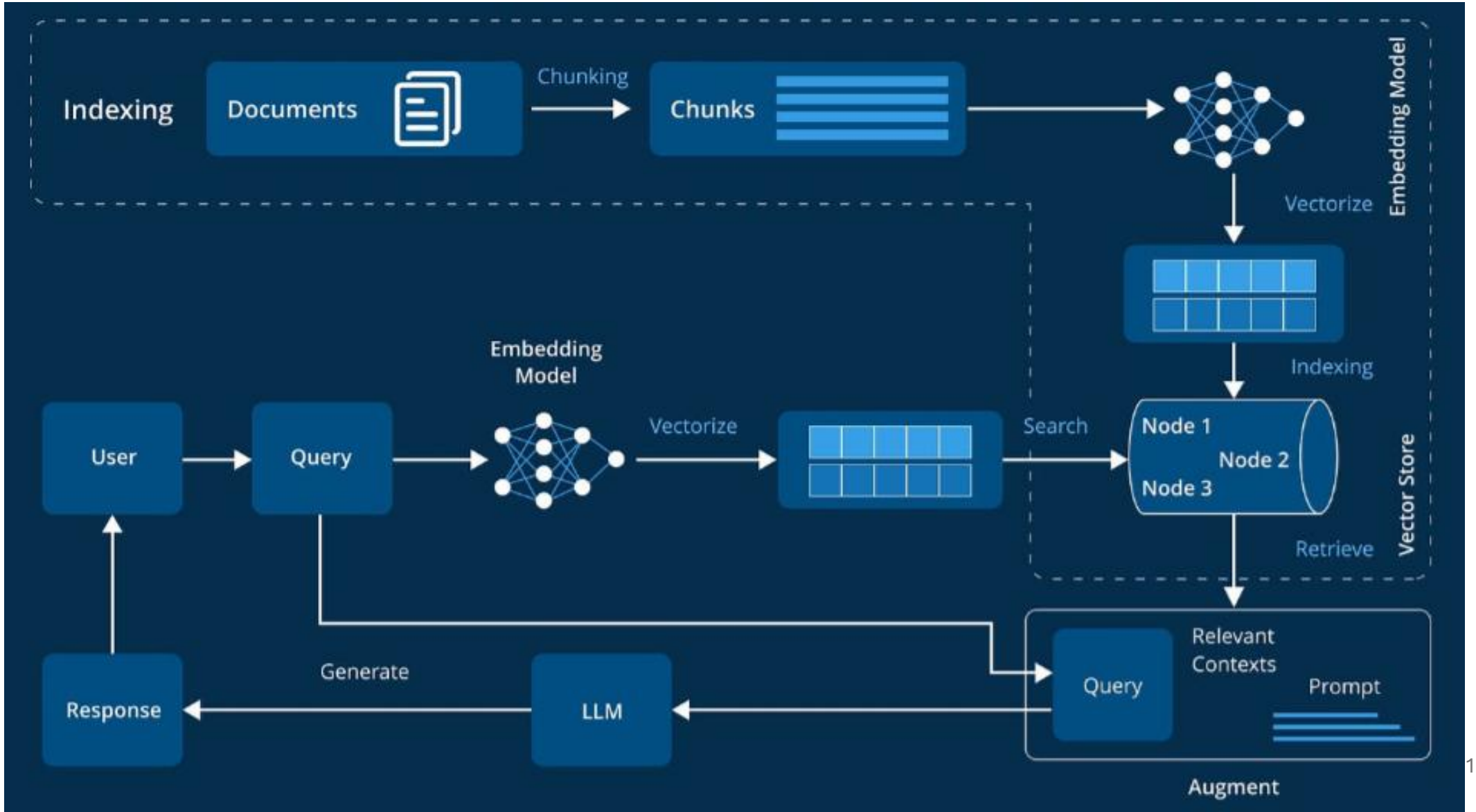
```
QUESTION:
<the user's question>

CONTEXT:
<the search results to use as context>

Using the CONTEXT provided, answer the QUESTION. Keep your answer
grounded in the facts of the CONTEXT. If the CONTEXT doesn't
contain the answer to the QUESTION, say you don't know.
```

- By sending both the search results and the user's question as context to the LLM, you are encouraging it to use the more accurate and relevant info from the search results during the next generation step.

