INF 385T · FALL 2026

TEXAS
The University of Texas at Austin

INF385T: Special Topics in Information Science: Language Models and Agentic Workflows

Week 3: Context Engineering: Retrieval, Memory, and State

Dr. Abhijit Mishra

Announcement

- Frist Assignment due tonight
- Second assignment posted:
  - to build a small evidence-grounded question-answering workflow over an approved document collection.
- Project group formation
  - https://utexas.instructure.com/courses/1454968/assignments/7800078
  - Form here: https://forms.gle/gmB16NsJDUZnGhxi6
- Deadline: Next Thursday (09/10)

Week2 : Recap

- Transformers, LLMs, In context learning recap
- Structured outputs and schemas
- Function calling and tool use interfaces
- Model limitations and failure modes
- Tool descriptions as interface contracts

Recap: What are Language Models?

- Models that estimate the joint probability distribution of words

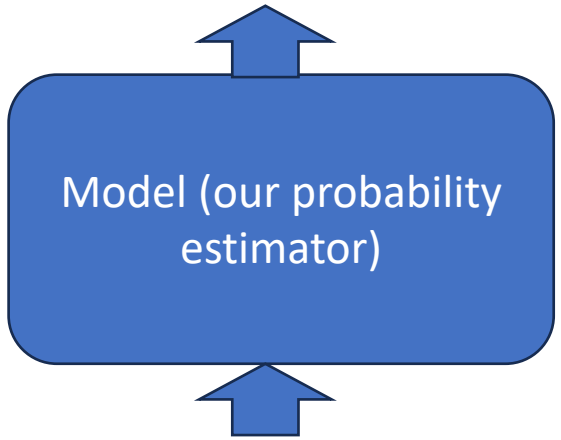
$$W = (w_1, w_2, w_3, ..., w_n)$$

$$P(W|\theta) = P(w_1|\theta) \cdot P(w_2|w_1, \theta) \cdot P(w_3|w_1, w_2, \theta) \cdot \ldots \cdot P(w_n|w_1, w_2, ..., w_{n-1}, \theta)$$

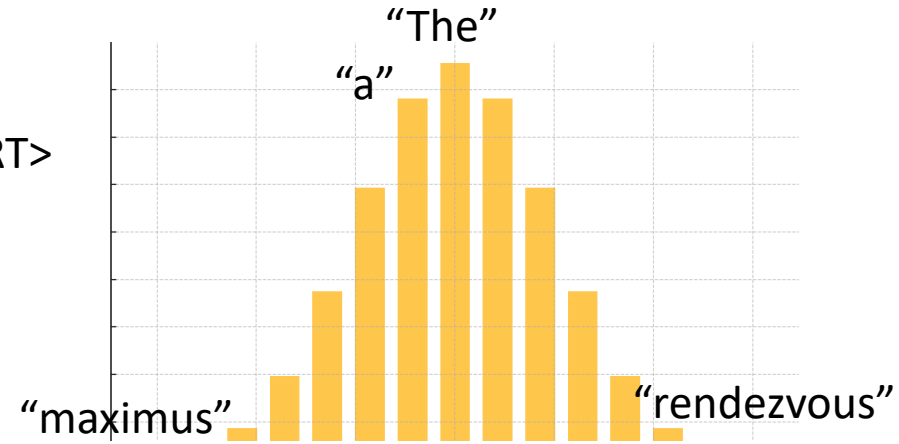
Model parameters

Recap: Word prediction = a sequenctial classification problem

Estimates the probability of every word given <START>



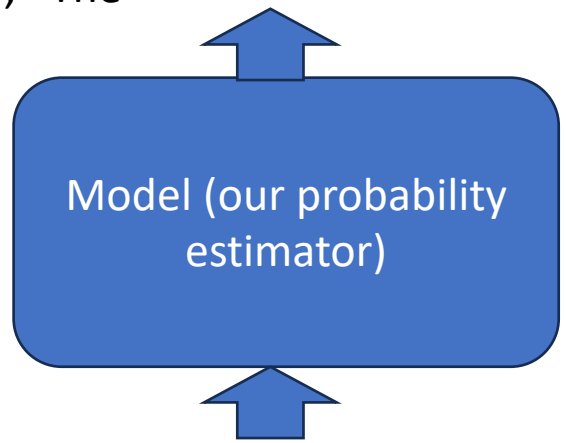
<START>



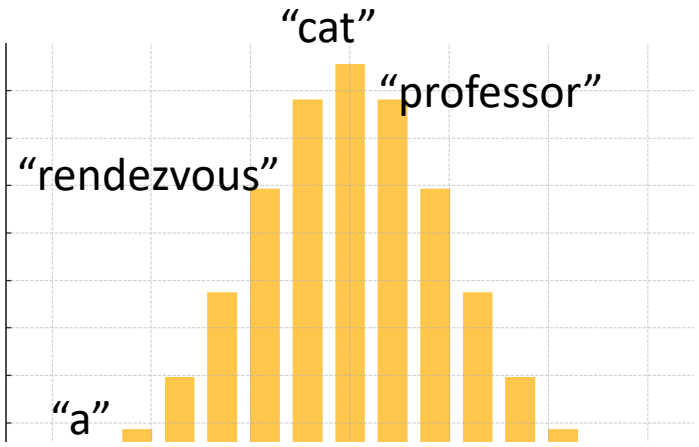
Picks “the” as the highest probable outcome

Predictions are auto-regressively done

Estimates the probability of every word given <START>, “The”



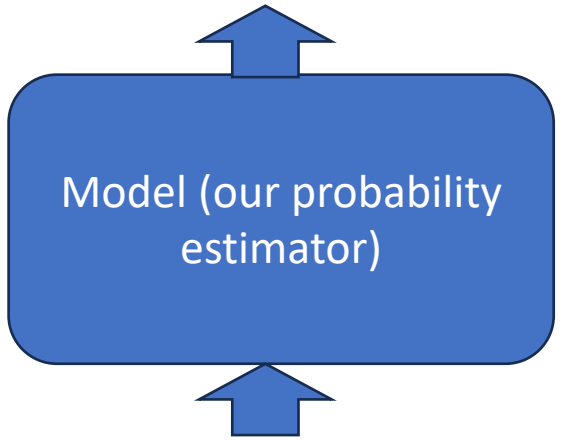
<START>, “The”



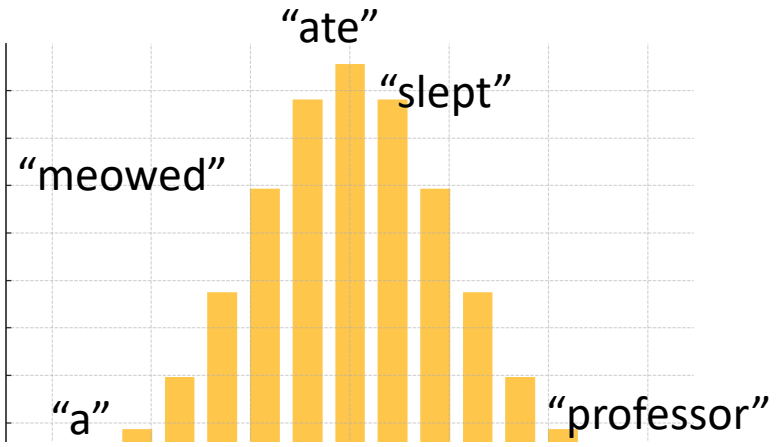
Notice how article “a” becomes least probable given the context

Predictions are auto-regressively done (…)

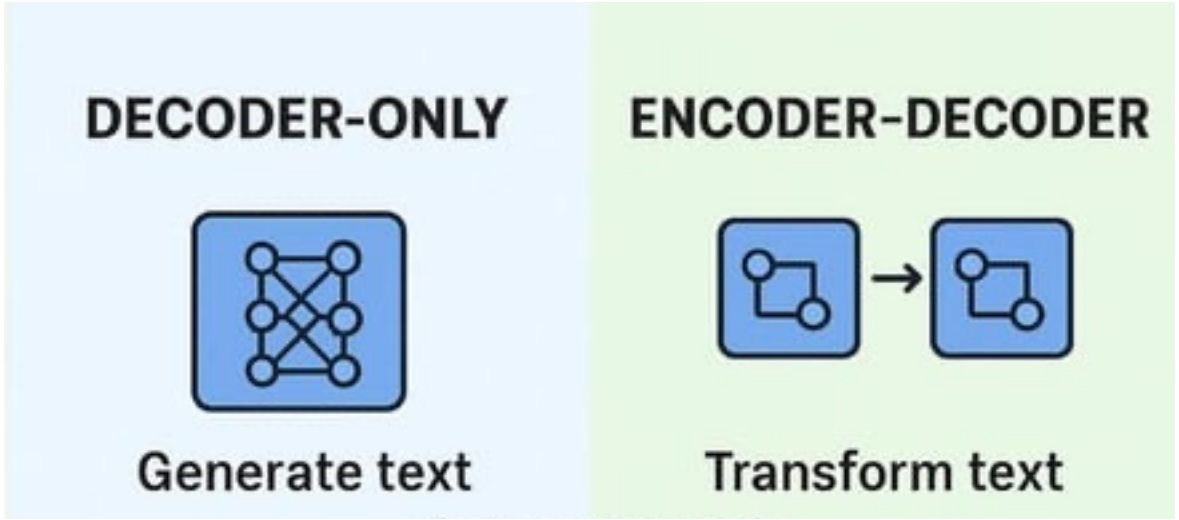
Estimates the probability of every word given <START>, “The”, cat”



