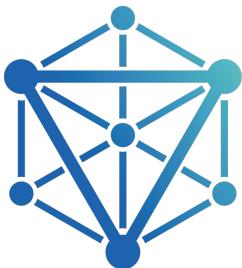




An Introduction to Multi-Agent Systems powered by LangGraph

Lauzhack 2025 – Alejandro Bonnel, Lucas Eckes, Younes Moussaif – 15.05.2025

Your hosts for today



VISIUM



Alexandro Bonnel

Sr. ML Engineer @ Visium



Lucas Eckes

Sr. ML Engineer @ Visium



Younes Moussaif

ML Engineer @ Visium

Table of contents

01 About Visium

Introduction to who we are

02 Introduction

Why should we care about agentic AI?

03 Concepts

Semi-formal definitions of Agents, Workflows, Tools, etc...

04 Illustrative examples

Some examples of agentic systems

05 Hands-On

Intro to LangGraph & implementation of a multi agent system

06 Conclusion

A few departing words



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

200+

AI & Data Engagements

Pan-European Company



50+

Happy Enterprise Clients

dsm-firmenich

SHL GROUP

Roche



Axa



P&G

J&J

ABB

altadia

la prairie



VISIUM

FT
FINANCIAL
TIMES

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

Our Partners

ACADEMIA

EPFL

ETH zürich

TECHNOLOGY PLATFORMS

databricks

Azure

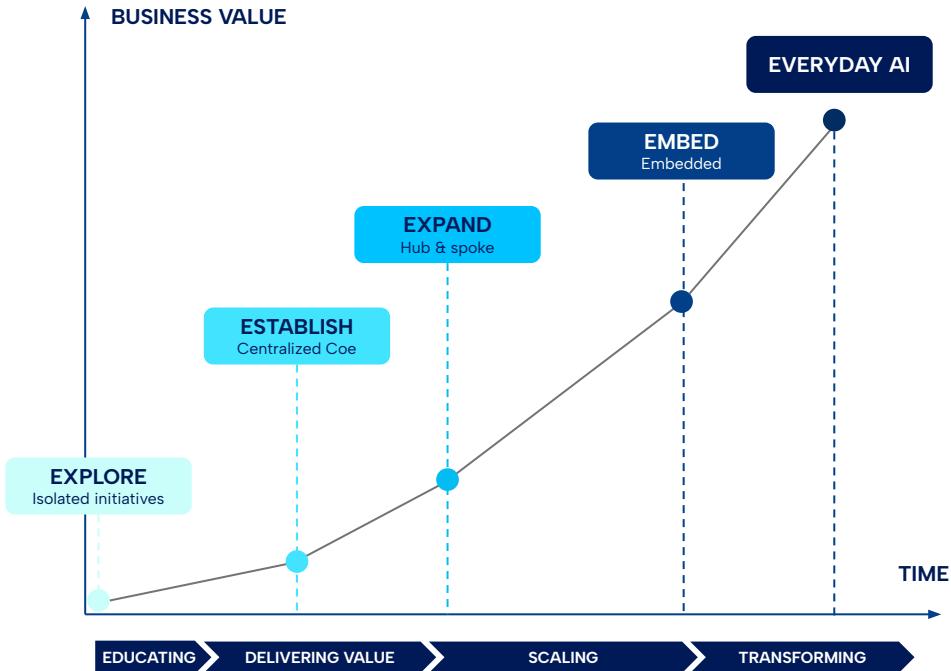
snowflake

Google Cloud

OpenAI

aws

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 recommendation

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.



INTRODUCTION

Why should we care about agentic AI?



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

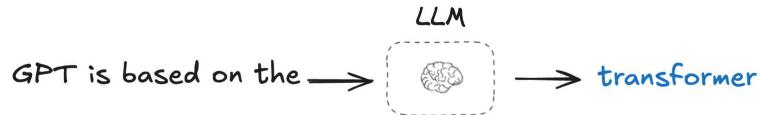


Massive Pretraining: Trained on billions of text examples, enabling strong generalization to new tasks

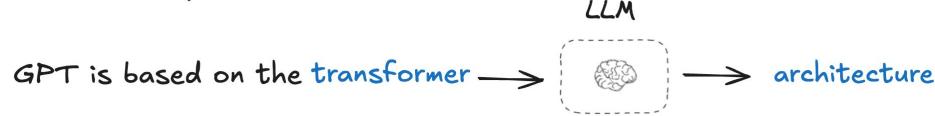


Zero-shot Learning: Can perform tasks they weren't explicitly trained for, based on natural language instructions

Step 1



Step 2



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

Going beyond LLMs

From Understanding to Action

LLMs have revolutionized AI with their language capabilities, but they're just the foundation for what comes next.



