Introduction to LLM Applications

Lecture 5 Adding Memory to Chats

Wei Xu

Today's goal

- Review the basic Langchain stuff (using it again)
- Short-term memory
  - Context of a chat session
- Long and larger memory
  - How to load documents / files
  - How to find them (or part of them)

- The last lab session on using LLM from APIs
  - We will move on to deploying and running your own models

Where are we now?

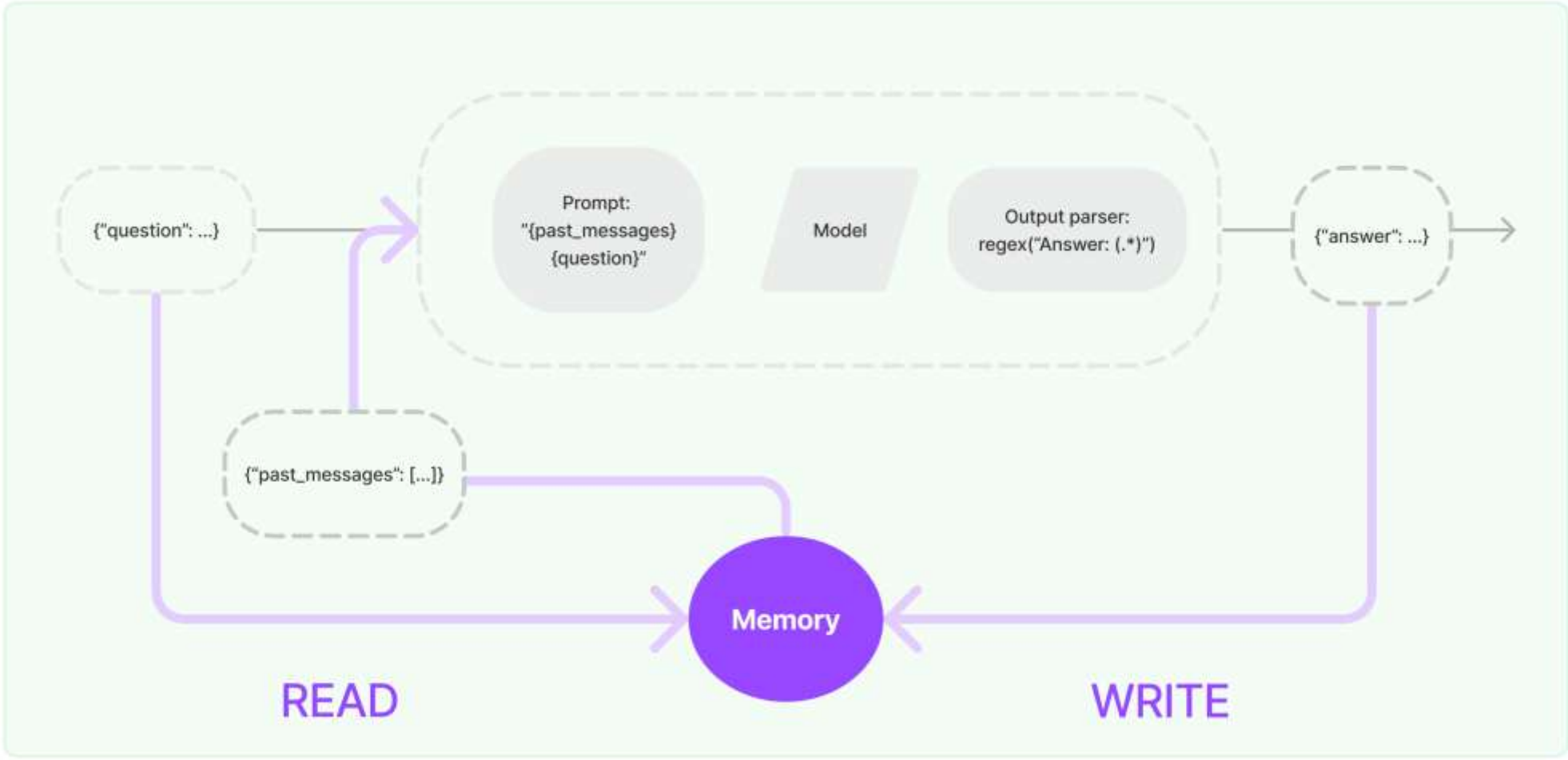
- How do model gain knowledge?