<START>, “The”, “cat”



Recap: Different modern probability estimation models

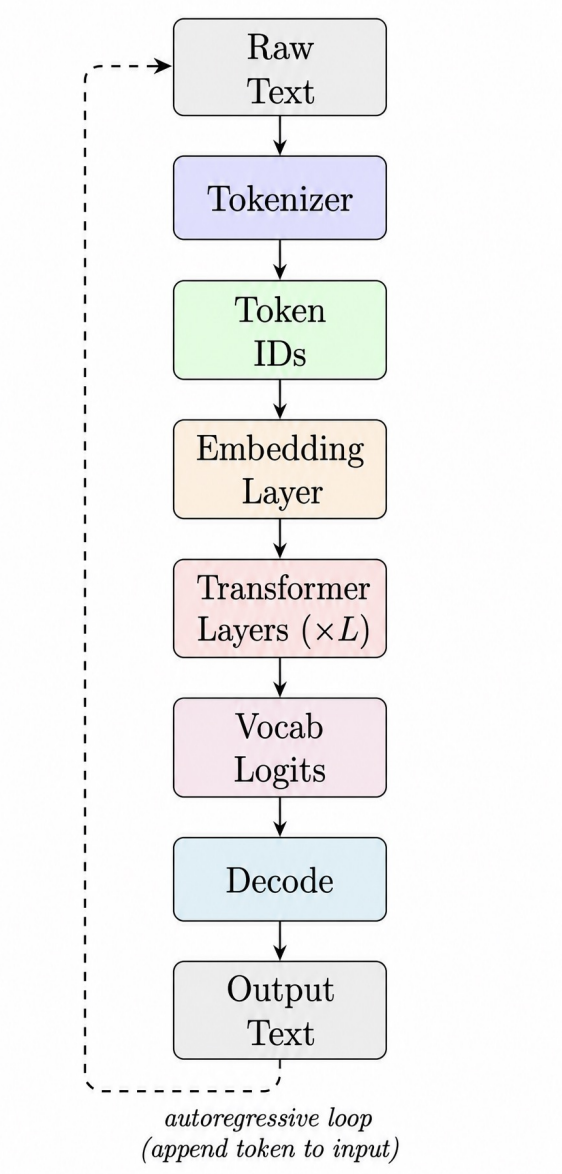


Example: **GPT**
(Next word prediction)

Example: **BART/T5**
(Denoised input-output)

Recap: What are LLMs?

- Language models made larger by introducing many transformer layers
- Trained on massive web-scale data and fine tuned on human and agentic tasks
  - E.g., train on books and fine tune to produce JSON / Structured outcomes



- **Tokenization:** Raw text is divided into subword units, not individual characters or necessarily complete words, using a learned vocabulary.
  - For example, “unhappiness” might be split into [“un”, “happiness”] or [“unhapp”, “iness”].
- **Embedding:** Each token ID is used to retrieve a learned dense vector from an embedding table. The vectors represent semantic information, so tokens with similar meanings often have similar representations.
- **Contextual Processing:** The transformer layers process the embeddings using self-attention, allowing each position to gather information from the permitted surrounding positions.
- **Prediction:** The final hidden representation is projected into logits over the complete vocabulary. These logits are converted into a probability distribution, and a decoding method selects the next token.

A typical LLM inference snapshot

Recap: Structured outputs and schemas

A schema turns prose into an interface

Example:

```
EventResult:
    event_name: text                    # required
    status: found | not_found |
            needs_clarification         # fixed choices
    location: text                      # optional
    date: text                          # required
    category: workshop | talk | social  # fixed choices
```

```
Define EventResult schema

Repeat up to 3 times:
    response = ask model for EventResult

    Parse response as JSON
    Validate response against EventResult

    If valid:
        If status = needs_clarification:
            ask user
        Else:
            accept result
        Stop

    If invalid:
        log output + validation error

After 3 failures:
    surface the error
```

```python
from typing import Literal
from openai import OpenAI
from pydantic import BaseModel, ValidationError

class EventResult(BaseModel):
    event_name: str
    status: Literal["found", "not_found", "needs_clarification"]
    location: str | None = None
    date: str
    category: Literal["workshop", "talk", "social"]

def validate(...):
  ...

client = OpenAI()

for attempt in range(3):
    response = client.responses.create(
        model="gpt-5-nano",
        instructions="Extract event information. Use YYYY-MM-DD dates.",
        input="The AI workshop is in Austin on September 18, 2026.",
        text={"format": {
            "type": "json_schema",
            "name": "event_result",
            "schema": EventResult.model_json_schema(),
            "strict": True,
        }},
    )

    try:
        event = validate(response.output_text, EventResult)
        break
    except ValidationError as error:
        log(response.output_text, error)
```

Recap: Function calling and tool-use interfaces

- A **tool** is an application-controlled capability that an LLM can request to retrieve information, perform computation, or take an action.
- Essential Components
  - **Name**: unique and action-oriented.
  - **Purpose**: when the model should select it.
  - **Input schema**: typed fields and constraints.
  - **Output structure**: stable data for the next step.
  - **Error behavior**: explicit codes and recoverable guidance.

Week 3: Roadmap

- Context Engineering: Retrieval, Memory, and State
  - Emphasize distinctions among retrieval, memory, state, and context
  - Context vs Prompt engineering
  - Retrieval augmented generation basics

The information problem

- A model can produce fluent text without possessing current or task-specific evidence
- Agentic systems need controlled ways to **acquire and preserve** information

Central question: what should enter the model context, when, and with what provenance?

Travel Planning Agent Example

"Plan my family's summer vacation to Italy."

The LLM itself has **none of the evidence required** to make that decision. Different information has to enter context through different mechanisms:

Q: What kind of information is needed?

Q: Is this a one shot question answering problem? Why or why not?

What's needed?

Retrieval

Finding relevant information in a large corpus

Thousands of travel guides, reviews, blogs, descriptions, and recommendations

Agent doesn't know beforehand which documents contain the useful information. It has to find them by meaning.

State

Remembering where we are in this particular task

Rome: 3 nights ✓
Florence: rejected — **too much moving around**
Puglia: currently considering
Budget allocated so far: $3,200 of $6,000
Still unresolved: final 4 nights

That's state. Without it, each new model call risks forgetting decisions, repeating suggestions, or exceeding the budget.

Memory

Carrying useful information across tasks

From previous conversations, agent may have learned

> Family travels with two young children.
> They prefer fewer hotel changes.
> They disliked an earlier trip that involved changing cities every two days.

That's memory. It wasn't created specifically for this Italy-planning session, but it can improve this and future planning.

So, in one picture

- **Retrieval:** *What does the outside world say that's relevant?*
- **State:** *Where are we in solving this task?*
- **Memory:** *What have I learned about you from previous tasks?*

Prompt Engineering vs Context Engineering

- **Prompt engineering:** how the task and response behavior are expressed
- **Context engineering:** what information, tools, history, state, and constraints are supplied
- Good wording cannot compensate for absent or irrelevant evidence
- Context engineering is a system-level activity

