VISIUM

PyData Lausanne @ Visium – June 2025

Lauzhack 2025 – Alejandro Bonell, Lucas Eckes, Younes Moussaif – 04.06.2025

Your hosts for today

VISIUM

Alejandro Bonell

Sr. ML Engineer @ Visium

Lucas Eckes

Sr. ML Engineer @ Visium

Younes Moussaif

ML Engineer @ Visium

VISIUM

ABOUT VISIUM

Introduction to who we are

AT VISIUM, WE ENABLE ENTERPRISE ORGANIZATIONS TO BECOME FUTURE-PROOF AND LEADERS IN AI & DATA.

With expertise in strategy, architecture, cloud engineering, data engineering, analytics, AI and ML, we empower you to unleash and scale the power of data.

50+

Visiumees throughout Europe

50+

Happy Enterprise Clients

200+

AI & Data Engagements

VISIUM

Our Partners

ACADEMIA

TECHNOLOGY PLATFORMS

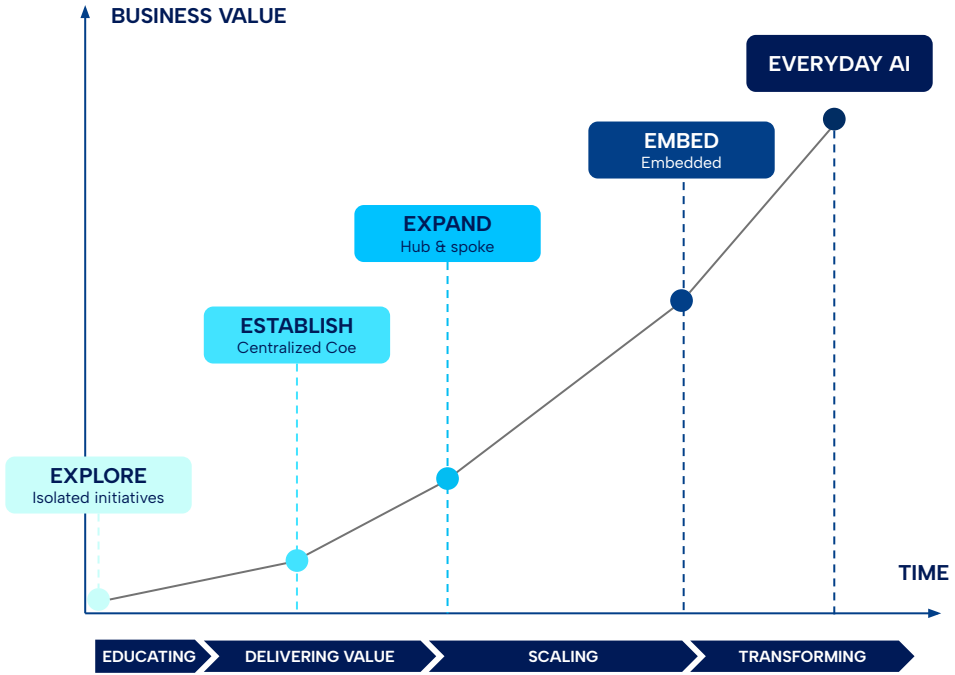
Pan-European Company

FINANCIAL TIMES

Recognized Leader

2nd Fastest Growing Company in Switzerland; Financial Times, 2023

We guide companies at any stage of their AI journey



EVERYDAY AI

AI is a competitive advantage within the company and allows it to re-create itself constantly.

EMBED

AI is integrated into various business processes. Constant conversion of empirical value into continuous value.

EXPAND

The value of AI has been proven. Scale-up by directly co-creating with subject matter expertise within the business.

ESTABLISH

Centre of excellence to explore AI. Professionalisation within the organisation, with a clear role matrix.

EXPLORE

Isolated initiatives in different parts of the organisation. First experience with AI, no strategic goal or objectives.

How Visium applied different type of AI across industries

GenAI Automated formulation

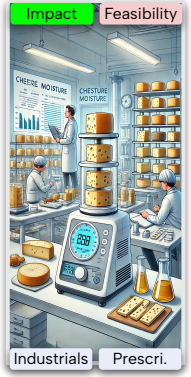
Created GenAI model that **automatically generates new chemical formulas** that match a specific taste or scent. The AI solution now allows the client to perform more successful creative explorations, to respond faster to clients' requests with innovative and personalized fragrances & flavors. **Reduced time-to-market of 50%** creating new business model for smaller clients.

AI-assisted document classification

Leveraged ML algorithms to analyze unstructured data and **automatically extract and classify information from +220'000 partially handwritten documents in various languages.** Significantly reduced time spent accessing historical insights, with an accuracy rate of 97%, and extraction time reduced by 80%. Reduced friction during company reorganization and digitization.