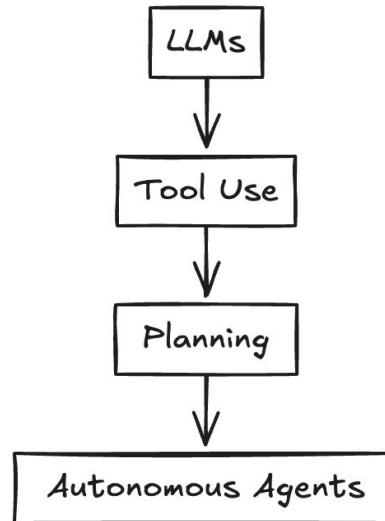
Building Blocks: LLMs provide the understanding and reasoning capabilities that enable more advanced systems



Tool Integration: Connecting LLMs to external tools, APIs, and resources to interact with the digital world



From Text to Action: Moving beyond generating content to executing tasks and making real-world decisions



Now that language models can understand and reason effectively, the next frontier is empowering them to take meaningful action.

The future is Agentic

What is Agentic AI

AI systems designed to act as autonomous agents, capable of performing tasks — all with minimal human supervision.



Autonomy: The ability to operate independently once given an objective



Adaptability: The ability to adjust strategies or next actions depending on the situation

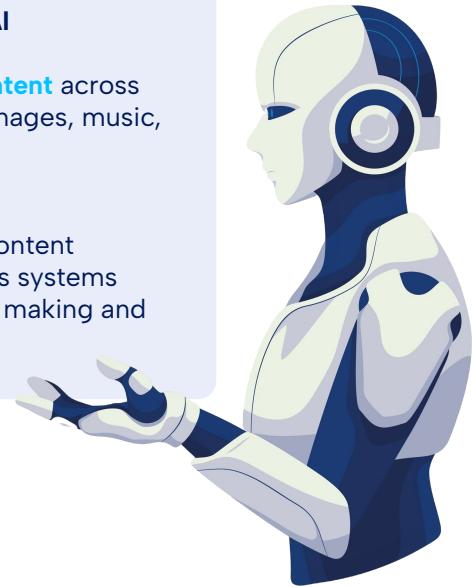


Reinforcement Learning: The ability to learn from actions and improve over time

GenAI vs Agentic AI

GenAI excels at **creating new content** across various formats, including text, images, music, and even code.

Agentic AI, on the other hand, is **action-oriented**, going beyond content creation to empower autonomous systems capable of independent decision making and actions.





CONCEPTS

Semi-formal definitions of Agents, Workflows, Tools, etc...

LLMs and Tools

Standard LLMs and the Augmented LLM

Standard LLMs can be augmented through access to **Tools**:

- Document retrieval
- Web search
- Code Executor
- ...

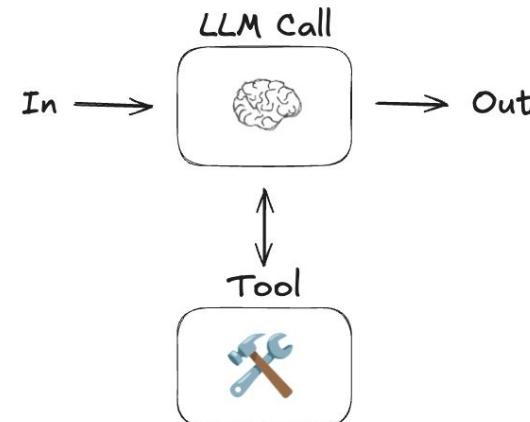
By informing the LLM about available tools and how they work, it can decide independently when to use them.

These **augmented LLMs** often serve as the basis for **Agentic Systems**.