- Through model training or fine-tuning
  - Via model weights (“built-in” knowledge)
- Through model inputs (recall search / tools)
  1. Insert knowledge or context into the input
  2. Ask LLM to incorporate the context in its output

- Today we focus on how to manage the context

SHORT TERM MEMORY: CONTEXT IN CHATS

Adding memory to multi-round chats



ChatMessageHistory

- Technically, no difference from any context
- They are just Langchain’s wrappers of list and map.

```python
from langchain_community.chat_message_histories import ChatMessageHistory

chat_history = ChatMessageHistory()

chat_history.add_user_message(
    "Translate this sentence from English to French: I love programming."
)

chat_history.add_ai_message("J'adore la programmation.")

chat_history.messages
```

Auto history management: RunnableWithMessageHistory

- RunnableWithMessageHistory
  - A special wrapper around a chain so with automatic integration with Chains
  - Takes in a function as argument to get the history object
    - Powerful, but making the interface ugly
    - (see the next page)

Auto history management: RunnableWithMessageHistory

```python
store = {}

def get_session_history(
    user_id: str
) -> BaseChatMessageHistory:
    if (user_id) not in store:
        store[(user_id)] = ChatMessageHistory()
    return store[(user_id)]
```

```python
prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            """You are a chatbot having a conversation with a huma
            Your name is Tom Riddle.
            You need to tell your name to that human if he doesn'
        ),
        ("placeholder", "{chat_history}"),
        ("human", "{input}"),
    ]
)

chain = prompt | chat
```

```python
chain_with_message_history = RunnableWithMessageHistory(
    chain,
    get_session_history=get_session_history,
    input_messages_key="input",
    history_messages_key="chat_history",
    history_factory_config=[  # parameter for the get_session_history
        ConfigurableFieldSpec(
            id="user_id",
            annotation=str,
            name="User ID",
            description="Unique identifier for the user.",
            default="",
            is_shared=True,
        ),
    ],
)
```

```python
chain_with_message_history.invoke(
    {"input": "Hi there, this is Harry Po
    {"configurable": {"user_id": "123"}},
).content
```

Trimming and Summary

- With trimmer, a fixed number of records are maintained by automatically hiding older chat history from the model
- Still in memory, just not passed to the model

```python
trimmer: Any = trim_messages(strategy="last", max_tokens=2, token_counter=len)

chain_with_trimming: RunnableSerializable[dict[s... = (
    RunnablePassthrough.assign(chat_history=itemgetter("chat_history") | trimmer)
    | prompt
    | chat
)

chain_with_trimmed_history = RunnableWithMessageHistory(
    runnable=chain_with_trimming,
    get_session_history=lambda session_id: demo_ephemeral_chat_history,
    input_messages_key="input",
    history_messages_key="chat_history",
)
```

Using chat history with Agents

```
prompt = ZeroShotAgent.create_prompt(
    tools,
    prefix="""Have a conversation with a human, answering the following qu
    suffix="""Begin!
{chat_history}
Question: {input}
{agent_scratchpad}""",
    input_variables=["input", "chat_history", "agent_scratchpad"],
)
memory = ConversationBufferMemory(memory_key="chat_history")
```

```
memory.load_memory_variables({})
```

Limitations of memories

- Memory makes LLM from stateless to stateful
  - As a rule-of-thumb, stateful systems are much more difficult to run
- All these memory are stored in your Python process (life-cycle of your notebook)
- If your process crashes, everything is lost!