RAG – Generation

- Using the augmented prompt, the LLM now has access to the most pertinent and grounding facts from your vector database so your application can provide an accurate answer for your user, reducing the likelihood of hallucination.

Typical RAG system



Without Query Expansion

Query: What is a bridge for teeth?

"What is a bridge?" Are you missing a tooth? A bridge is one of the dental treatments available to replace missing teeth. Bridges are fixed into the mouth by attaching them to neighboring teeth. Therefore, unlike dentures (which are removable), these stay in your mouth. The most common type is a conventional bridge where the teeth on either-side of the gap are crowned, with a "false" tooth stuck to those crowns, in order to fill the gap.

With Query Expansion

Query: What is a bridge for teeth?

"What is a bridge?" Are you missing a tooth? A bridge is one of the dental treatments available to replace missing teeth. Bridges are fixed into the mouth by attaching them to neighboring teeth. Therefore, unlike dentures (which are removable), these stay in your mouth. The most common type is a conventional bridge where the teeth on either-side of the gap are crowned, with a "false" tooth stuck to those crowns, in order to fill the gap.

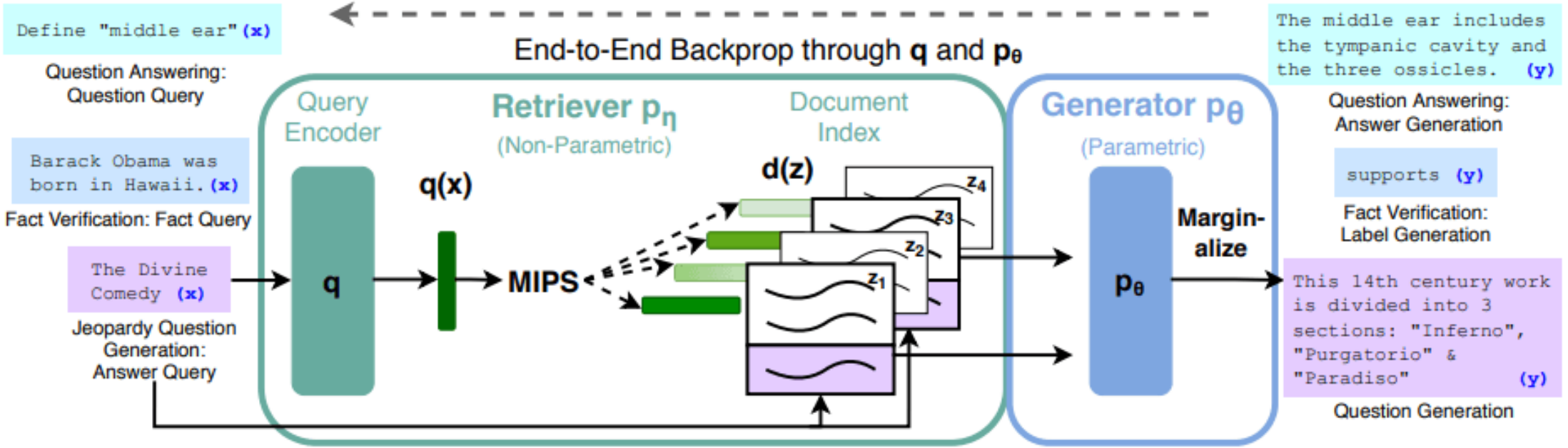
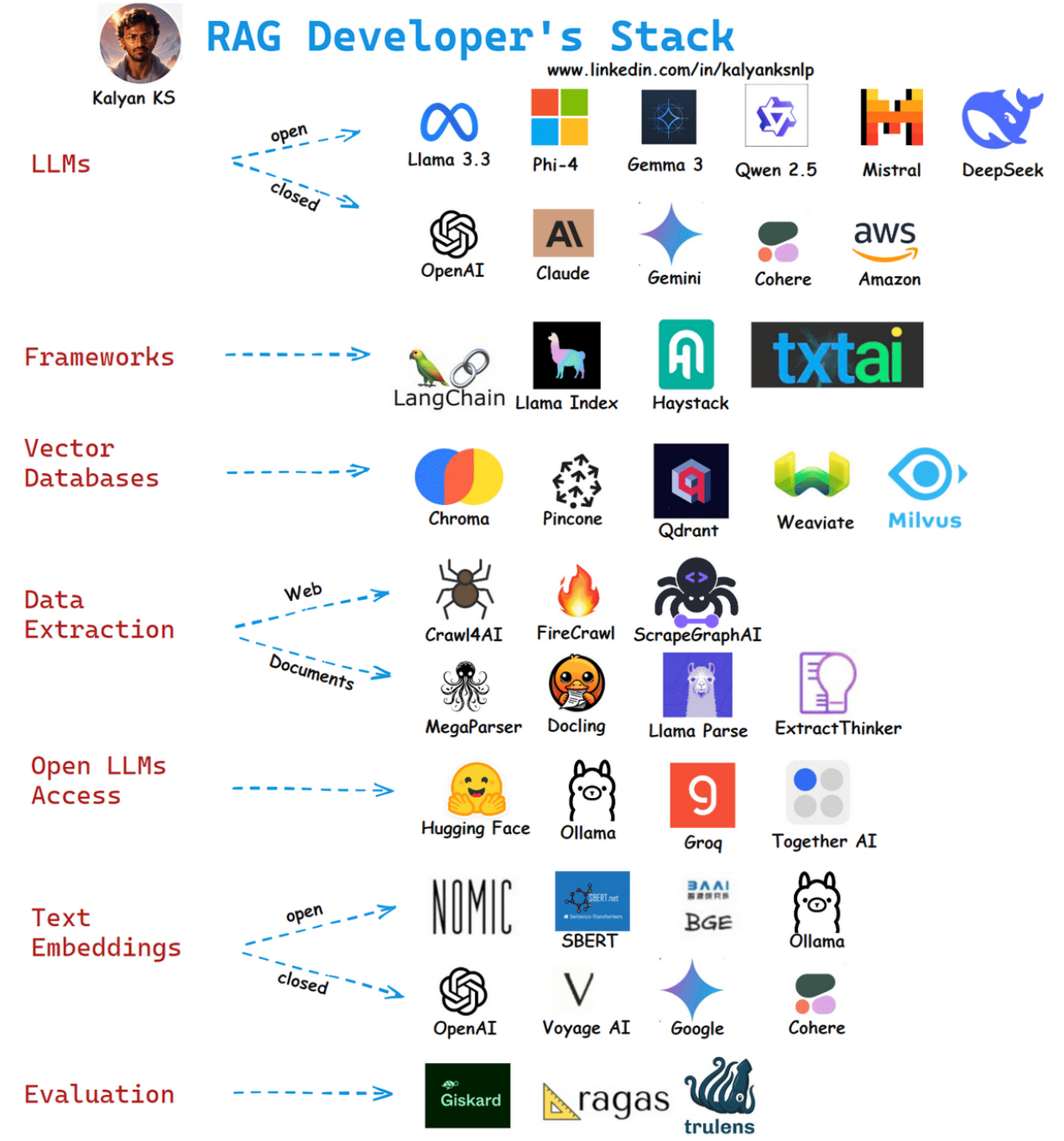