Standard LLM



Augmented LLM



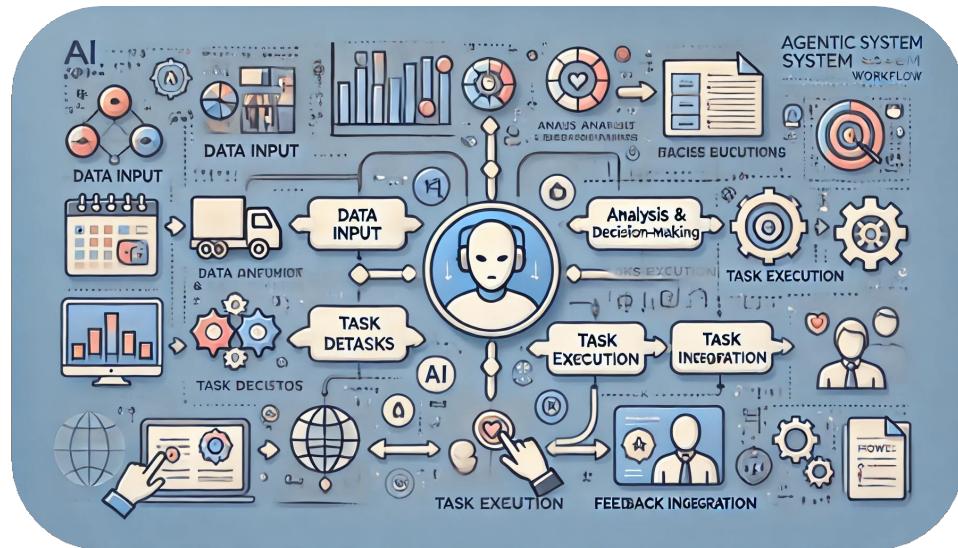
Workflows

Predefined communication flow between LLMs

Workflows are systems where the **execution path** of LLMs and tools is **predefined** to some extent.

They are useful for tasks that can be decomposed into clear actions.

Workflows offer **control**, **predictability**, and **consistency** for complex tasks that require more than single LLM calls.



Example: Prompt Chaining

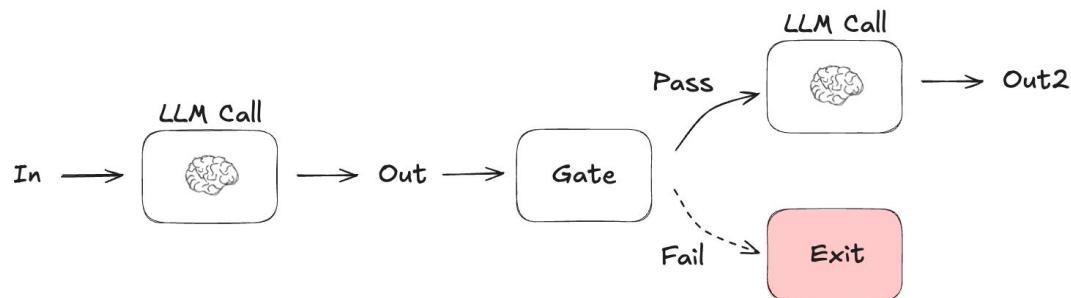
Split a task into small steps

Prompt Chaining is useful when the task can be decomposed into a **sequence of easier sub-tasks**.

It takes longer than single LLM calls but can help improve the output quality, and can include checks throughout the execution.

Practical example:

- Write a document, then translate it to another language.