Firestop scene similarity

Client manufactures firestops to impede spread of fire and smoke. Determining appropriate firestop is complex due to many dependencies (e.g. type of wall, pipe type) making task highly technical, time-consuming, and risk-prone. Built **a computer vision AI tool recommending appropriate firestop products given a photo of the scene**, by analysing similarity with past installations.



Cheese moisture optimization

Using AI, **optimize cheese production lines parameters** (set temperature, cooking time, cooking temperature) **to maximize cheese yield by increasing moisture** and reducing its variability (waste reduction) while constraining other parameters (e.g. salt). Production parameters are optimised live depending on real-time lines conditions and tests of cheese quality/quantity.

Prescriptive model for Zinc prices hedging

Zinc spot purchasing can be costly due to potential 10% of monthly price fluctuations. Developed **ML-based model to forecast Zinc prices for upcoming months giving** accurate and reliable view of future price movements. Equipped our client with an improved, reliable and stable **strategy to make data-driven informed purchasing**, reducing cost of acquisition while reducing risk for energy hedging (1.7% opt.).

Next best banking offers recommandation

Developed a **recommender system to suggest the next best offer** based on client needs analysis. It provides sales recommendations for both incoming and targeted outbound calls, offering diverse and relevant services. The algorithm predicts products of interest by analyzing historical behavior of customers with similar characteristics. 20% conversion rate vs 8%

An international but connected team

VISIUM

World-Class Talent

- Graduated from **top institutions**: EPFL, ETH Zurich, UPV, ESADE, Imperial, LBS…
- **65%+** hold advanced degrees in AI, data science, computer science…
- **20+** publications in leading AI and ML journals.

Continuous Learning and Upskilling

- We invest **60+ hours** of training annually per team member on the latest AI advancement, soft skills & leadership
- **77%** engineers are certified in **7** modern AI architectures and cloud solutions

Culture of Collaboration & Innovation

- All team members, regardless of seniority, work closely with clients to **co-create** SOTA* AI solutions.
- **Innovation is in our DNA**: We pioneered the GenAI fragrance formulation with Firmenich in 2018, 3 years ahead of the rise of LLMs like ChatGPT.

Proven Expertise Across Industries

- Each team member has delivered **10+** AI projects on average.
- Our talents are versatile, with **cross-industry** expertise spanning healthcare, finance, chemical, manufacturing, energy, and more.

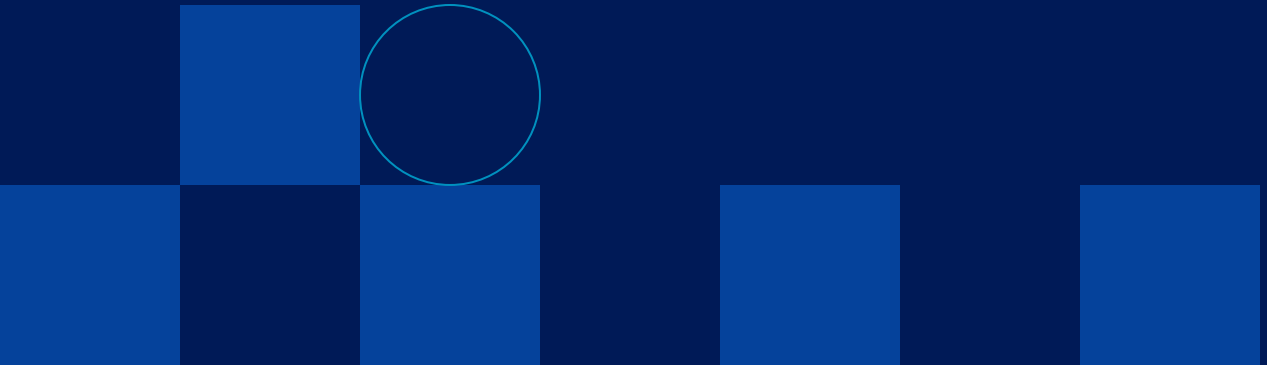
*SOTA – State-of-the-art

Table of contents

VISIUM

INTRODUCTION

What are LLMs and why are they so powerful?