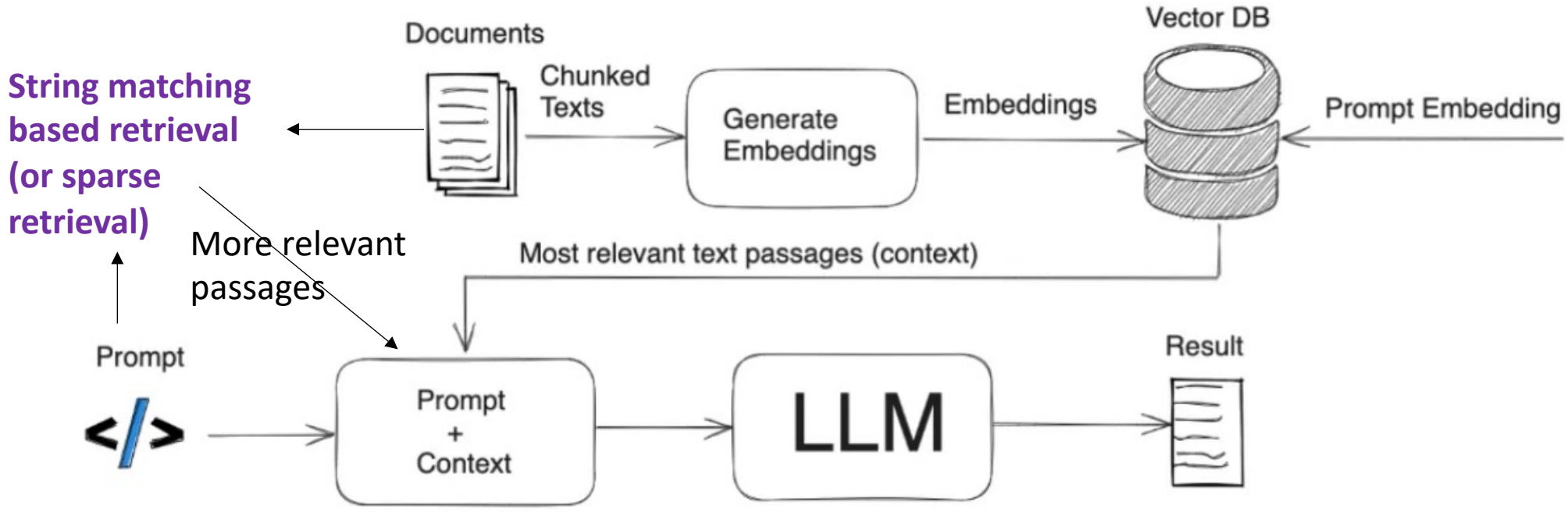
Retrieval Augmented Generation

What is RAG?

RAG (Retrieval-Augmented Generation) retrieves relevant information from an external knowledge source and puts it into the model's context before it generates an answer.

- External corpus or knowledge source
- Retrieval stage before generation
- Evidence supplied at inference time
- Potential citations and abstention
- No model-weight update required

A typicall RAG workflow



https://medium.com/@kushalsharma0508/rag-workflow-retrieval-augmented-generation-ai-5baee937d71c

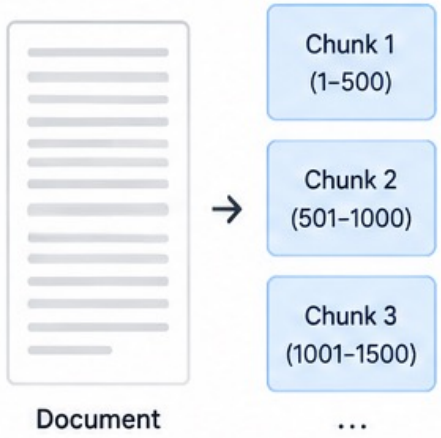
Step1: Documents & Chunked Texts

- Example:
  - Consider you have a collection of scientific research papers on climate change.
  - These documents are divided into manageable text chunks (e.g., paragraphs or sections) to make processing easier.
- Why perform chunking?
  - Whole documents may exceed retrieval or context limits
  - Queries usually target a passage rather than an entire document
  - Smaller chunks improve localization and precision
  - However, excessively small chunks lose surrounding meaning

Common Chunking Methods

1. Fixed-size Chunking

Split text into equal-sized chunks (e.g., 500 tokens).

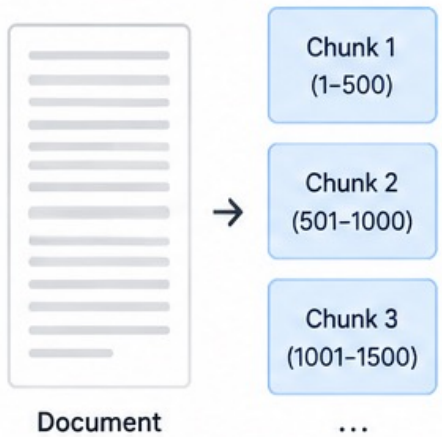


Simple and fast.
Good baseline.

Can split in the middle of a sentence or idea.

1. Fixed-size Chunking

Split text into equal-sized chunks (e.g., 500 tokens).

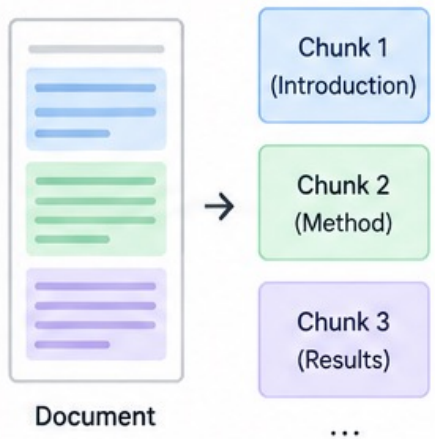


Simple and fast.
Good baseline.

Can split in the middle of a sentence or idea.

2. Semantic Chunking

Split at natural boundaries (e.g., paragraphs, headings, or meaning-based splits).

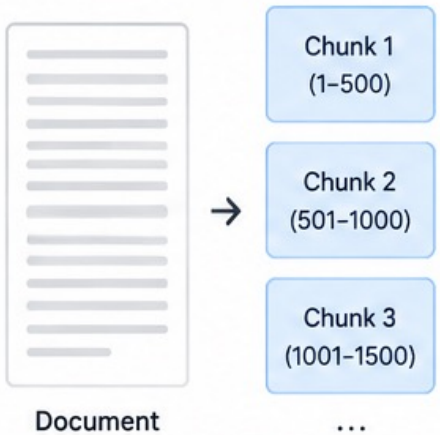


Keeps related content together.

Better context and coherence.

1. Fixed-size Chunking

Split text into equal-sized chunks (e.g., 500 tokens).

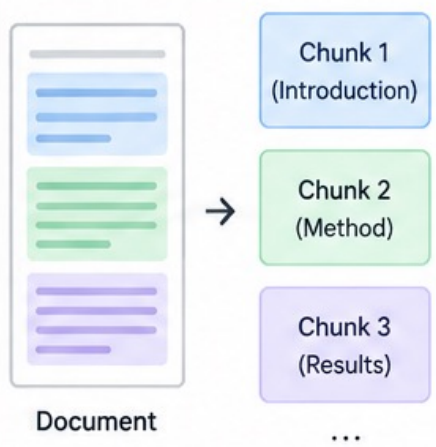


Simple and fast.
Good baseline.

Can split in the middle of a sentence or idea.

2. Semantic Chunking

Split at natural boundaries (e.g., paragraphs, headings, or meaning-based splits).

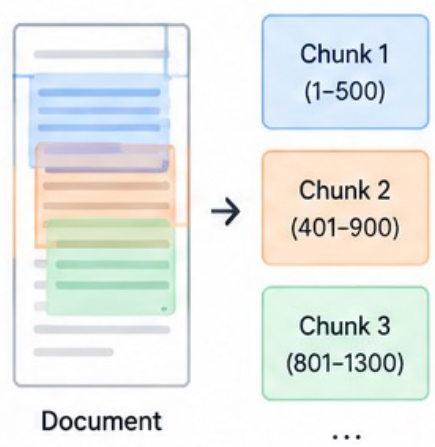


Keeps related content together.

Better context and coherence.

3. Overlapping Chunking

Split into chunks with overlaps (e.g., 500 tokens with 100 token overlap).

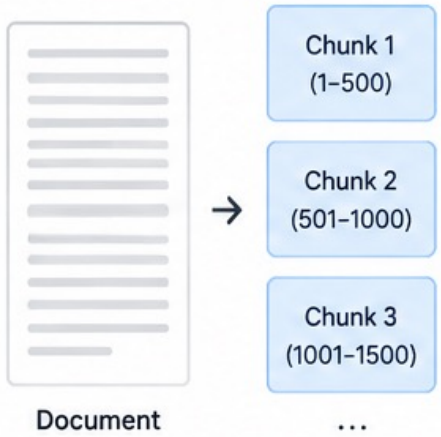


Reduces loss of context at chunk boundaries.

Common and effective in practice.

1. Fixed-size Chunking

Split text into equal-sized chunks (e.g., 500 tokens).

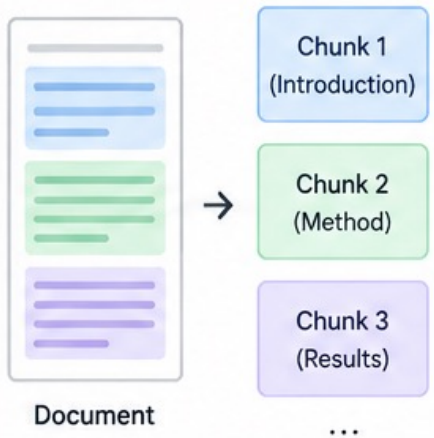


Simple and fast.
Good baseline.