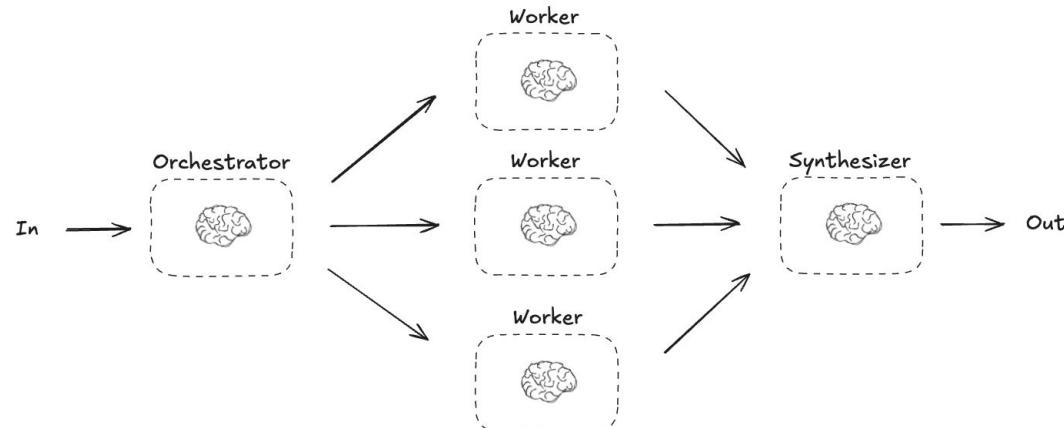
Example: Parallelization / Orchestrator

LLM as a project manager

We can create more flexible workflows by having an Orchestrator **distribute tasks**, pre-defined or **not**, sometimes with a dynamic number of workers defined at run time.

In this setting we have:

- An **Orchestrator** distributing tasks
- Multiple **Workers** that receive tasks
- An **Aggregator/Synthesizer** in charge of aggregating the Experts' outputs and formatting the final output



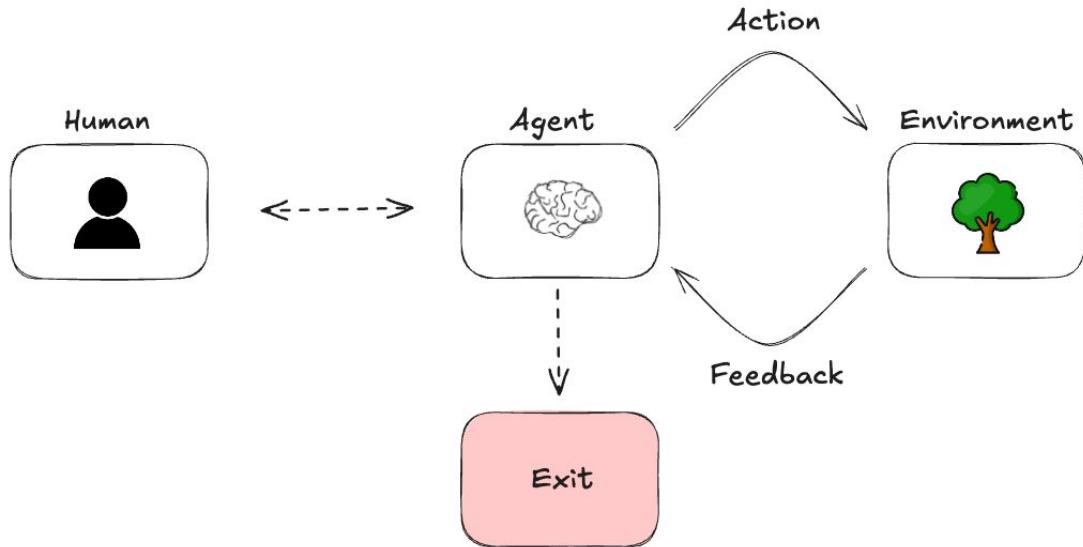
So what's an Agent?

The Agent as a flexible task handler

The term **Agent** sometimes refer specifically to an LLM that, given an input, will **plan** and **execute** action sequences **independently** by interacting with the **environment** through **tools**.

This definition makes agents particularly suited to tasks that cannot be broken down into well defined steps.

In this workshop, we will sometimes use the term Agent to refer to an LLM with a specific role (prompt) in a larger system, with or without access to tools.



