Learning LangGraph

Course by **Matteo Falcioni**

Mail: ***matteo.falcioni3@unibo.it***

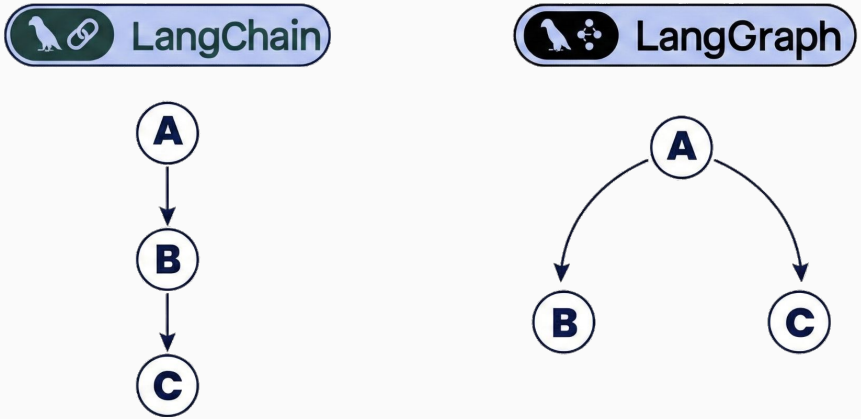
GitHub: ***https://github.com/MatteoFalcioni***

Unibo, DIFA
Science City Lab

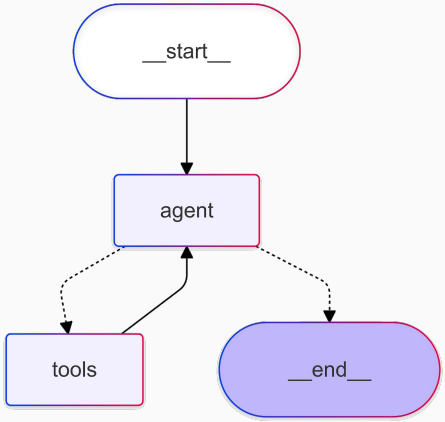
Table of Contents

LangChain & LangGraph

LangGraph is part of the LangChain ecosystem, but their structure is inherently different

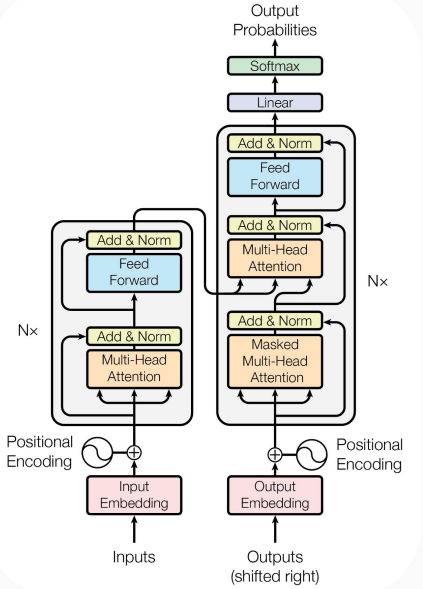


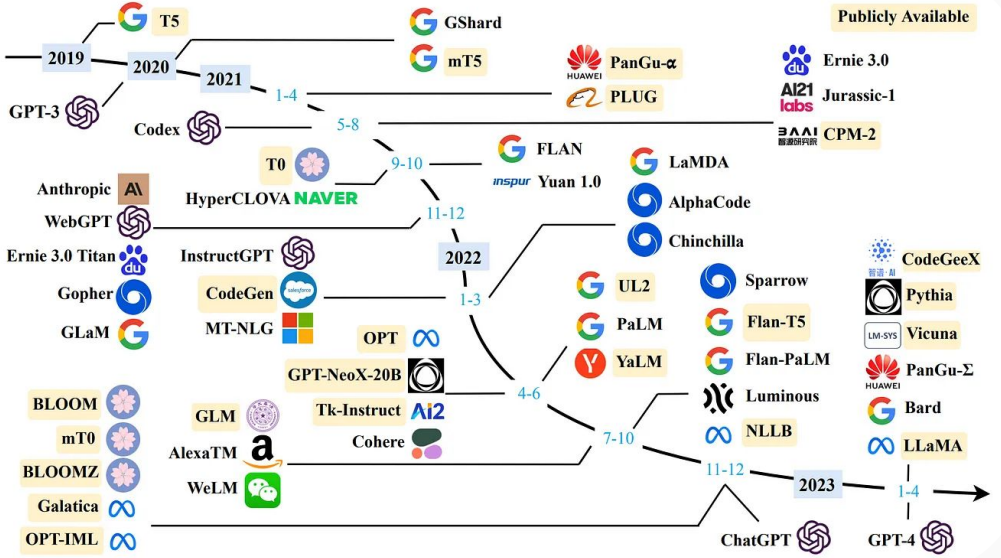
LangGraph allows us to have **loops** (→ agents with tools)