Can split in the middle of a sentence or idea.

2. Semantic Chunking

Split at natural boundaries (e.g., paragraphs, headings, or meaning-based splits).

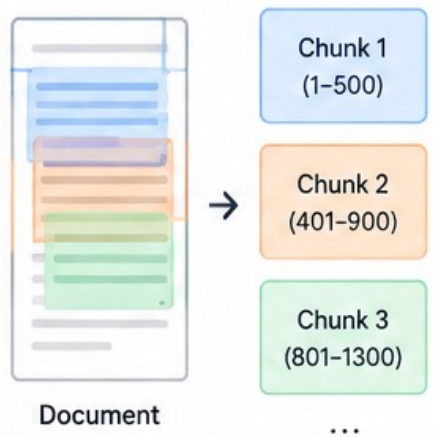


Keeps related content together.

Better context and coherence.

3. Overlapping Chunking

Split into chunks with overlaps (e.g., 500 tokens with 100 token overlap).

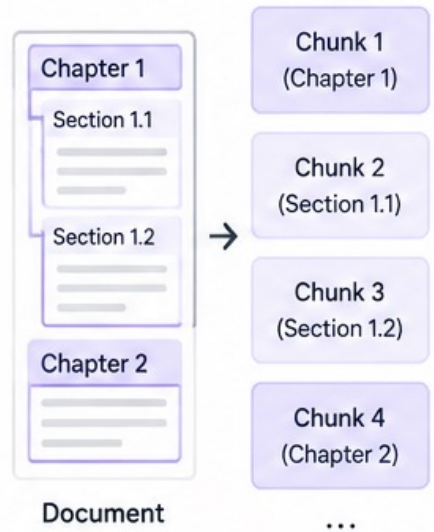


Reduces loss of context at chunk boundaries.

Common and effective in practice.

4. Hierarchical Chunking

Split by document structure (e.g., chapters → sections → paragraphs).

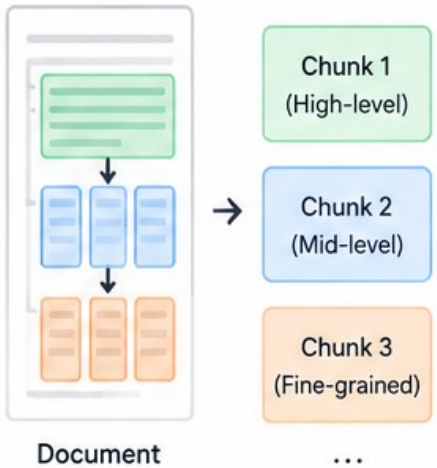


Preserves document structure and hierarchy.

Useful for long, structured documents (e.g., reports, manuals).

5. Recursive Chunking

Try larger semantic units first, then recursively split until chunks fit a target size.



Balances meaning and size automatically.

Adapts to content of different types and lengths.

Step 2: Generate Embeddings

- Each chunk of text is converted into a numerical representation known as an embedding using a **pre-trained model**, for example, BERT or LLaMa
- For example, an embedding for a chunk about "the impact of greenhouse gases" would capture its semantic meaning in vector form.

Step 3: Vector Database and Indexing

- All generated embeddings are stored in a **vector database**. This database can be queried efficiently to find embeddings that are most relevant to a specific search or prompt.
- Example: Using FAISS to index and create a vector database

Step 4: Prompt Embedding & Retrieval

- **Search:** Convert the prompt into an embedding and find similar document chunks.
- **Retrieve:** Return the most relevant chunks for the prompt.

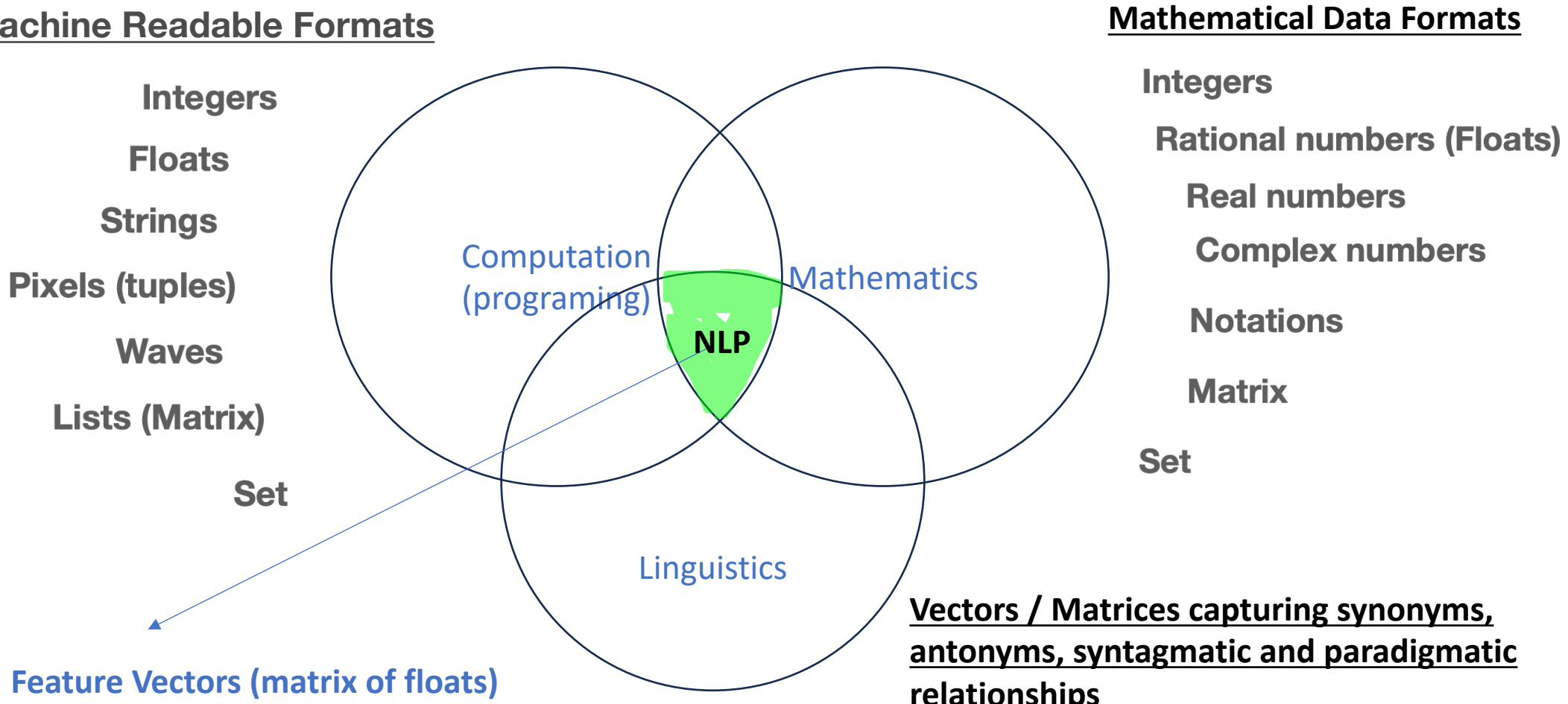
Step 5: Combine Prompt + Context

- **Combine:** Prompt + retrieved context are sent to the LLM.
- **Generate:** The LLM uses the context to create an informed response.
- **Output:** The final response combines the prompt with relevant retrieved information.

Embeddings and Dense Passage Retrieval

Premble: Need for numerical representations

- Text is represented as strings, and mathematical algorithsms don't understand strings.
- **Extract a computer understandable, mathematically and linguistically acceptable format**



How to Represent Tokens / Words in Documents

- **Unigram (Bag-of-words) vector for words and N-hot vectors for sentences.**
- **Word vectors learned using unlabeled corpora**
  - Matrix Factorization based (e.g., Latent Semantic Analysis)
  - Neural Network based (Word2Vec, Glove)
- **Recently:** Sentence and paragraph vectors learned Large Language Models (LLMs)

Traditionally: 1-hot Vectorization

- Words are categorical in nature – can represent in 1-hot format
- Consider this example

Let us learn machine learning.

- Extract words (or more formally tokens in the sentence)

[“Let”,“us”, “learn”, “machine”, “learning”, “.”]