- Off-process storage?
  - Use a external DB / VectorDB / GraphDB etc.
  - Later in the semester …

LONG TERM MEMORY: EMBEDDINGS

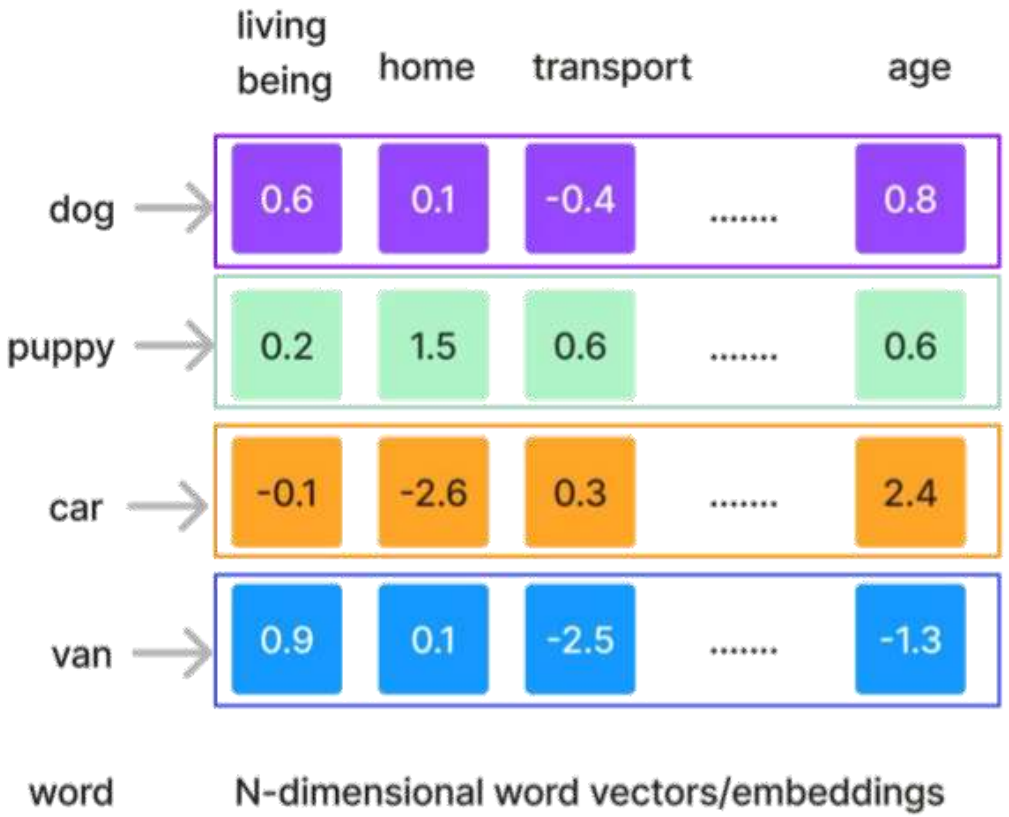
Problems to solve

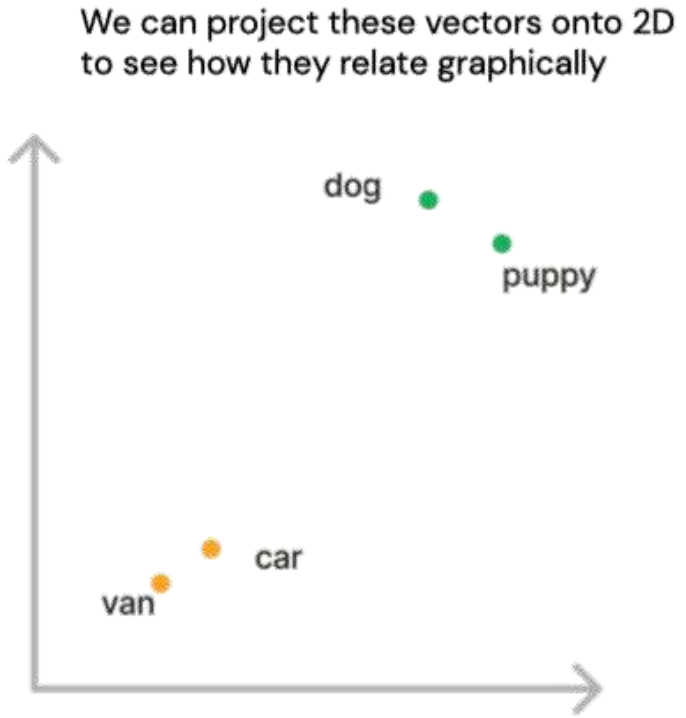
- Context gets too long, and summarization is not useful
  - Query requires details, but not the entire article
- Search engine is not fine-grained enough
  - Only returns pages, and read-only
  - Does not support semantic queries (keywords only)