LLMs the Basics

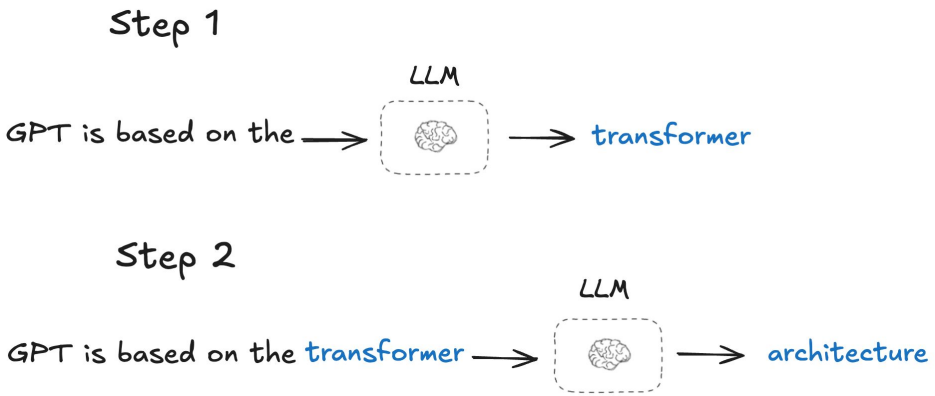
What are LLMs

NLP models trained on vast amounts of text data, capable of understanding and generating human-like language.

Auto-regressive Generation: Produces text by predicting one token at a time, using previous tokens as context

Massive Pretraining: Trained on billions of text examples, enabling strong generalization to new tasks

Zero-shot Learning: Can perform tasks they weren't explicitly trained for, based on natural language instructions



Token-by-token generation: LLMs predict each word based on previous context, building meaning sequentially.

VISIUM

CONCEPTS

When Architecture Meets Big Data

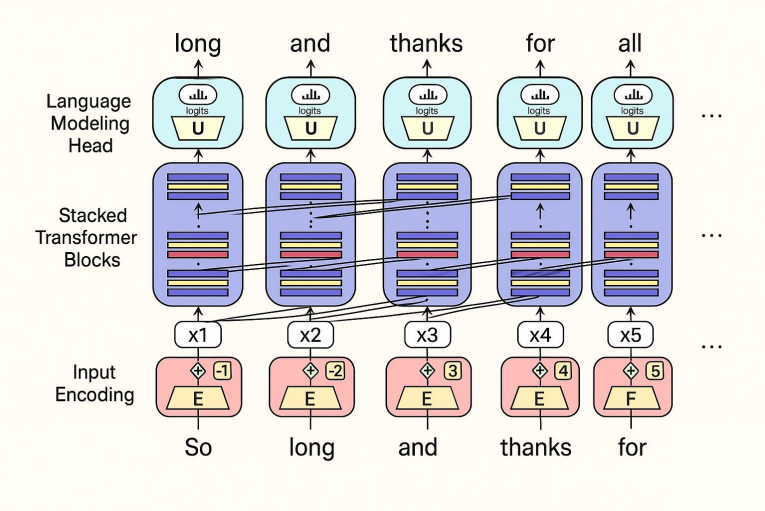
Pretraining

Let's make it simple

- Start with massive text datasets (billions of examples)
- For each sequence of tokens (words):
  - Let the model predict the next word
  - Train the model using an optimization algorithm (gradient descent) to minimize the prediction.

Pretraining on vast amounts of text is what equips llms with the knowledge and capability to perform a wide range of tasks.

Many tasks can be turned into tasks of predicting words!



The transformer architecture enables the model to understand relationships between words across long sequences, making predictions based on full context rather than just adjacent words.

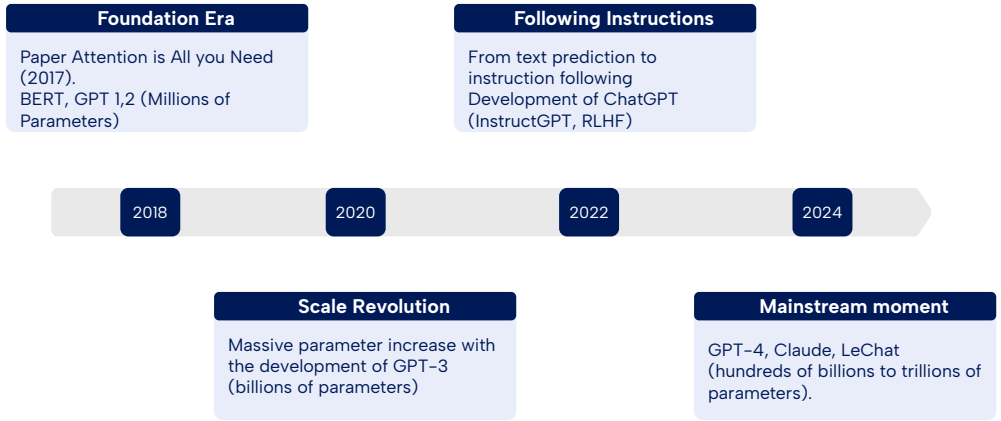
From transformer models to todays LLM’s

From Lab to Chat

The journey from the original transformer paper to ChatGPT represents a transformation from research breakthrough to mainstream AI assistant. Three key innovations made this possible:

Key Developments:

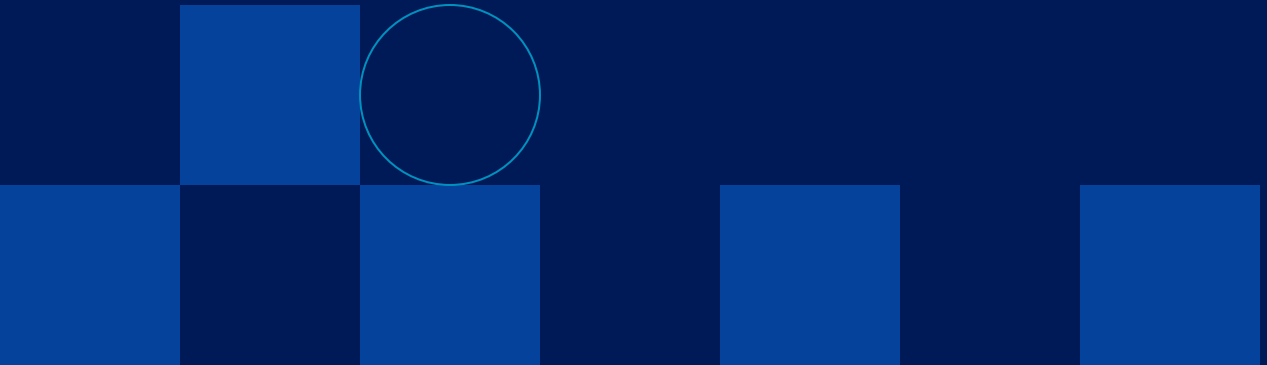
- Massive Scale: From millions to hundreds of billions of parameters
- Instruction Tuning: Models learned to follow specific user requests
- Human Feedback Training: RLHF aligned models with human preferences
- User Experience: From technical interfaces to conversational chat



VISIUM

Illustrative examples

Some examples of LLMs applications and Chatbots

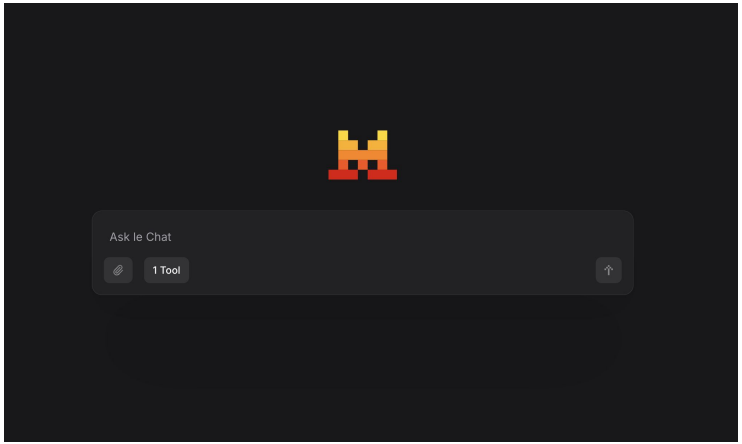


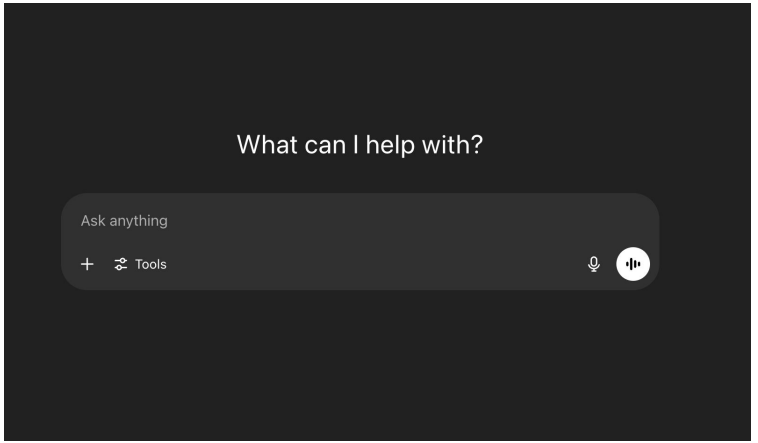
Some practical examples of LLMs and Chatbots

LLaMA: Open and Efficient Foundation Language Models

Hugo Touvron[*], Thibaut Lavril[*], Gautier Izacard[*], Xavier Martinet
Marie-Anne Lachaux, Timothee Lacroix, Baptiste Rozière, Naman Goyal
Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin
Edouard Grave[*], Guillaume Lample[*]