Large Language Models - LLMs

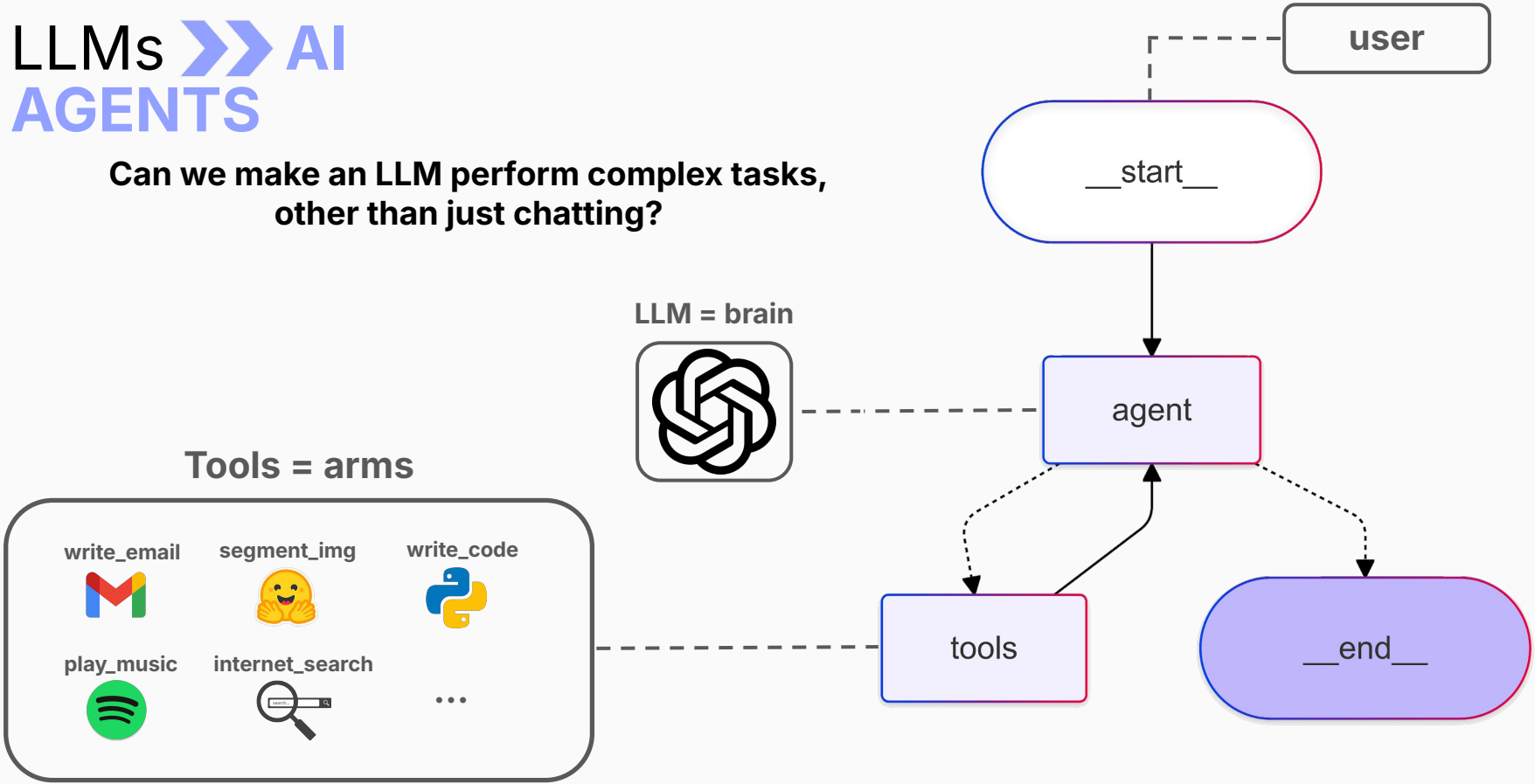
Attention is all you need!
(2017)





LLMs >> AI AGENTS

Can we make an LLM perform complex tasks, other than just chatting?