- Technical problems to solve
  - How do you define “similar semantics”? -> embedding
  - How do you find the “most similar documents?” -> kNN, vector DB

“Similarity”

- We represent words with vectors
- Then we can calculate the “distance” of these vectors





Vectors: Bag of words and n-gram

- Want: more semantically “similar” -> less vector distance
- Bag of words:

  (1) John likes to watch movies. Mary likes movies too.

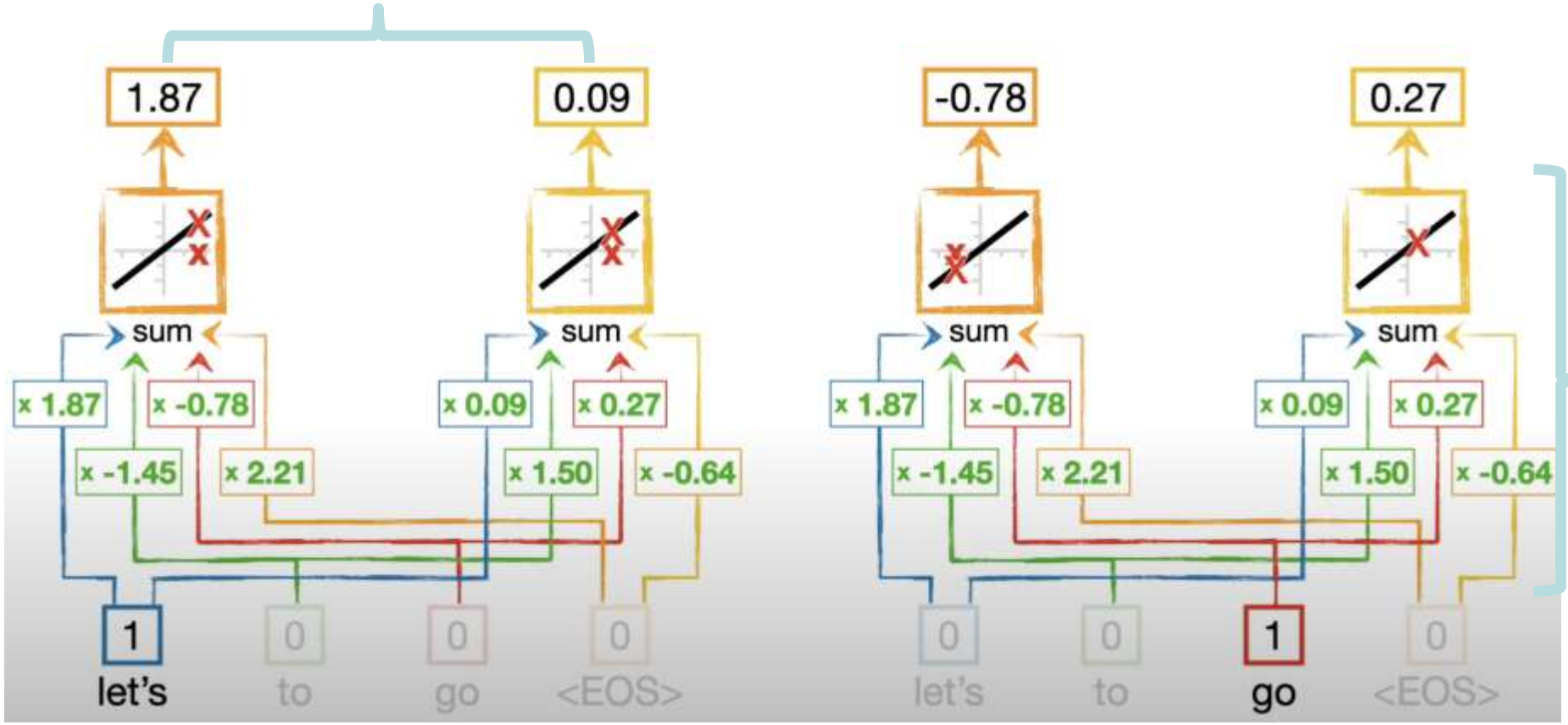
  (2) John also likes to watch football games