Meta AI

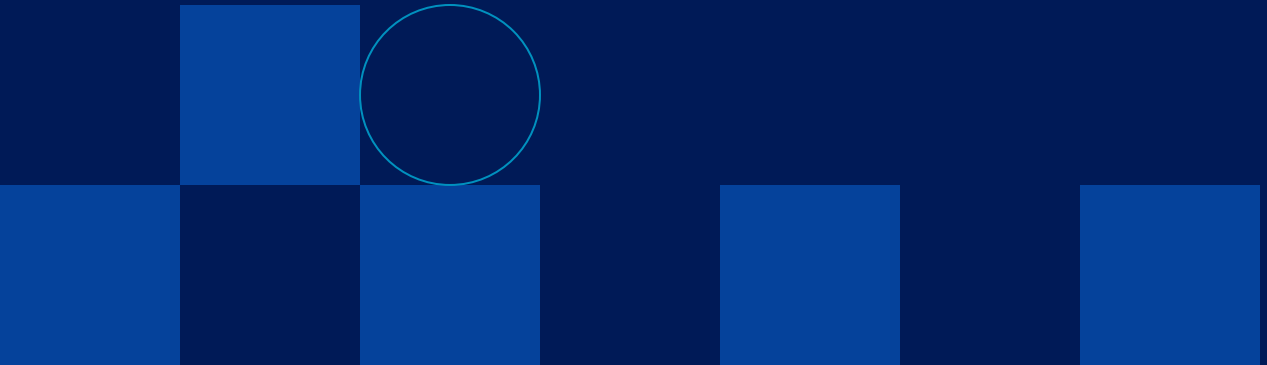




VISIUM

LangChain

Basics of LangChain and comparison with other frameworks



Most popular frameworks for developing LLM Applications and Chatbots

	LangChain	Haystack	LlamaIndex	Semantic Kernel
Ease of Use	🟧	🟩	🟩	🟨
Strengths	Modular and very flexible for all kind of LLM applications	The best tool for production-ready Chatbots and RAGs	Fast and Flexible for search in complex data	Integration with .NET and Microsoft ecosystem
Weaknesses	Documentation, complex for non developers and beginners	Not modular for custom applications	Focus mostly on RAGs	Limited support outside Microsoft ecosystem
Summary	LLM applications and Chatbots. Integrating with APIs, databases, or tools.	Production ready search applications	Document Q&A and retrieval systems on big datasets	Enterprise chatbots with planning and memory

Sources: Visium analysis, BCG X.

LangChain Basics (1/2)

What is LangChain

An open-source framework for building applications powered by Large Language Models (LLMs)

It connects LLMs with external tools, data sources, and memory to build complex, multi-step reasoning systems.

Helps move beyond single-prompt LLM use — enabling agentic behavior (where the system makes decisions, calls tools, and plans actions)

Why use LangChain?

Modular & Flexible — Easily combine LLMs, tools, and data for custom workflows.
Rich Ecosystem — Leverage a growing set of integrations, tools, and community support.

LangChain Basics (2/2)

The *"Hello World"* of LLMs

Initialize a chat model using **ChatOpenAI**:

- Various parameters can be set, such as model and temperature.

Use **invoke** on a list of messages to call the model with:

- **SystemMessage** to pass a system prompt
- **HumanMessage** to pass user messages
- **AIMessage** to pass LLM messages
- **ToolMessage** to pass tool call results
- ...

```python
from langchain_core.messages import HumanMessage, SystemMessage
from langchain_openai import ChatOpenAI

base_model = ChatOpenAI(model=LLM_MODEL, temperature=LLM_TEMPERATURE)

response = base_model.invoke(
    [
        SystemMessage(BASE_PROMPT),
        HumanMessage(request),
    ]
)