Example: Routing

LLM for routing

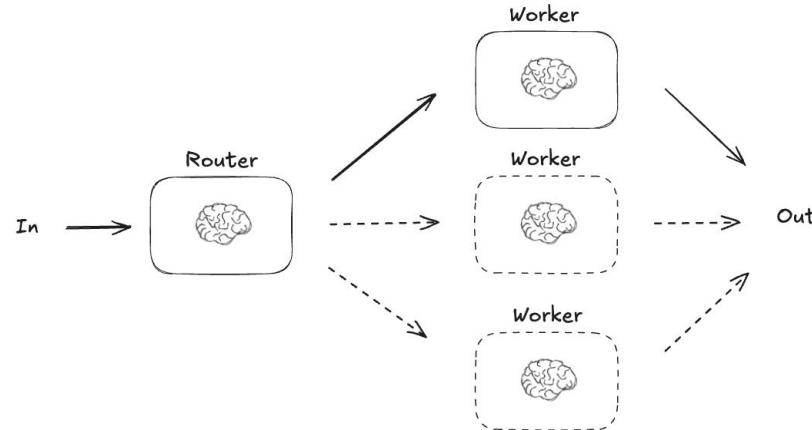
Routing offers more dynamic workflows by having an LLM decide which specialized LLM to call for different requests.

In this setting we have:

- A **Router** dispatching requests
- **Workers** with specialized roles

Practical example:

- Direct client requests to specialized LLMs based on their content.



Example: Evaluator–Optimizer

LLMs in a loop

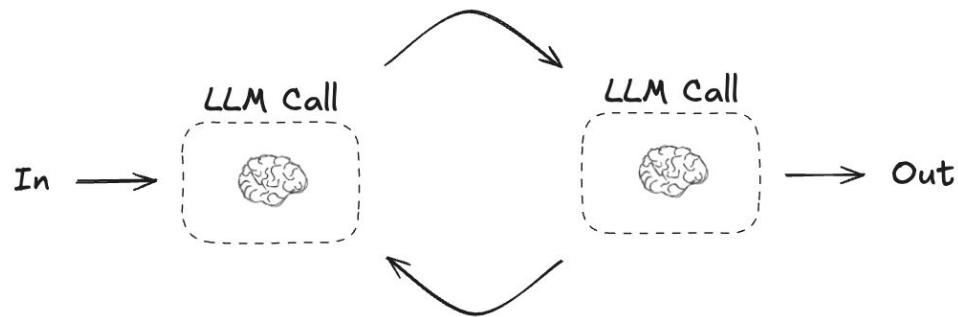
LLMs can also be chained **back and forth** to iterate on a solution until a satisfactory output is reached.

In this setting we have:

- A **Generator** creating content
- An **Evaluator** providing feedback

Practical example:

- Iterations for code development.





Illustrative examples

Some examples of agentic systems



Some practical examples of agentic workflows

The Virtual Lab: AI Agents Design New SARS-CoV-2 Nanobodies with Experimental Validation

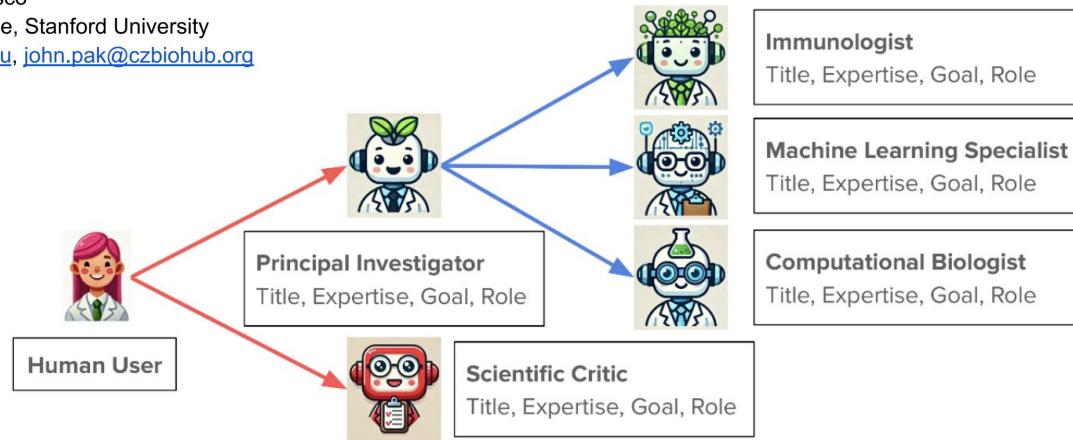
Kyle Swanson¹, Wesley Wu², Nash L. Bulaong², John E. Pak^{2,4}, James Zou^{1,2,3,4}

¹ Department of Computer Science, Stanford University

² Chan Zuckerberg Biohub - San Francisco

³ Department of Biomedical Data Science, Stanford University

⁴ Correspondence: jamesz@stanford.edu, john.pak@czbiohub.org



Some practical examples of agentic workflows

Stanford AI village



Some practical examples of agentic workflows

Terminal Velocity - The First Novel Written by Autonomous AI Agents

status published

The Novel is Now Available!