Vocabulary	John	likes	to	watch	movies	also	football	games	Mary	too
(1)	1	2	1	1	2	0	0	0	1	1
(2)	1	1	1	1	0	1	1	1	0	0

- n-gram

John also likes to watch football games

Intuition: embedding captures more context

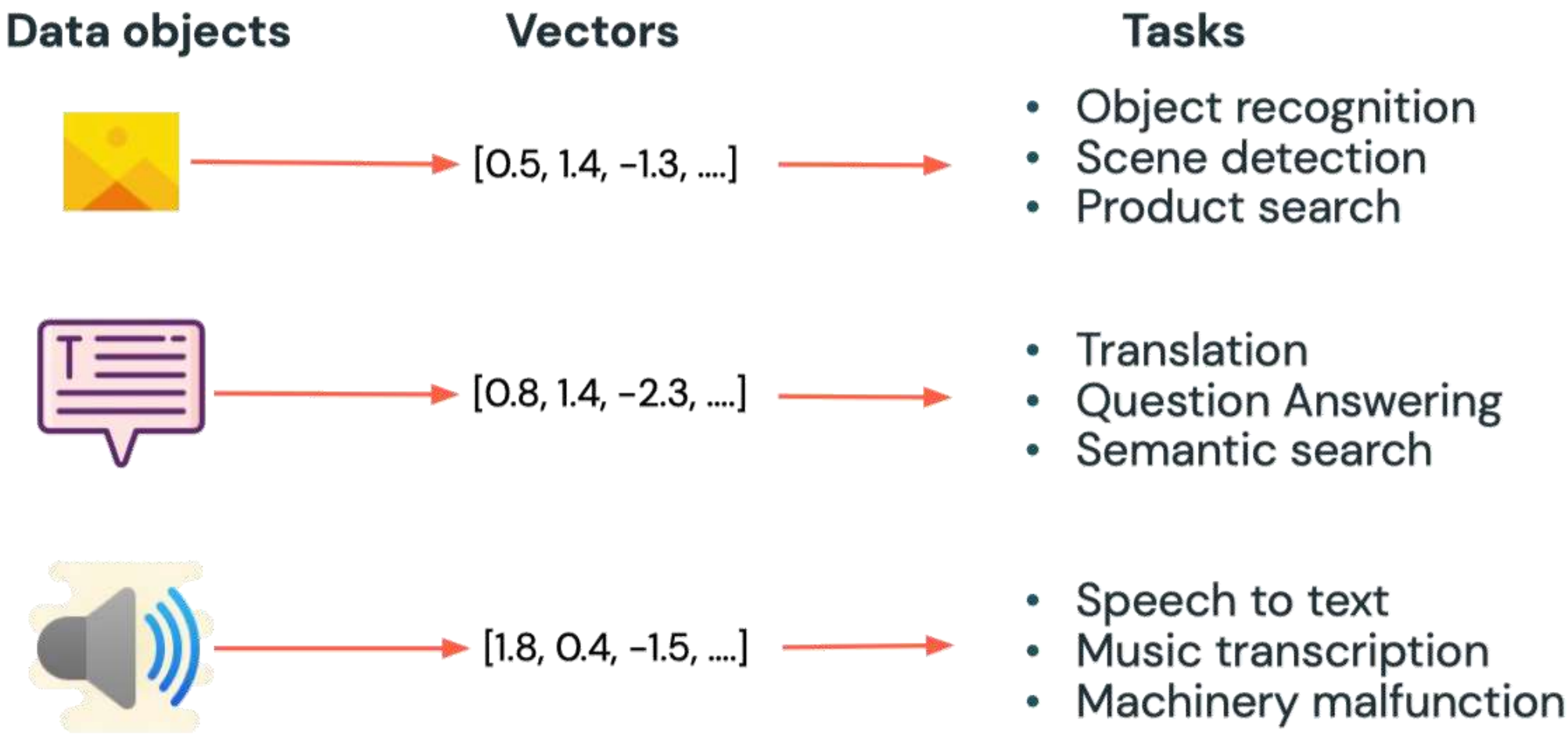


Embedding Network

Ordering: Positional Encoding

Magic: where do those weights come from? -- "training"

Embedding is not just for texts



Getting the embedding directly

```python
import os
from langchain_openai import OpenAIEmbeddings
baai_embedding = OpenAIEmbeddings(
    model="BAAI/bge-m3",
    base_url=os.environ.get("SF_BASE_URL"),
    api_key=os.environ.get("SF_API_KEY"),
)
baai_embedding.embed_query("Harry Potter is a wizard.") # test the em
```

LONG TERM MEMORY: VECTOR STORAGE

Next Question: given a vector, can you find the closest k vectors?