```

VISIUM

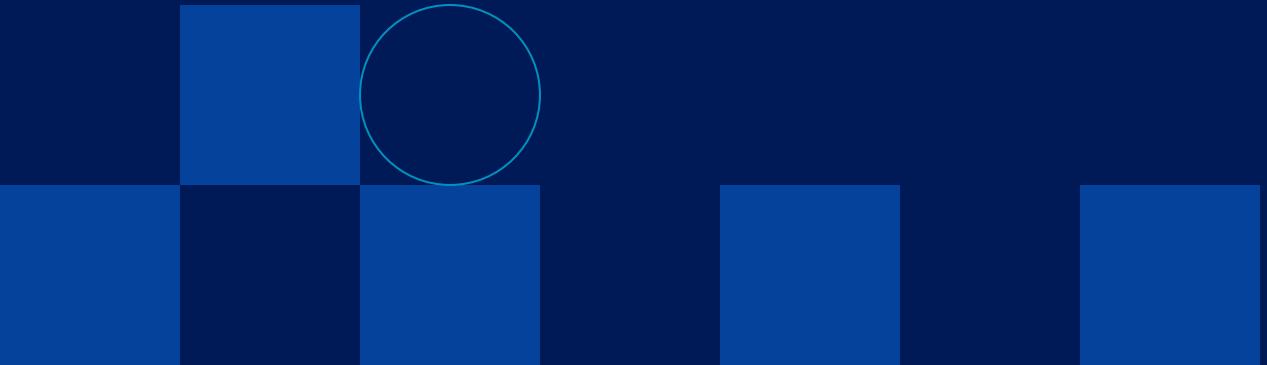
Financial Analyst ChatBot

Live coding session of a simple financial analyst chatbot using LangChain

VISIUM

Hands–On

Intro to LangChain & an implementation of a simple multi agent system



Hands on

Github repo:
https://github.com/VisiumCH/pydata-agentic-2025

OpenAI API Keys: https://pastebin.com/qiZZshyy

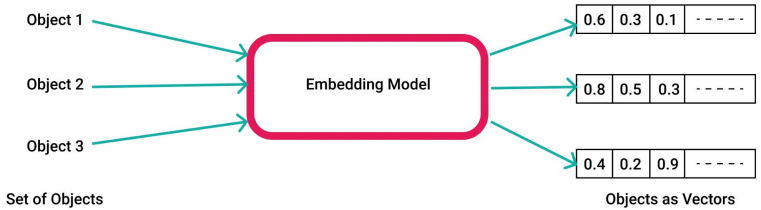
Vector Databases and RAG

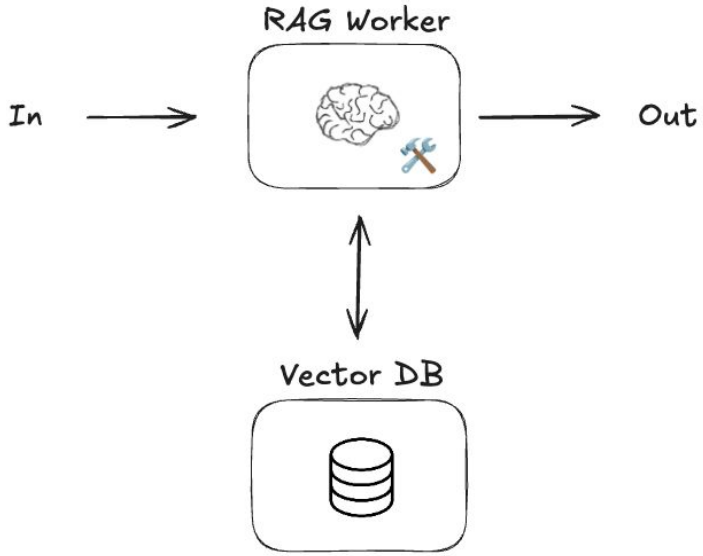
Storing and retrieving documents

Documents can be stored as embeddings in a Vector Database.

In a **Retrieval** Augmented **Generation** (RAG) workflow, an LLM has can perform queries to a vector database to retrieve relevant documents.

These documents are returned to the LLM and can be used to generate an answer grounded by the retrieved information.





Agentic RAG

Retrieve documents from a vector database in LangChain

Multiple VectorDB providers are available in LangChain (**Chroma**, Pinecone, FAISS, etc…).

Load the documents to a Vector store where their embeddings will be computed with **OpenAIEmbeddings**.

Use **as_retriever** to create a retrieval system. It can also be customized with various parameters (number of documents to retrieve, distance metrics, confidence…).

```python
from langchain_chroma import Chroma
from langchain_openai import OpenAIEmbeddings

embedding_model = OpenAIEmbeddings(model=EMBEDDING_MODEL)
vector_store = Chroma.from_documents(
    documents=document_chunks, embedding=embedding_model
)

vector_store = initialize_vector_store(documents)
retriever = vector_store.as_retriever(search_kwargs={"k": 3})

```

RAG as a Tool in LangChain

Augmenting LLMs with retrieval

We can use the **retriever** defined previously as a tool in LangChain with the **@tool** decorator.

Augment a model with **bind_tools**, and access the **tool_calls** attribute in the response to see tool queries made during the model call.