- Treat each unique word as a categorical representation
- Say, convert words to 1-hot vector

[“000001”,“000010”, “000100”, “001000”, “010000”, “100000”]

Example Corpus

- Consider we have three example sentences

1. *let us learn machine learning*
2. *machine learning emphasizes on learning programs from data*
3. *machine learning is a branch of AI*

Example

- Consider we have three example sentences

1. *let us learn machine learning*
2. *machine learning emphasizes on learning programs from data*
3. *machine learning is a branch of AI*

Unique words (a.k.a. vocabulary):

[“let”, “us”, “learn”, “machine”, “learning”, “emphasizes”, “on”, “programs”, “from”, “data”, “is”, “a”, “branch”, “of”, “AI”]

Example

- **Represent words in one hot form**

“let”: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]
“us”: [0,0,0,0,0,0,0,0,0,0,0,0,0,1,0]
“learn”: [0,0,0,0,0,0,0,0,0,0,0,0,1,0,0]
“machine”: [0,0,0,0,0,0,0,0,0,0,0,1,0,0,0]
“learning”: [0,0,0,0,0,0,0,0,0,0,1,0,0,0,0]
“emphasizes”: [0,0,0,0,0,0,0,0,0,1,0,0,0,0,0]
“on”: [0,0,0,0,0,0,0,0,1,0,0,0,0,0,0]
“programs”: [0,0,0,0,0,0,0,1,0,0,0,0,0,0,0]
“from”: [0,0,0,0,0,0,1,0,0,0,0,0,0,0,0]
“data”: [0,0,0,0,0,1,0,0,0,0,0,0,0,0,0]
“is”: [0,0,0,0,1,0,0,0,0,0,0,0,0,0,0]
“a”: [0,0,0,1,0,0,0,0,0,0,0,0,0,0,0]
“branch”: [0,0,1,0,0,0,0,0,0,0,0,0,0,0,0]
“of”: [0,1,0,0,0,0,0,0,0,0,0,0,0,0,0]
“AI”: [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

Representing sentences / paragraphs

let us learn machine learning =>

[0,0,0,0,0,0,0,0,0,0,0,0,0,0,1] +
[0,0,0,0,0,0,0,0,0,0,0,0,0,1,0] +
[0,0,0,0,0,0,0,0,0,0,0,0,1,0,0] +
[0,0,0,0,0,0,0,0,0,0,0,1,0,0,0] +
[0,0,0,0,0,0,0,0,0,0,1,0,0,0,0]

= [0,0,0,0,0,0,0,0,0,0,1,1,1,1,1]

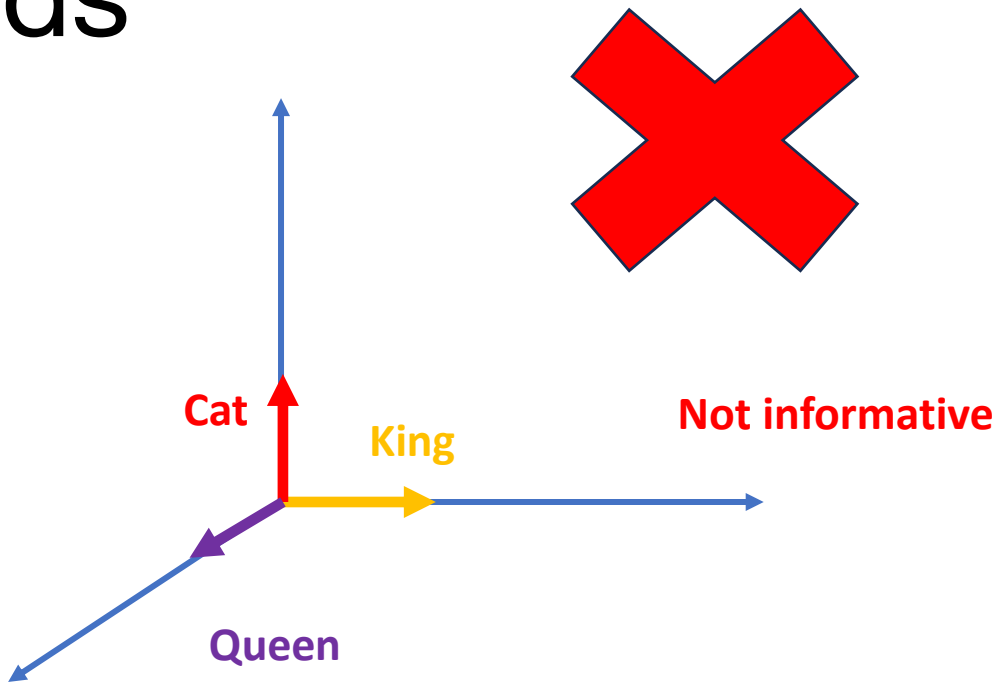
N-hot vectors

Problem with 1-hot vectorization

- Presence / absence based featurization does not specify the strength of the word
- Sometimes we maintain a count of occurrence of the words instead of presence / absence
  - Term frequence
  - Normalized Term frequency (TF-IDF)
- They all suffer from sparsity issues (too many zeros), little information related to meaning.

One-hot encoding of words

Word	1-hot vector
Queen	[1, 0, 0]
King	[0, 1, 0]
Cat	[0, 0, 1]

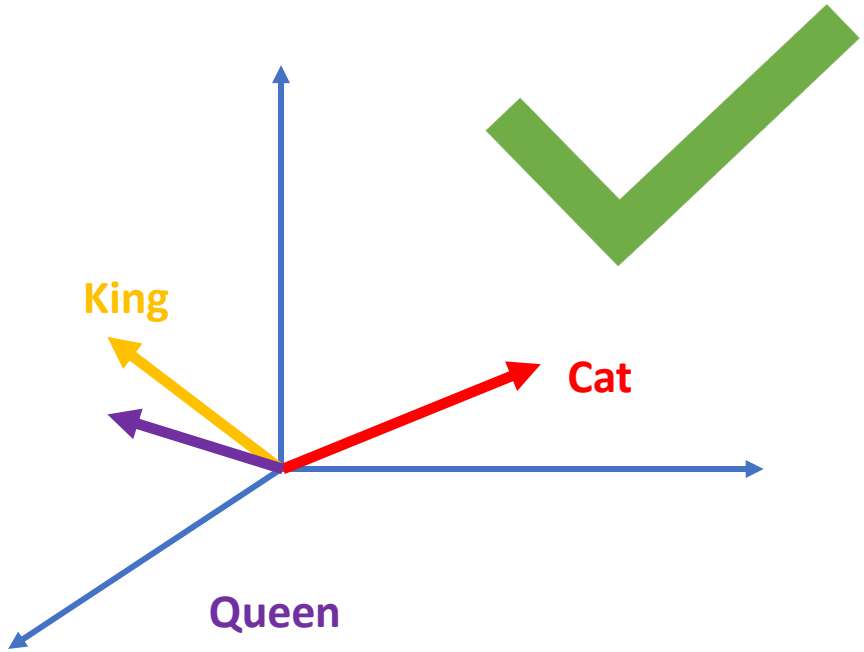


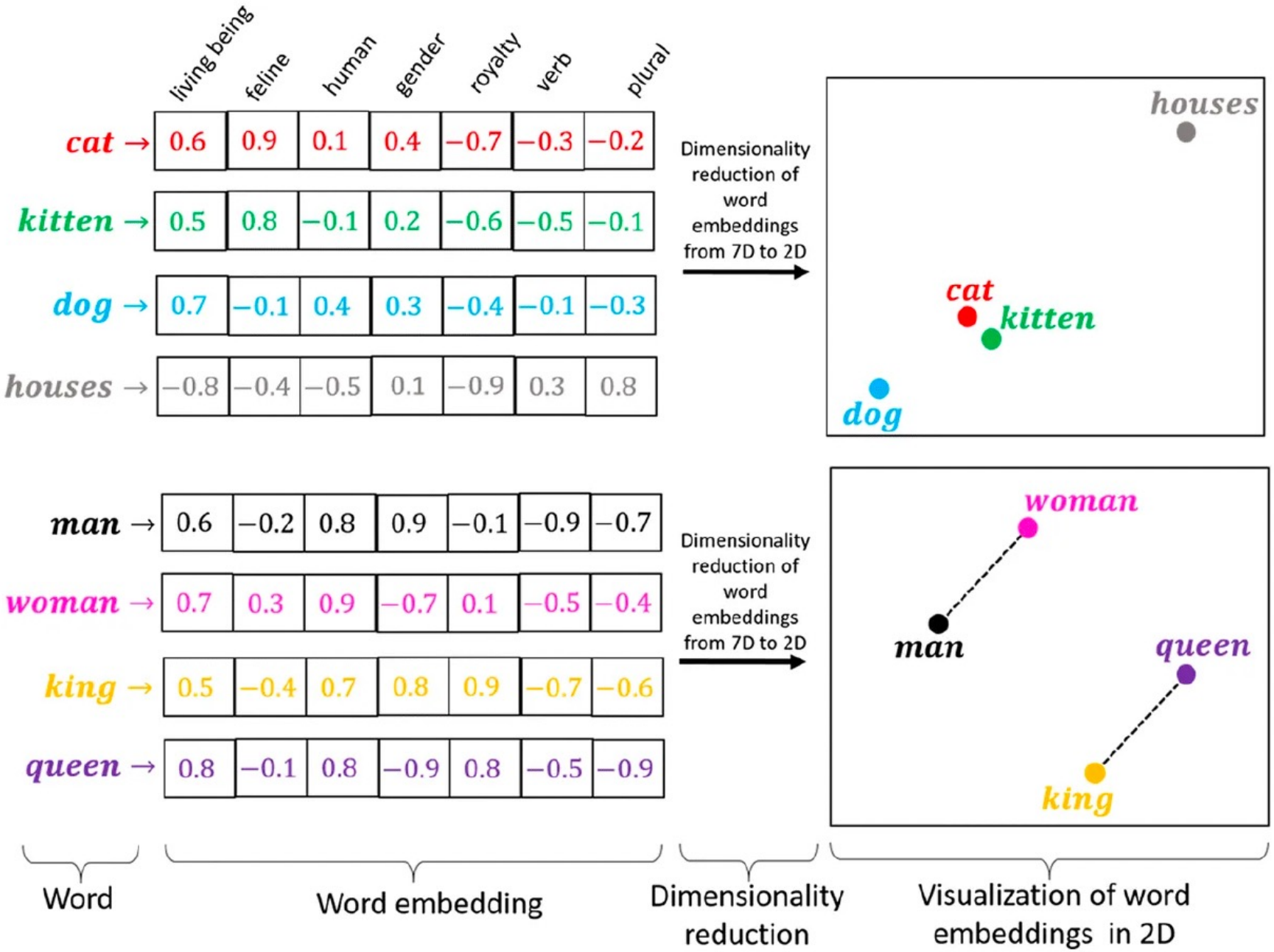
Word Embeddings (2013-)