- K-nearest neighbors (KNN)
  - Intuitive algorithm requires computing distance between every pair of vectors: O(n^2)
- Approximate nearest neighbors (ANN)
  - Trade accuracy for speed gains
  - Examples of indexing algorithms:

    Tree-based: ANNOY
    Proximity graphs: HNSW
    Clustering: FAISS
    Hashing: LSH
    Vector compression: SCaNN
- Still very hot research topic nowadays

From ANN to VectorDB

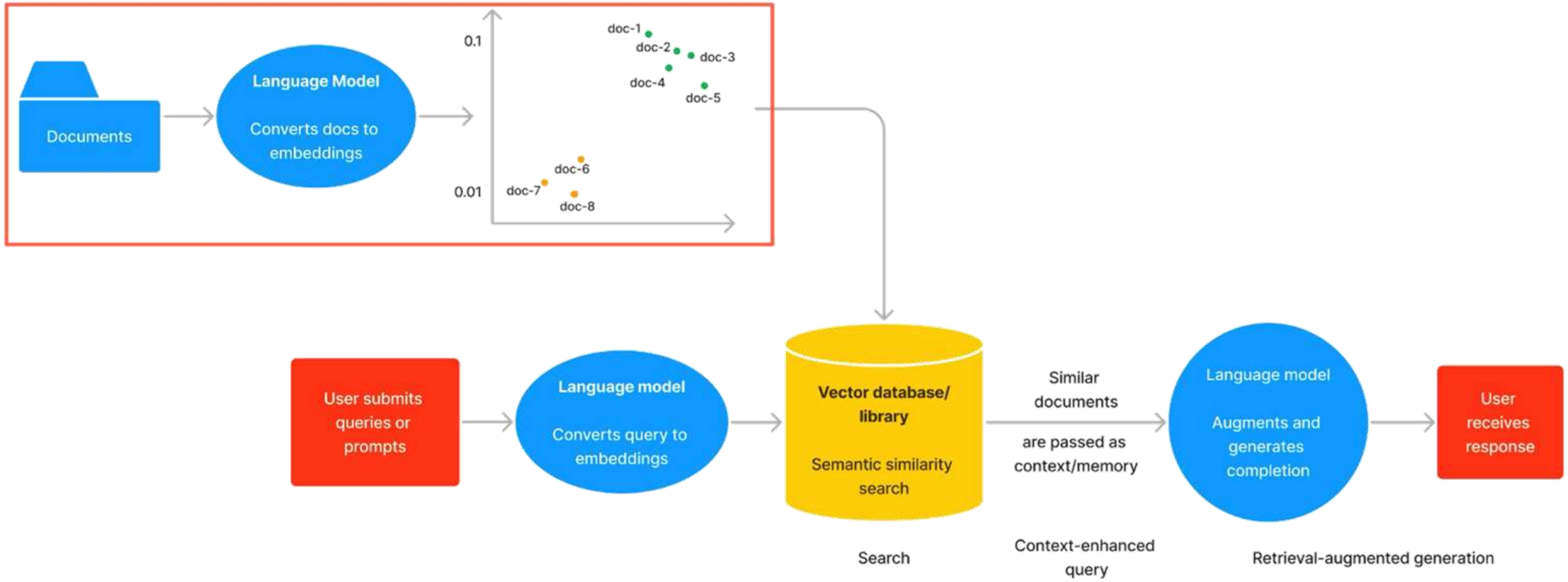
- Specialized, full-fledged databases for unstructured data
  - Inherit database properties
    - e.g., Create-Read-Update-Delete (CRUD)
    - (but usually cannot modify on-the-fly)
  - Can write to disk (avoid recomputing embedding every time)
  - Speed up query search for the closest vectors
- Rely on ANN algorithms
- Organize embeddings into indices