In this simple RAG system, document retrieval is performed with the tool, and the model is invoked again with the retrieved documents added to the message list.

```python
from langchain_core.documents import Document
from langchain_core.messages import HumanMessage, SystemMessage, ToolMessage
from langchain_core.tools import tool


@tool
def retrieval(retrieval_query: str) -> list[Document]:
    """Retrieve documents based on a query."""
    return retriever.invoke(retrieval_query)


tools = [retrieval]

rag_model = base_model.bind_tools(tools)

rag_response = rag_model.invoke(
    [
        SystemMessage(RAG_PROMPT),
        HumanMessage(request),
    ]
)

tool_call = rag_response.tool_calls[0]
documents = tool.invoke(tool_call["args"])
documents_str = "\n\n".join([doc.text for doc in documents])

response = base_model.invoke(
    [
        SystemMessage(RAG_PROMPT),
        HumanMessage(request),
        rag_response,
        ToolMessage(content=documents_str, tool_call_id=tool_call["id"]),
    ]
)

```

Hands-On 1 !

Github repo: **bit.ly/amld2025_langchain**

Open the 1st notebook and go through the exercises.

You will use LangChain to implement a RAG solution for financial advising using a database of Bloomberg financial news articles.

Remember:

- Use **ChatOpenAI** to create a Chat model
- Use **model.invoke**(messages) to perform calls and queries
- Use **model.bind_tools**(tools) to augment a model

If you are fast, feel free to experiment with model parameters, try your own prompts or tune the retriever...

If you have an OpenAI api key, you can use it for the workshop. Running today's workshop shouldn't cost more than 0.1$

If you don't have an OpenAI api key, use one from pastebin.com/iSE2DS30 (you can only use the **gpt-4o-mini** and **text-embedding-3-small** models though). Please don't spam these too much as the credit limit is shared and they are on my account :)

VISIUM

Hands-On 2

Building a Multi-Agent System

Orchestrator Workflow with RAG for Financial Advising

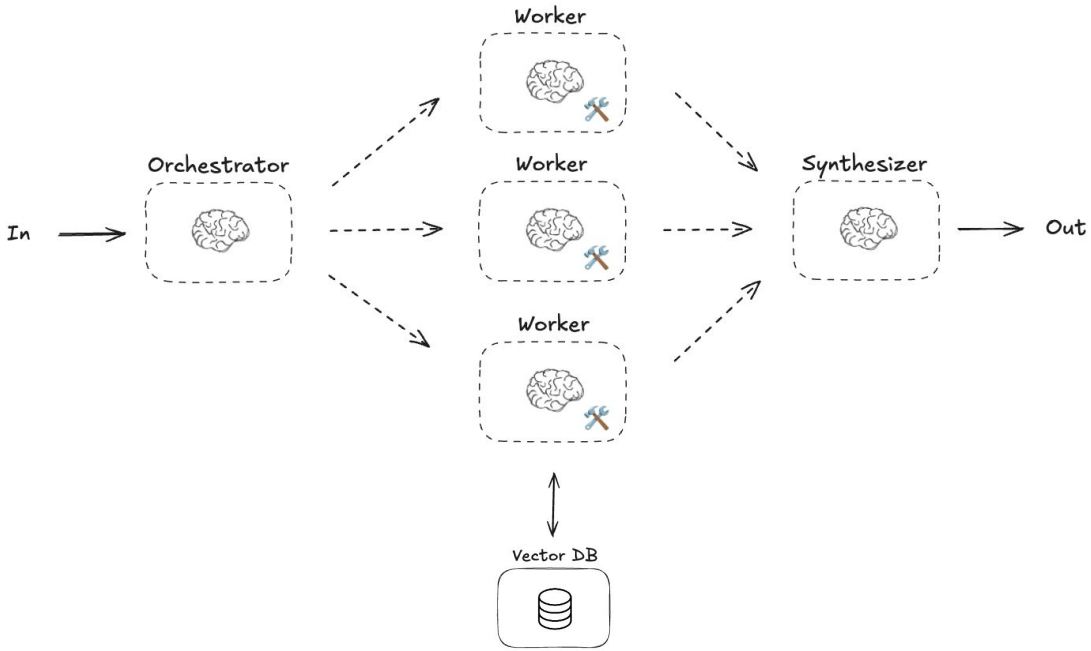
Leveraging dynamic workflow and retrieval

In this second part, you will use LangGraph to implement an orchestrator workflow for financial advising with the following LLMs:

Client Interface Agent (CIA)
Receives client requests and plans research tasks for the research agents.

Bloomberg Research Agent (BRA)
Conducts specific research tasks with retrieval from the Bloomberg news database.

Research Synthesis Agent (RSA)
Synthesizes research results into final recommendations



LangGraph to build workflows

Shaping the problem as graph

LangGraph is an **AI agent framework** developed by LangChain to build **complex generative AI agent workflows**.

It uses customizable **state** classes to manipulate and share information between agents during a workflow execution.

Graph nodes are implemented as functions that receives a state, perform operations, and return updates to the state for the following node.