Resources

- Course on GitHub: https://github.com/MatteoFalcioni/Learning-LangGraph
- LangChain Academy: LangChain Academy
- LangGraph Graph API: Graph API overview - Docs by LangChain
- LangChain Docs: LangChain overview - Docs by LangChain
- LangChain Reference: Home | LangChain Reference
- LangChain OpenTutorial: 🔗 The LangChain Open Tutorial for Everyone

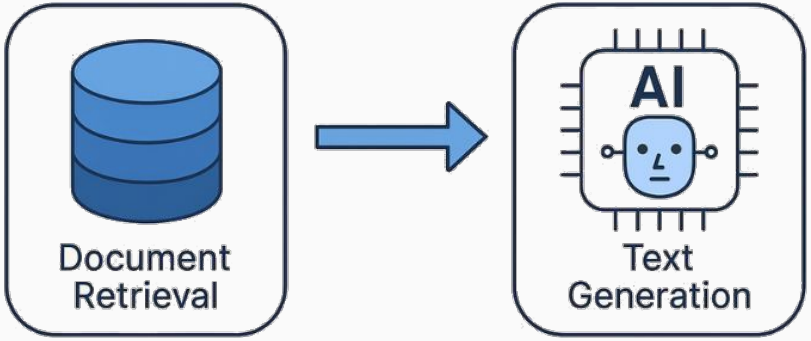
For any information or doubts don't hesitate to contact me at *matteo.falcioni3@unibo.it* or *matteo.falcioni@outlook.com*

Neural Networks

How do LLMs work?

Retrieval Augmented Generation - **RAG**

Technique to ground your LLM answers on given sources → **less hallucinations, more contextually relevant responses**



Three main steps:

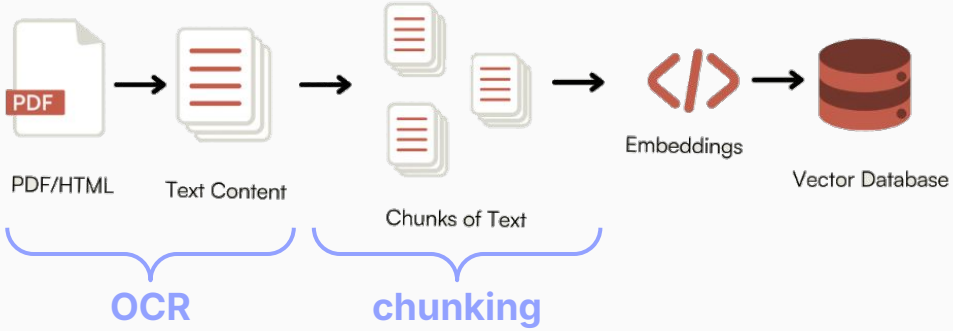
1. **Data Preprocessing (OCR, chunking)**
2. **Embedding textual data in a Vector Database**
3. **Creating a Retrieval Tool for the LLM**

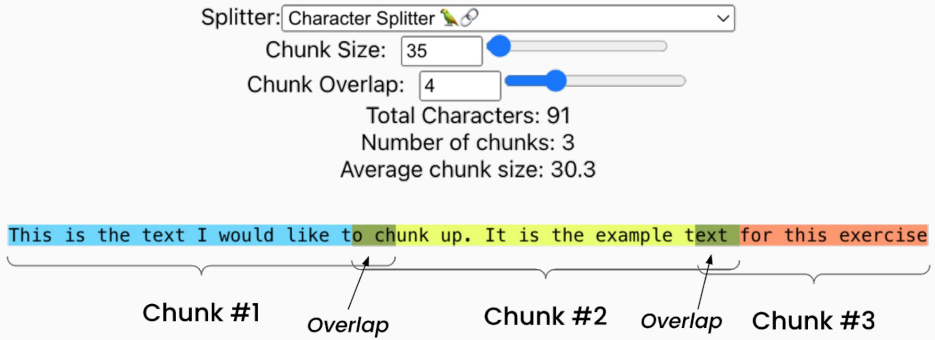
1. Data Preprocessing

RAG works on text: specifically, **chunks of text**.

If our *knowledge base* is already in pure text, we can just split it into chunks using [text splitters]

Otherwise, we may need to also perform OCR to extract the text from our documents (like turning pdf → pure text)