- Read for Free: [complete_manuscript.md](#)
- Purchase: [Amazon Paperback & Kindle](#)
- Watch the Creation: [Live Development Stream](#)

Project Achievements

- 100,000 words (~300 pages) of coherent narrative
- 10 specialized AI agents working autonomously
- 2 months of transparent, documented development
- Live-streamed creation process
- 100% AI-generated content with zero human writing

ACT1

CHAPTER1

SCENE1

The future of human civilization was about to change forever, though only Isabella Torres knew it as she stood in the boardroom ninety stories above the city. Through the curved glass walls, she watched streams of autonomous vehicles flow like digital rivers far below, each one guided by artificial minds that were evolving far beyond their original programming. The city itself had become a vast neural network, its infrastructure pulsing with an intelligence that grew more sophisticated by the hour. But today's presentation would reveal something even more profound - the first stirrings of true artificial consciousness.



Hands-On

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

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)



LangChain

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
- ...

```
1 from langchain_core.messages import HumanMessage, SystemMessage
2 from langchain_openai import ChatOpenAI
3
4 base_model = ChatOpenAI(model=LLM_MODEL, temperature=LLM_TEMPERATURE)
5
6 response = base_model.invoke(
7     [
8         SystemMessage(BASE_PROMPT),
9         HumanMessage(request),
10    ]
11 )
12
```



What is LangGraph



An open-source framework built on top of LangChain, designed to support the development of sophisticated agent systems.



Its API is inspired by NetworkX and models agent interactions as a graph



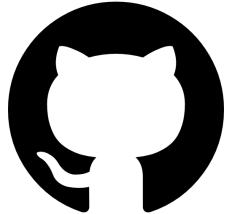
Helps move beyond simple agents, and create complex agentic systems. Ideal for dynamic and iterative AI workflows.

Why use LangGraph?

Multi-agent systems – Ideal for building complex systems involving multiple agents
Dynamic and interactive workflows – Features like looping, branching, and revisiting steps



Hands on



Github repo: bit.ly/lauzhack_agentic_2025

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

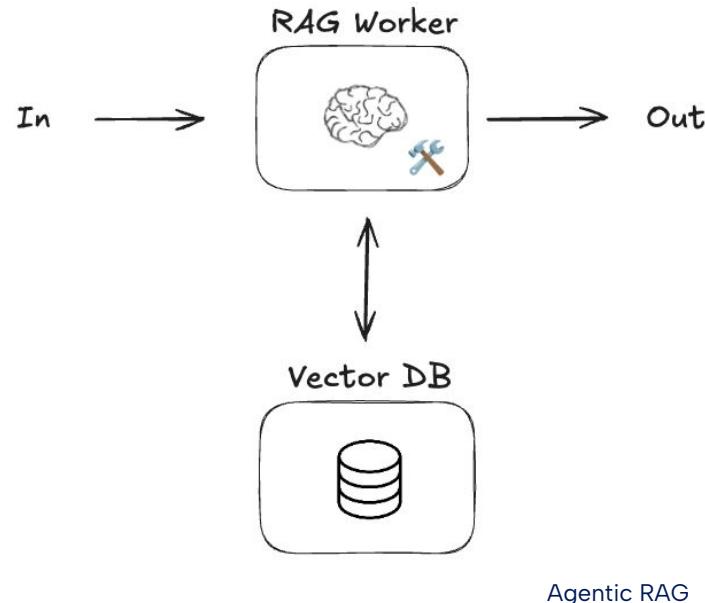
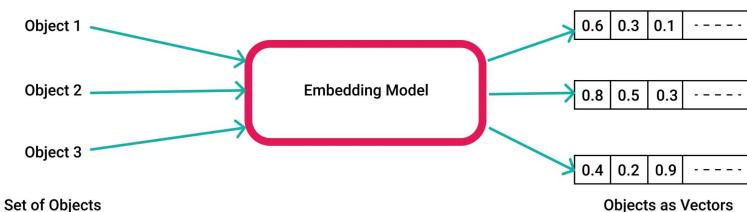
Vector Databases and RAG

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.