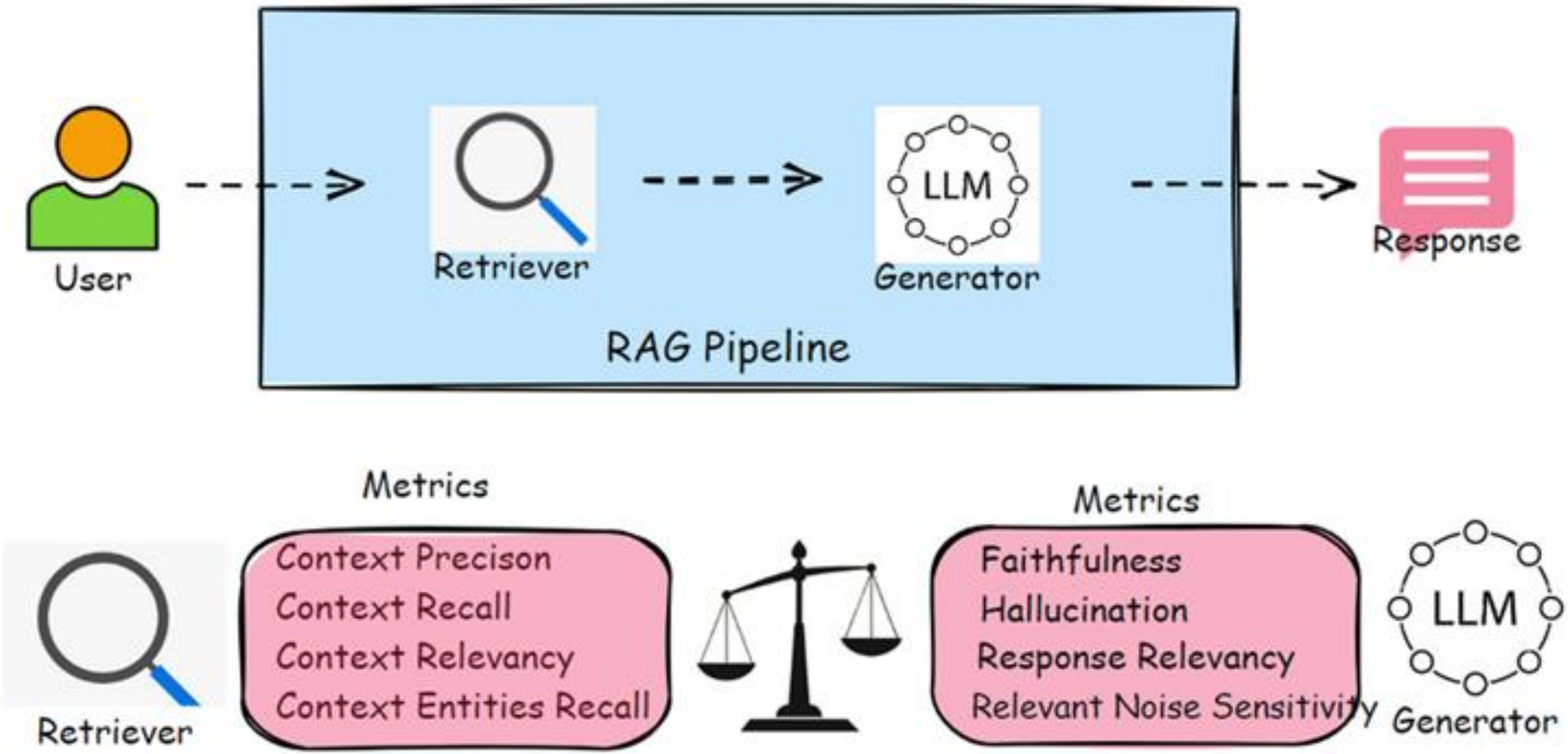
Figure 1: Overview of our approach. We combine a pre-trained retriever (*Query Encoder* + *Document Index*) with a pre-trained seq2seq model (*Generator*) and fine-tune end-to-end. For query x, we use Maximum Inner Product Search (MIPS) to find the top-K documents z_i. For final prediction y, we treat z as a latent variable and marginalize over seq2seq predictions given different documents.