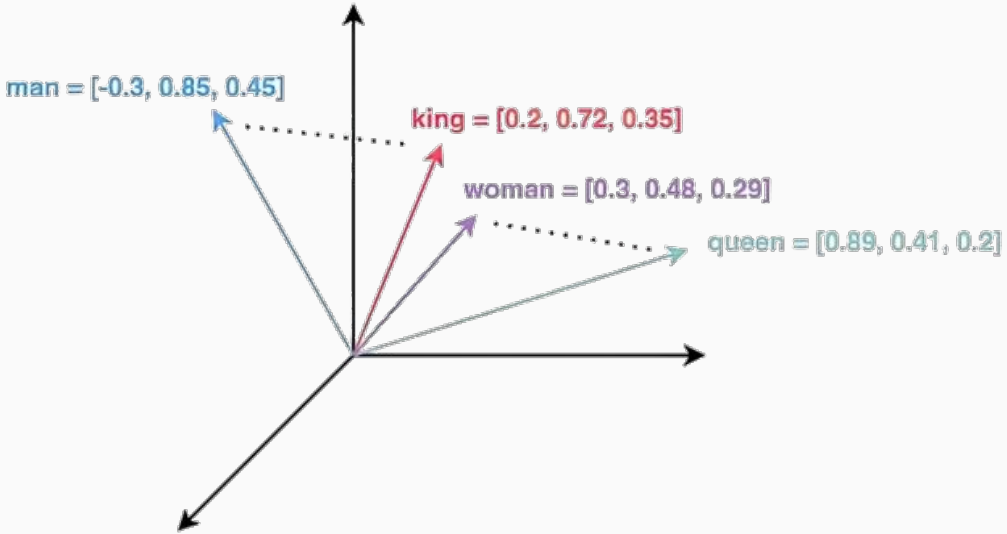


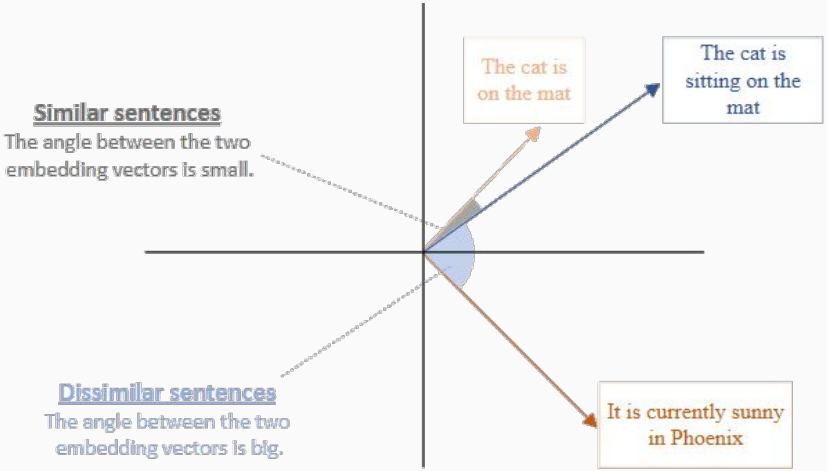
2. Embedding

Embedding = converting the text from our chunks into numerical representations (*through an embedding model*)

In this way, **our chunks become vectors in a vector space** (or *"vector database"*)

→ Chunks with similar meanings will be close





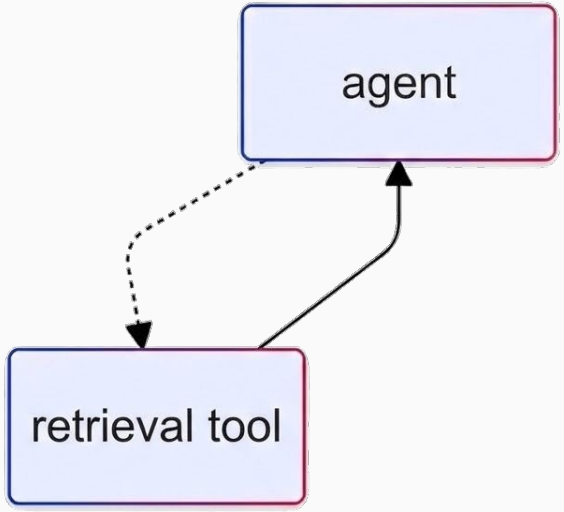
This allows for **efficient search**:

1. ***Convert a given query into its numerical representation***
2. ***Find the k most similar chunks (computing the scalar product)***

3. Create a Retrieval Tool

Last thing we need to do is plug this efficient search into an LLM

→ Create a tool that leverages the vector store retriever



In practice, this is the most simple step:

```
@tool
def retrieve_context(query: str):
    """Retrieve information to help answer a query."""
    docs = vector_store.similarity_search(query, k=3)

    result = "\n\n".join(
        (f"Source: {doc.metadata['source']}, Page: {doc.metadata['global_page_number']}\nContent:
{doc.page_content}")
        for doc in docs)

    return {"messages" : [ToolMessage(content=result)]}
```

References

- Full Implementation on GitHub: pt.1, pt.2 ← multimodal RAG from the course
- Build a Custom RAG Agent with LangGraph (tutorial) ← slightly outdated, but useful
- Build a RAG agent with LangChain (LangChain tutorial) ← simple RAG agent, no graph