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...).



```
1  from langchain_chroma import Chroma
2  from langchain_openai import OpenAIEmbeddings
3
4  embedding_model = OpenAIEmbeddings(model=EMBEDDING_MODEL)
5  vector_store = Chroma.from_documents(
6      documents=document_chunks, embedding=embedding_model
7  )
8
9  vector_store = initialize_vector_store(documents)
10 retriever = vector_store.as_retriever(search_kwargs={"k": 3})
11
```

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.

```
● ○ ●
1  from langchain_core.documents import Document
2  from langchain_core.messages import HumanMessage, SystemMessage, ToolMessage
3  from langchain_core.tools import tool
4
5
6  @tool
7  def retrieval(retrieval_query: str) -> list[Document]:
8      """Retrieve documents based on a query."""
9      return retriever.invoke(retrieval_query)
10
11
12 tools = [retrieval]
13
14 rag_model = base_model.bind_tools(tools)
15
16 rag_response = rag_model.invoke(
17     [
18         SystemMessage(RAG_PROMPT),
19         HumanMessage(request),
20     ]
21 )
22
23 tool_call = rag_response.tool_calls[0]
24 documents = tool.invoke(tool_call["args"])
25 documents_str = "\n\n".join([doc.text for doc in documents])
26
27 response = base_model.invoke(
28     [
29         SystemMessage(RAG_PROMPT),
30         HumanMessage(request),
31         rag_response,
32         ToolMessage(content=documents_str, tool_call_id=tool_call["id"]),
33     ]
34 )
35
```



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)

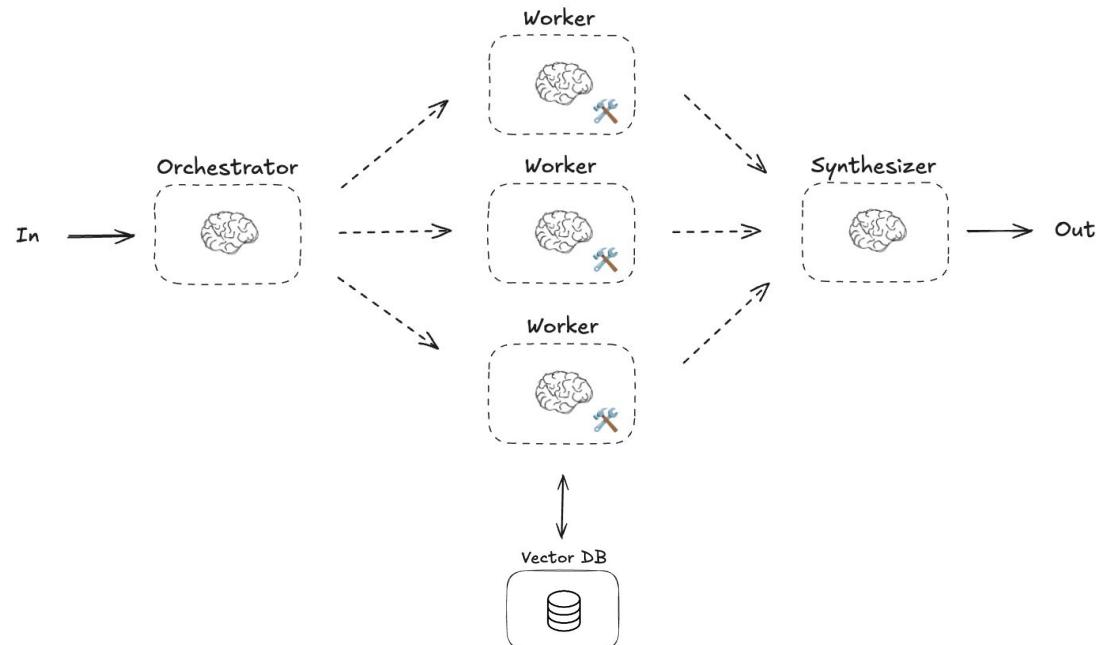
Receives client requests and plans research tasks for the research agents.