RAG Components – Developer's Stack



github.com/KalyanKS-NLP/rag-zero-to-hero-guide

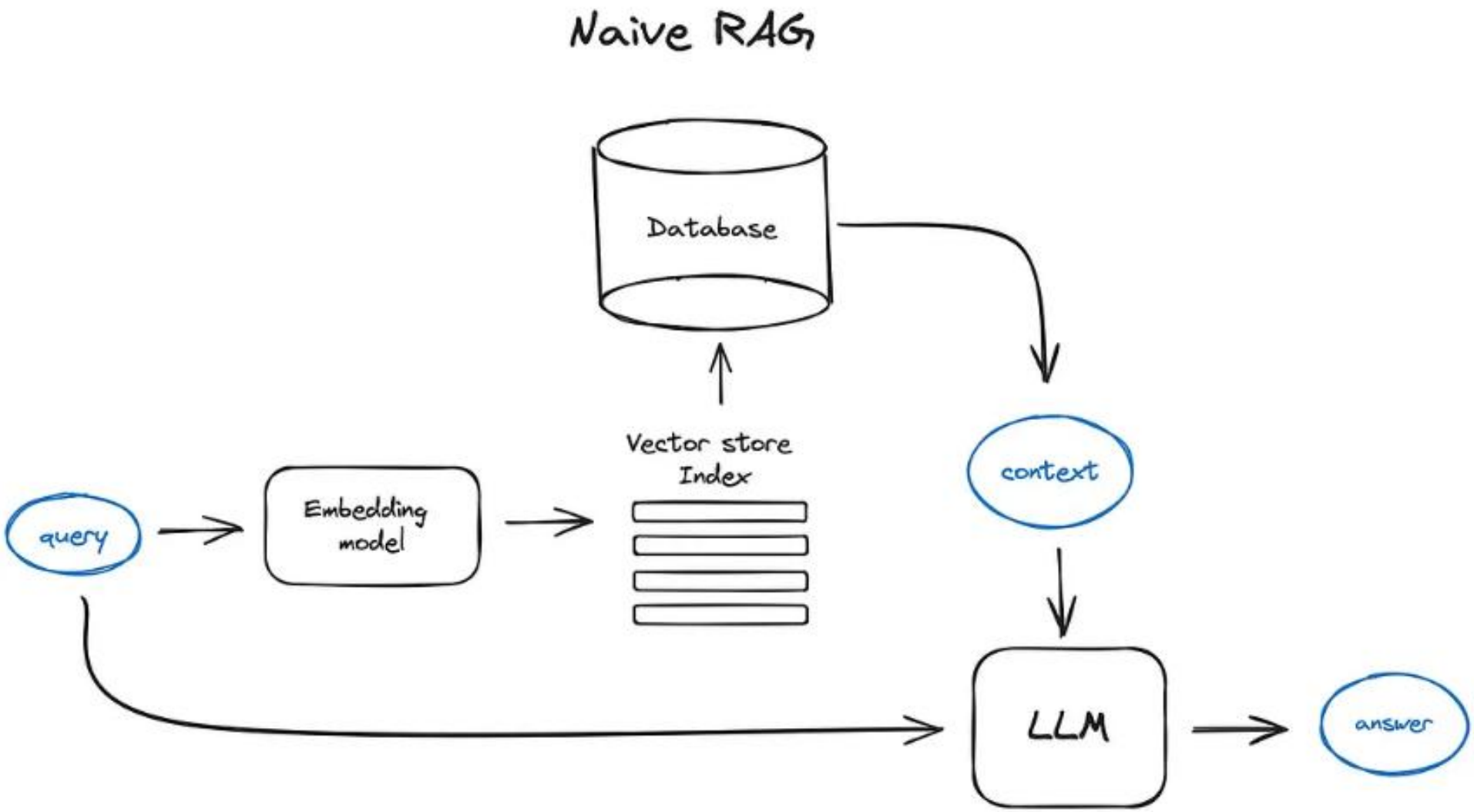
Evaluation Metrics



github.com/KalyanKS-NLP/rag-zero-to-hero-guide

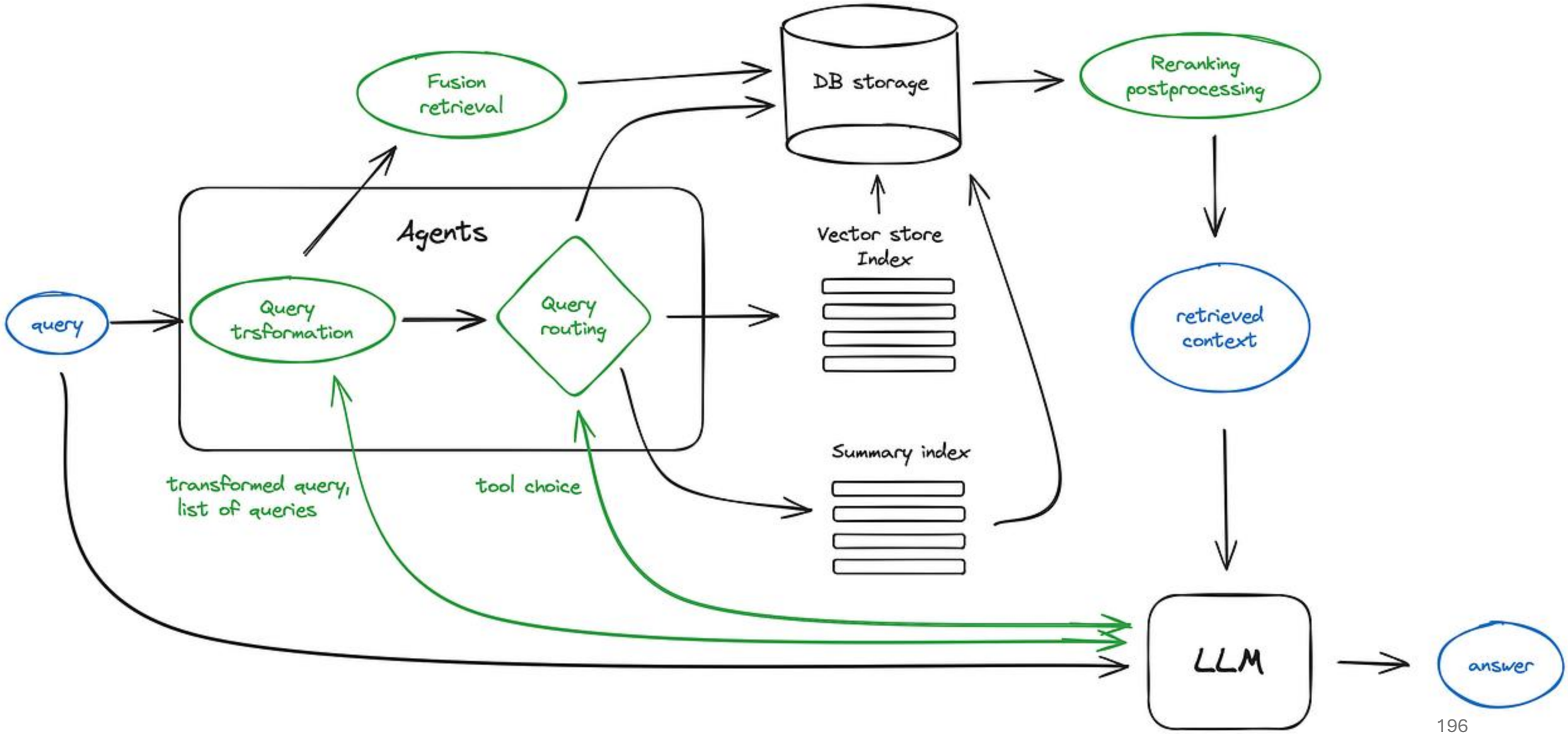
Agentic RAG

- **Differentiator:** Uses autonomous agents to plan, choose tools (vector DBs, web/APIs, SQL), and iteratively refine retrieval steps, turning a static pipeline into an adaptive reasoning loop.
- **When to use:** For queries that may need tool choice and escalation, such as “Summarize current pricing for Vendor Z and verify with their API if the document set lacks 2025 data.”
- **Costs/trade-offs:** Multi-step planning/tool calls add latency and token spend, and orchestration complexity raises observability and failure-handling burdens.