- One-hot / n-hot vectors are sparse, shallow, are not linguistically well motivated
- We rather want dense representations that capture semantic relationships between words
- **“Word vectors / embeddings”** learned from large text corpora offer such dense representations
- Can generalize across tasks and reduce sparsity

Word Embeddings

Word	1-hot vector
Queen	[1.5, -1.3, -0.9]
King	[2.1, -0.7, 0.2]
Cat	[0.3, 1.9, -0.4]





https://medium.com/@hari4om/word-embedding-d816f643140

Word vectors are pre-trained

- **Word2Vec:**
  - Developed by Google, Word2Vec is one of the earliest and most well-known word vector models.
  - It learns word embeddings by predicting the context words given a target word or vice versa.
- **GLoVE:**
  - GloVe is another popular word vector model that focuses on capturing global word co-occurrence statistics.
  - It leverages both local and global context to create word embeddings.
  - Pre-trained GloVe models are available in different dimensions and trained on diverse text sources.

Getting Sentence / Document embeddings from Word vectors

- Document vectors => average of all word vectors in the document
- **Problem?**
  - Order does not matter
  - Context does not matter
    - Polysemous words (words with different meanings) always map to same word embedding

Solution?

- Direct context vector extraction from sentences / paragraphs
  - E.g., Bidirectional Encoder Representations from Transformers (BERT)
  - Recently developed Large Language Models e.g., GPT series

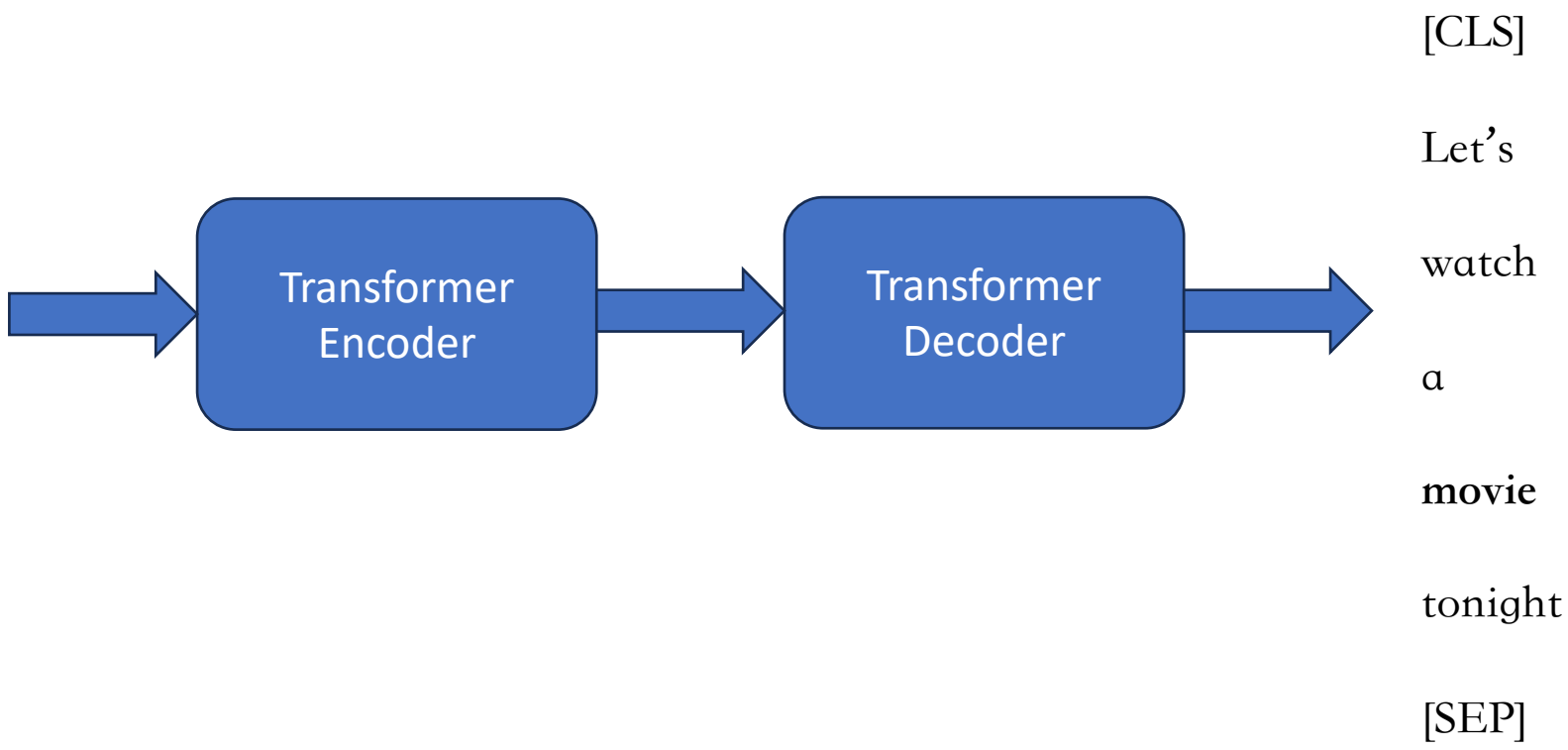
Sentence embeddings: BERT Example

- BERT: **B**idirectional **E**ncoder **R**epresentation **T**ransformers
- Trained with two objectives :
  - Masked token prediction
  - Next sentence prediction

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

BERT Training Objective: Masked Token Prediction



BERT Example

[CLS]

Let's

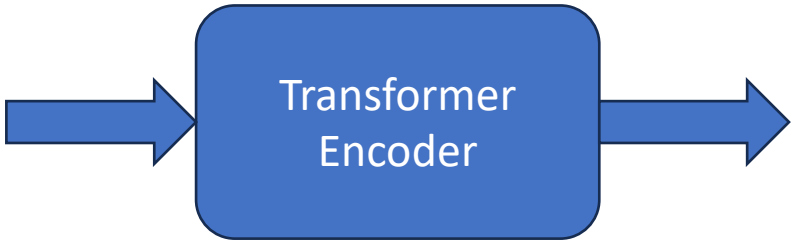
watch

a

movie

tonight

[SEP]



Sentence Embeddings Extraction /
Sentence Feature Extraction

We only use a portion of a network that helps extract features

(also known as encoder)

Popular Text Embedders

- **OpenAI** (text-embedding-3)
- **Cohere** (embed-3)
- **Microsoft** (E5)
- **BAAI** (BGE)
- **Sentence-Transformers**
- **Google** (text-embedding)
- **Amazon** (Titan Embeddings)
- **Mistral** (mistral-embed)
- **Voyage AI** (voyage-3)
- **Jina AI** (jina-embeddings)
- **Nomic** (nomic-embed)
- **Snowflake** (Arctic Embed)

Dense Passage Retrieval using Document Embeddings

- Treat the query as a tiny document
- Represent the query and every document each as a word vector in a word space
- Rank documents according to their proximity to the query in the word space
- Vector Similarity is Key