After defining the nodes and edges, a **StateGraph** can be compiled and run.

```python
import operator
from dataclasses import dataclass, field
from typing import Annotated

from langchain_core.messages import BaseMessage
from langgraph.graph import StateGraph
from langgraph.graph.message import add_messages


@dataclass(kw_only=True)
class WorkflowState:
    """Graph state tracking for the financial analysis workflow."""

    messages: Annotated[list[BaseMessage], add_messages] = field(default_factory=list)
    analyses: Annotated[list[str], operator.add] = field(default_factory=list)


graph_builder = StateGraph(WorkflowState)

graph_builder.set_entry_point("starting_node")

graph_builder.add_node("starting_node", action)
graph_builder.add_node("some_node", other_action)

# The edges are defined by the actions (commands) !

app = graph_builder.compile()

```

LLMs with Structured Output

Ensuring consistency

You can use **Pydantic** in combination with LangChainto to define **output structures** for your LLMs.

The LLM will receive information about your **custom data structure**, and LangChain will parse its response to it, making it easy to obtain consistent outputs and alleviating the need to define it in the prompt.

```python
from pydantic import BaseModel, Field


class ResearchTask(BaseModel):
    """Task for the financial analysis workflow."""

    topic: str = Field(description="Topic of the research task.")
    description: str = Field(
        description="Brief description of the task and its objectives."
    )


class OrchestratorDecision(BaseModel):
    """List of research tasks for the financial analysis workflow."""

    response: str = Field(description="Rationale for the decision and research tasks.")
    in_scope: bool = Field(
        description="Wether the client request is in scope for the financial analysis."
    )
    research_tasks: list[ResearchTask] | None = Field(
        description="List of research tasks to be completed."
    )


orchestrator_model = base_model.with_structured_output(OrchestratorDecision)

```

Defining Edges with Command

Communication flow directly from the nodes

There are multiple ways to define your communication flow using LangGraph. Previously, edges were defined in a similar manner to nodes during the graph building.

The most recent approach (released in December 2024) is to use **Command**.

With **Command**, you can define both the state updates (**update**) and the following edge (**goto**) directly from the return of the node functions.

For the graph to compile properly, remember to indicate the possible next nodes in the function type hints.

```python
from typing import Literal

from langchain_core.messages import SystemMessage
from langgraph.constants import Send
from langgraph.graph import END
from langgraph.types import Command

orchestrator_model = base_model.with_structured_output(OrchestratorDecision)


def orchestrator_node(state: WorkflowState) -> Command[Literal["research", END]]:
    """Orchestrator that generates a plan for the report."""
    orchestrator_output = orchestrator_model.invoke(
        [
            SystemMessage(CIA_PROMPT),
            *state.messages,
        ]
    )

    return Command(
        update={"messages": orchestrator_output.response},
        goto=[Send("research", task) for task in orchestrator_output.research_tasks]
        if orchestrator_output.in_scope
        else END,
    )

```

VISIUM

Conclusion

A few departing words

What's next?

Some food for thought

Workshop summary

During this workshop, we:

- Explored the basics of **LangChain** and **LangGraph**
- Implemented an agentic workflow to solve complex tasks

Key takeaways

Some points to keep in mind when building your own workflows:

- You can augment your LLM with tools
- You can easily create LLMs projects with LangChain and LangGraph

To go further...

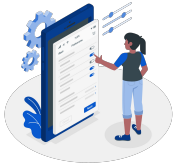
Exciting times ahead:

- Frameworks are still young and evolving fast. Agentic AI is becoming more **accessible** every day
- RAGs
- Feedback loops, more complex workflows

Thank you for having us!

Follow these simple steps to kickstart your career at Visium!

Our recruitment process gives us the possibility to know you are the right fit



Submit your application online

Meet with our Talent Acquisition team to see if we're a match

Show us your Competency skill set

Work on a use case to demonstrate your expertise

Celebrate — you're now a Visiumee!

Apply today and let’s build a data-drive future together!

https://www.visium.ch/join-us/

VISIUM

Contact Info

WEBSITE

www.visium.com

PHONE

+41 21 691 55 30 [Lausanne]
+41 44 201 00 09 [Zürich]

EMAIL

contact@visium.com

alejandro.bonell@visium.com

lucas.eckes@visium.com

younes.moussaif@visium.com

ADDRESS

Unlimitrust Campus, 1008 Prilly

Technopark Zürich, 8005 Zürich

Sources and further readings

- Building effective agents (Anthropic)
- Building Effective Agents with LangGraph
- Generative Agents: Interactive Simulacra of Human Behavior
- The Virtual Lab: AI Agents Design New SARS-CoV-2 Nanobodies with Experimental Validation
- Terminal Velocity – The First Novel Written by Autonomous AI Agents