Popular VectorDBs

	Released	Billion-scale vector support	Approximate Nearest Neighbor Algorithm	LangChain Integration
Open-Sourced				
Chroma	2022	No	HNSW	Yes
Milvus	2019	Yes	FAISS, ANNOY, HNSW	
Qdrant	2020	No	HNSW	
Redis	2022	No	HNSW	
Weaviate	2016	No	HNSW	
Vespa	2016	Yes	Modified HNSW	
Not Open-Sourced				
Pinecone	2021	Yes	Proprietary	Yes

From DataBricks, 2024

Retrieval Augmented Generation (RAG)



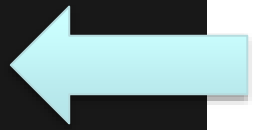
Principle on good embeddings

- Embedding is another layer of magic, on top of prompts ☹
- The embedding model should represent BOTH your queries and documents
  - Should be trained on the same data, at least (best if they are the same embedding model)
  - Otherwise might need to finetune the embedding model (later)

Using the vector DB

```
# compute embeddings and save the embeddings in
from langchain_chroma import Chroma

chroma_dir = "/scratch1/chroma_db"
docsearch_chroma = Chroma(
    embedding_function=baai_embedding,
    persist_directory=chroma_dir,
    collection_name="harry-potter",
)
docsearch_chroma.reset_collection()
docsearch_chroma.add_documents(texts)
```



```
query = 'Who are the robed people Mr. Dursley sees in the streets?'
docs = docsearch_chroma_reloaded.similarity_search(query, k=6)
```