Example

- Query: *“Can I get my money back if my flight is cancelled?”*

Document Embedding (hypothetical): [0.90, 0.80, 0.10]

Passage	Text	Embedding (hypothetical)	Similarity
P1	Cancelled flights are eligible for a full refund.	[0.88, 0.82, 0.12]	0.99 ✓
P2	You can change your seat before departure.	[0.15, 0.25, 0.85]	0.38
P3	Baggage fees depend on ticket type.	[0.10, 0.18, 0.91]	0.29
P4	Refunds return to the original payment method.	[0.76, 0.68, 0.20]	0.97 ✓

Vector Similarity Metrics

- **Euclidean Distance** between two N-dimensional embeddings X^1, X^2

$$d(X^1, X^2) = \sqrt{\sum_{i=1}^{N} (x_i^1 - x_i^2)^2}$$

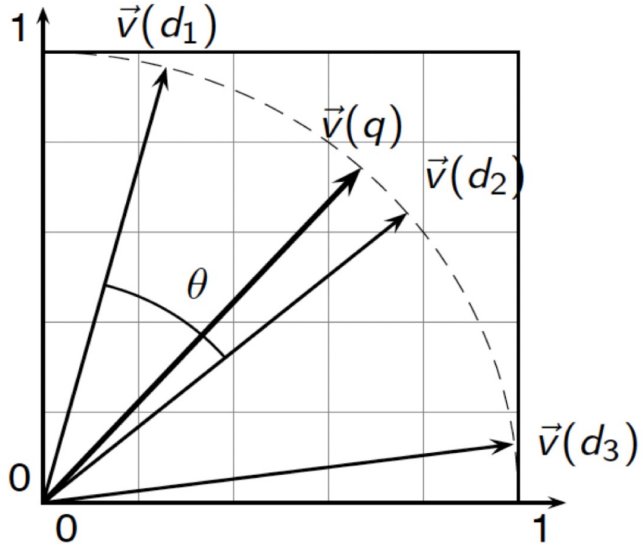
- Not well suited for high dimensional vector spaces
- Short query and long documents will always have big Euclidean distance (if not normalized)

Vector Similarity Metrics

- Cosine Distance

$$\text{cosine}\,(X^{1``}\,X^{2}) = \;\; 1 - \frac{\sum_{i=1}^{N} x_i^1 x_i^2}{\sqrt{\sum_{i=1}^{N} (x_i^1)^2} \cdot \sqrt{\sum_{i=1}^{N} (x_i^2)^2}}$$

- Measures angular distance (similarity) and hence normalized by design



Why Dense Retrieval Isn't Always Enough

- Dense retrieval is great at **meaning**, but sometimes the **exact words matter**.
- **Query:** *“What does error code XJ-417 mean?”*

- A dense retriever might return:
  - P1: *“Troubleshooting common connection errors…”* — **0.89**
  - P2: *“How to resolve authentication failures…”* — **0.84**
- But the most useful passage may be:
  - P3: *“XJ-417: Certificate expired on gateway.”*

Sparse Retrieval

- Sparse retrieval represents text using **words/tokens** rather than dense semantic vectors.
- **BM25** is the classic sparse retrieval algorithm. It ranks passages based roughly on:
  - **BM25: Exact term match + term rarity + term frequency**, while normalizing for document length.
- **Dense retrieval:** “Find passages that *mean* something similar.”
- **BM25:** “Find passages containing the *specific terms* that matter.”

Combining Retrieval Techniques — Hybrid Search

- **Query:** *"What does error code XJ-417 mean?"*

Rank	Dense Retrieval	BM25
1	Authentication troubleshooting	XJ-417: Certificate expired
2	Connection error guide	Gateway error codes
3	XJ-417: Certificate expired	Certificate troubleshooting
4	Certificate troubleshooting	Authentication troubleshooting
5	Gateway error codes	Connection error guide

Reciprocal Rank Fusion (RRF)

- Instead of combining **scores**, combine **ranks**:

$$\mathrm{RRF}(d) = \sum_{r \in R} \frac{1}{k + \mathrm{rank}_r(d)}$$

- Where:
  - **d** = a document/passage being ranked
  - **R** = set of retrieval methods, e.g. {Dense, BM25}
  - **r** = one retrieval method
  - $\mathrm{rank_r(d)}$ = position of document d in retriever r's results
  - **k** = smoothing constant that reduces the impact of very high ranks (commonly **60**)

Query: *"What does error code XJ-417 mean?"*

Using K = 60

$$RRF(d) = \frac{1}{60 + \text{rank}_{Dense}(d)} + \frac{1}{60 + \text{rank}_{BM25}(d)}$$

RRF Rank	Document	Dense Rank	BM25 Rank	RRF Score
1	**XJ-417: Certificate expired**	**3**	**1**	**0.03227**
2	Authentication troubleshooting	1	4	0.03202
3	Gateway error codes	5	2	0.03151
4	Connection error guide	2	5	0.03151
5	Certificate troubleshooting	4	3	0.03150

Questions?

Back to context engineering (Retrieval, Memory and State)

Retrieval is not context assembly

- Retrieved candidates **may overlap or conflict**
- Context budget limits what can be included
- Ordering can affect model behavior
- Evidence should retain source and chunk identifiers
- The system needs an explicit selection policy

Selection, ordering, and compression

- **Select** for **relevance, diversity, authority, and recency**
- **Deduplicate** overlapping chunks
- **Order by** task needs, not merely retrieval score
- **Compress** only when the loss can be assessed
- **Preserve** quotations, dates, qualifications, and provenance

Grounded generation contract

Agents should:

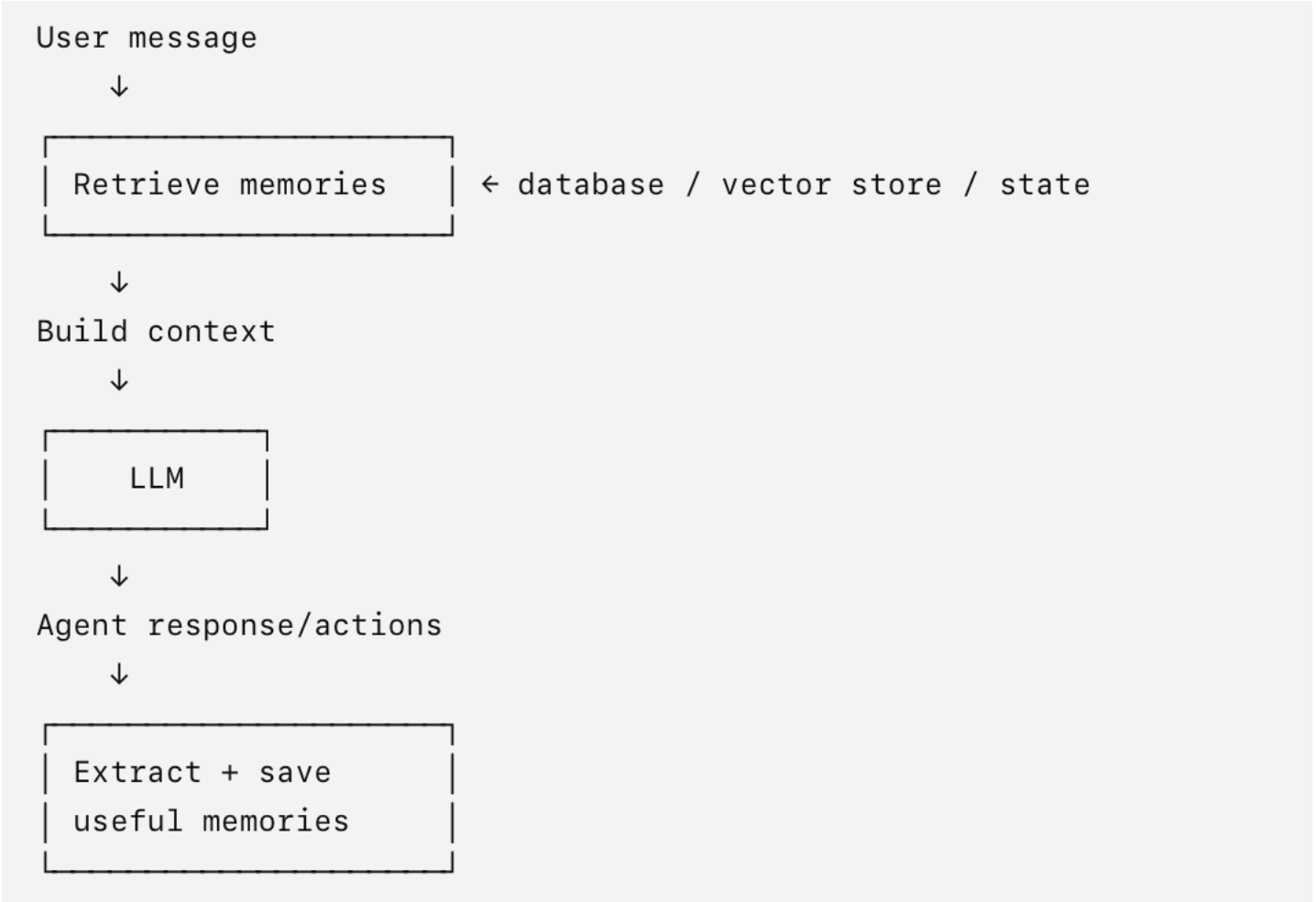
- Answer only from supplied evidence
- Cite source or chunk identifiers
- Distinguish evidence from inference
- Preserve uncertainty and conflicts
- Abstain when evidence is insufficient
- NOT claim retrieval or actions that did not occur