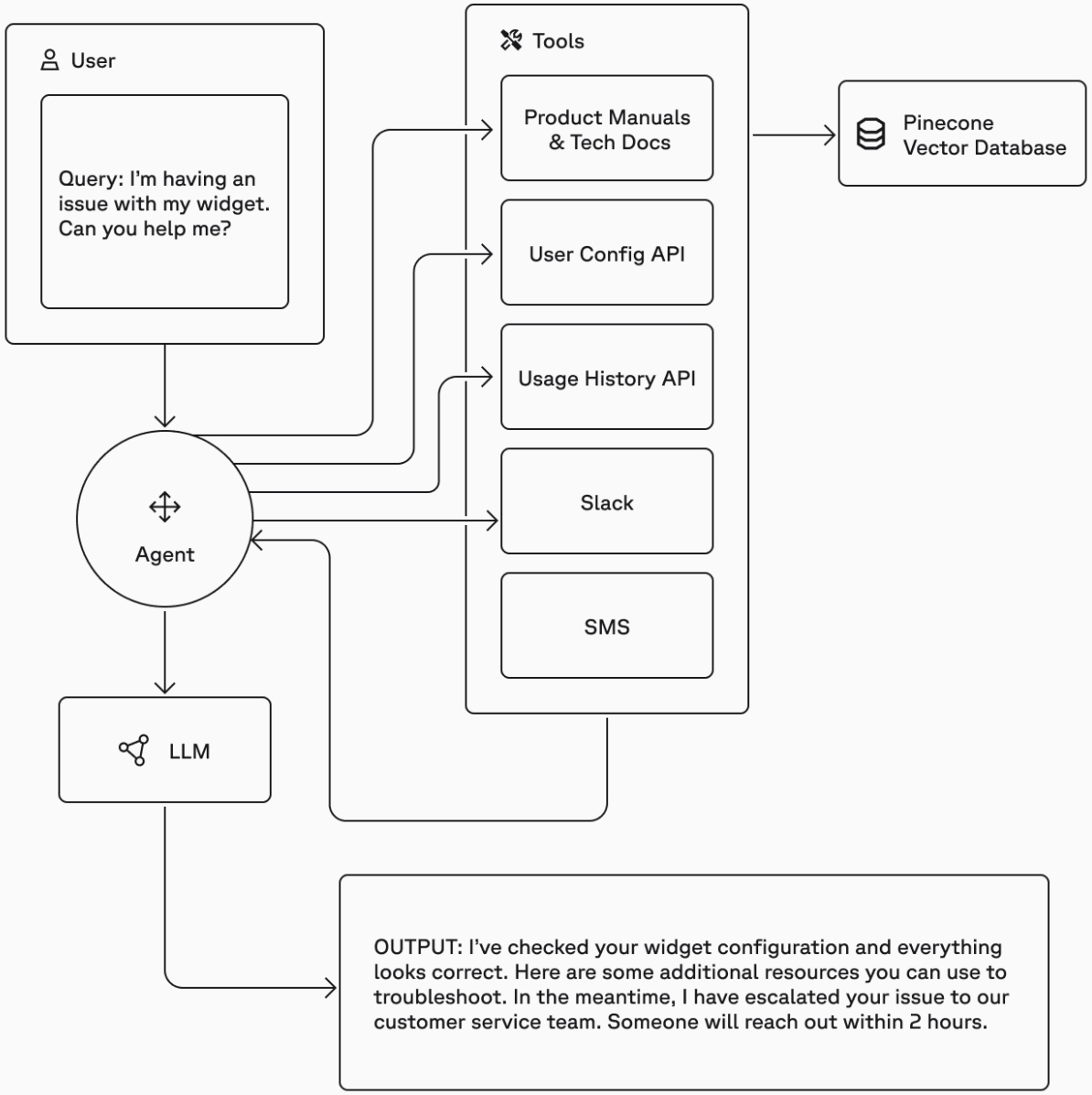


A scheme by author, as well all the schemes further in the text

Advanced RAG



Agentic RAG


User
Query: I'm having an issue with my widget. Can you help me?
Tools
Product Manuals & Tech Docs
Pinecone Vector Database
User Config API
Usage History API
Slack
SMS
Agent
LLM
OUTPUT: I've checked your widget configuration and everything looks correct. Here are some additional resources you can use to troubleshoot. In the meantime, I have escalated your issue to our customer service team. Someone will reach out within 2 hours.

Feature	Traditional RAG	Agentic RAG
Data Processing	Linear, retrieves data from specific sources	Multi-source, refines and adapts data in real-time
Decision-Making	Passive, follows user-guided queries	Autonomous, adjusts responses based on real-time analysis
Complexity Handling	Suitable for simple or direct queries	Handles multi-step reasoning and dynamic queries
Application Examples	Customer support, FAQ bots	Healthcare diagnostics, legal research, financial analytics

Until the next time ☺