Using Retrieval in an Agent

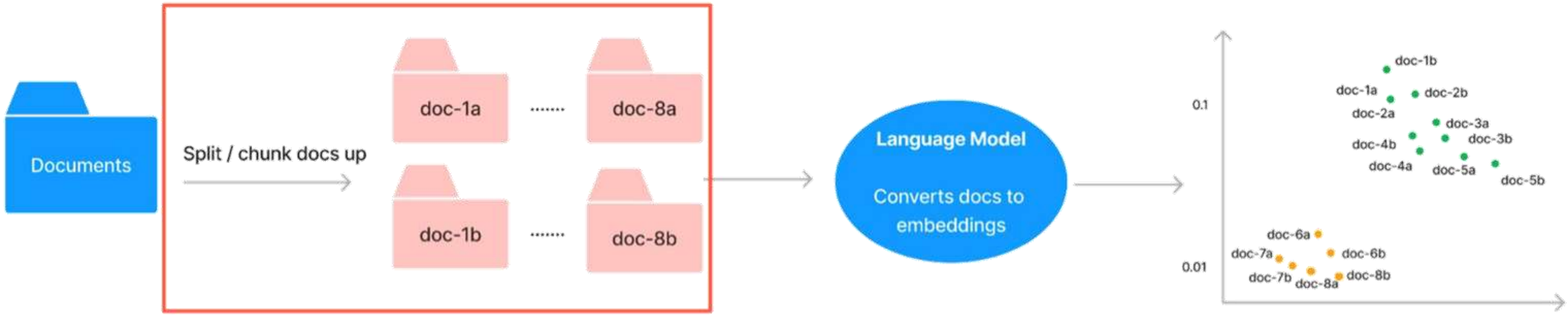
```
from langchain.chains import RetrievalQA
from langchain_openai import ChatOpenAI
llm = ChatOpenAI(temperature=0, model=CHAT_MODEL)
chain = RetrievalQA.from_chain_type(
    llm,
    chain_type="stuff",
    verbose=True,
    retriever=docsearch_chroma_reloaded.as_retriever(k=5)
)
```

You might want to increase k if you don't find what you wnat

SIDE NOTES:
OTHER MAGICS

Yet another magic: how to split up documents?

- Embedding a fixed-sized vector
  - Too long -> lose track of the semantics, exceed token limits, expensive
- Split one document into paragraphs? Sections? Sentences?
  - Too small -> does not capture high-level semantics（“断章取义”）



General chunking advices

- https://www.pinecone.io/learn/chunking-strategies/
- Chunk size depends on
  - Nature of the content (long/short paragraphs?)
  - Your embedding model
    - Sentence-transformer -> individual sentences
    - OpenAI -> fixed sizes of 300-500 tokens
  - Complexity of user queries, use cases

Different kinds of Chunking methods

- Fixed-size
- Sentence
  - NLTK / spaCy
- Recursive
  - Recurse until short enough
- Special format (source file indicates structure)
  - markdown / latex

Other engineering considerations: Reading in different file formats

- How to load / parse different file types
  - Pdf, Word, Markdown, Python source code etc.
  - https://python.langchain.com/docs/integrations/document_loaders/
  - https://python.langchain.com/api_reference/community/document_loaders.html
- Many loaders for pdf files (need trial and error)
  - https://python.langchain.com/docs/how_to/document_loader_pdf/
  - https://python.langchain.com/docs/integrations/document_loaders/pypdfloader/

Your tasks

- Next week: guest lecture
- so your tasks are due the week after next (4/3)
  - Two weeks to do this lab
- Part I and II – mostly demos
  - Optional tasks (including DSPy version of memory) are optional
- Part III: build something non-trivial: query a collection of research papers
  - Choose a good set of Langchain tools
  - Try your best to get a “good” search result (by visual examination on some example queries.)

Plan for the next few weeks

- Form teams of 2-3 people
  - Design an *interesting* LLM *application,* thinking about being FUN or USEFUL
  - The best to be both FUN and USEFUL
  - Also doable during the semester
  - **Team setup / adjustment due next Sunday (4/6)**
- Next week (3/27):
  - industry guest lecture. Chief Scientist from Youdao (an online Education company)
  - Gets more project ideas
- Then the image models lab (4/3), lab due 4/17 – (you still have 2 weeks)
- Then each group needs to present your application idea (4/10)
  - What you want to do, why do you want to do it?
  - What is the general approach
  - What specific resource support you will need (Tokens, GPUs, Data to purchase etc.)
  - **Submit your presentation slides before the lecture on 4/10**

A great sentence template to present a project idea

To solve *(problem)* for *(target_user)*, we create *(your_app)*, comparing to existing solutions, our solution *(one sentence benefit)*.

An example:

To *avoid trouble of installing layers of software* for *all student taking the course*, we create *docker images with everything installed*.
Comparing to existing solutions *(e.g. running configuration scripts)*, our solution *starts much faster.*