Retrieval, memory, and state are different

- **Retrieval:** obtain information relevant to the current task
- **Memory:** retain information for later use
- **State:** represent current execution and control status
- **Context:** information supplied to a particular model call

Memory

- Could be plain text, JSONS or database
- Relevant memory could be retrieved using database querying, dense / sparse retrieval



The main kinds of memory

- 1. Working / short-term memory
  - This is usually just the agent's current context

```
messages = [
    {"role": "user", "content": "I'm building an ecommerce app"},
    {"role": "assistant", "content": "..."},
    {"role": "user", "content": "Use Postgres for it"}
]
```

Long-term semantic memory

- Information worth remembering across sessions is stored externally

```
memory_id | user_id | content                          | embedding
-------------------------------------------------------------------
183       | 42      | "User prefers PostgreSQL"        | [0.18, ...]
184       | 42      | "Project is ecommerce app"       | [0.42, ...]
```

Episodic memory

- Instead of storing facts, the agent remembers things that happened
- Summarizing Compressing agent actions before storing in memory

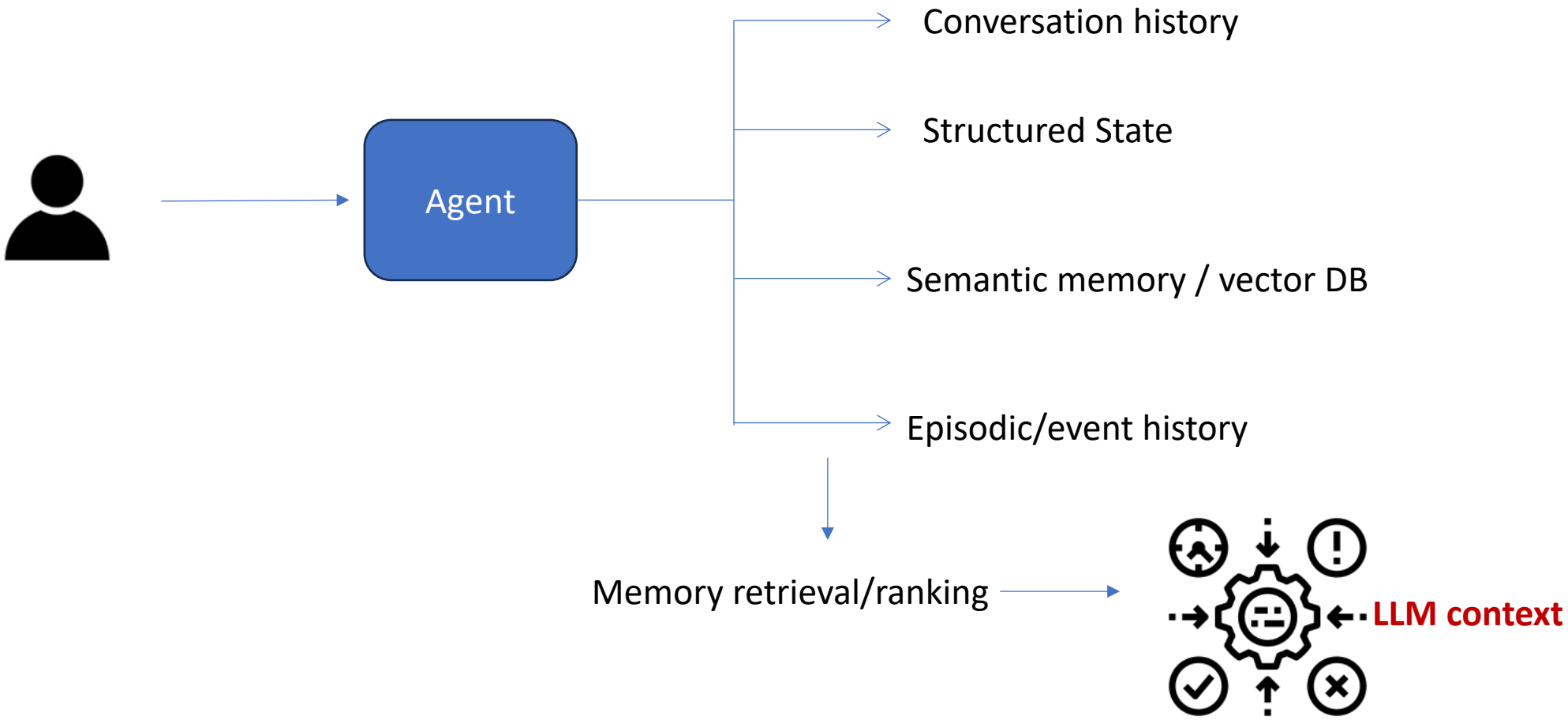
```
2026-09-01
Agent deployed checkout service.
Deployment failed because DATABASE_URL was missing.
Agent fixed the environment variable and redeployed successfully.
```

Structured/state memory

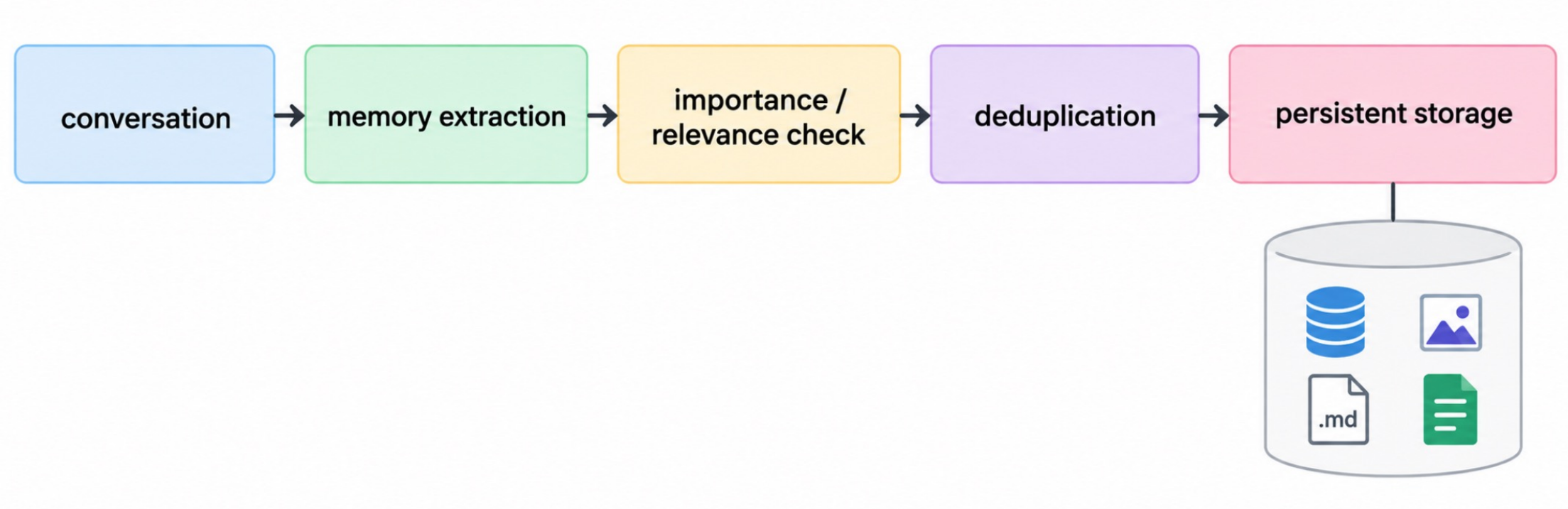
- For many agents, this is actually the most important type

```
state = {
    "goal": "launch ecommerce site",
    "completed": ["schema", "authentication"],
    "current_task": "checkout",
    "blocked_by": ["Stripe credentials"],
}
```

A production architecture



Common memory operations



Summary

- **Context matters as much as the prompt.**
- **Retrieval, memory, and state are different.**
- **Diagnose RAG failures at the right stage.**

Build reliable AI systems by **controlling context, grounding answers in evidence, and diagnosing failures at the right stage.**

Over to tutorial