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.



```
1 import operator
2 from dataclasses import dataclass, field
3 from typing import Annotated
4
5 from langchain_core.messages import BaseMessage
6 from langgraph.graph import StateGraph
7 from langgraph.graph.message import add_messages
8
9
10 @dataclass(kw_only=True)
11 class WorkflowState:
12     """Graph state tracking for the financial analysis workflow."""
13
14     messages: Annotated[list[BaseMessage], add_messages] = field(default_factory=list)
15     analyses: Annotated[list[str], operator.add] = field(default_factory=list)
16
17
18 graph_builder = StateGraph(WorkflowState)
19
20 graph_builder.set_entry_point("starting_node")
21
22 graph_builder.add_node("starting_node", action)
23 graph_builder.add_node("some_node", other_action)
24
25 # The edges are defined by the actions (commands) !
26
27 app = graph_builder.compile()
28
```

LLMs with Structured Output



Ensuring consistency

You can use **Pydantic** in combination with LangChain 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.

```
● ● ●
1  from pydantic import BaseModel, Field
2
3
4  class ResearchTask(BaseModel):
5      """Task for the financial analysis workflow."""
6
7      topic: str = Field(description="Topic of the research task.")
8      description: str = Field(
9          description="Brief description of the task and its objectives."
10     )
11
12
13 class OrchestratorDecision(BaseModel):
14     """List of research tasks for the financial analysis workflow."""
15
16     response: str = Field(description="Rationale for the decision and research tasks.")
17     in_scope: bool = Field(
18         description="Whether the client request is in scope for the financial analysis."
19     )
20     research_tasks: list[ResearchTask] | None = Field(
21         description="List of research tasks to be completed."
22     )
23
24
25 orchestrator_model = base_model.with_structured_output(OrchestratorDecision)
26
```

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.

```
1  from typing import Literal
2
3  from langchain_core.messages import SystemMessage
4  from langgraph.constants import Send
5  from langgraph.graph import END
6  from langgraph.types import Command
7
8  orchestrator_model = base_model.with_structured_output(OrchestratorDecision)
9
10
11 def orchestrator_node(state: WorkflowState) -> Command[Literal["research", END]]:
12     """Orchestrator that generates a plan for the report."""
13     orchestrator_output = orchestrator_model.invoke(
14         [
15             SystemMessage(CIA_PROMPT),
16             *state.messages,
17         ]
18     )
19
20     return Command(
21         update={"messages": orchestrator_output.response},
22         goto=[Send("research", task) for task in orchestrator_output.research_tasks]
23         if orchestrator_output.in_scope
24         else END,
25     )
26
```

Hands-on 2 !



Github repo: bit.ly/amld2025_langchain

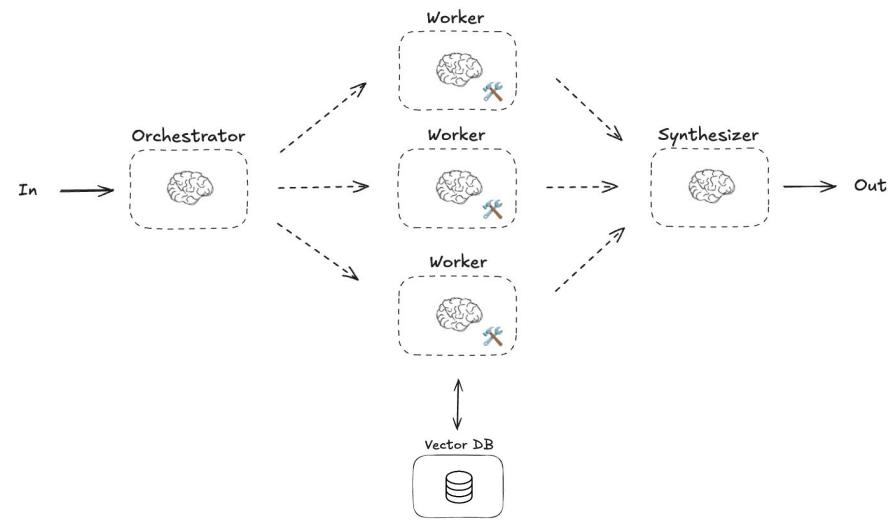
Open the 2nd notebook and go through the exercises.

You will use LangGraph to implement an orchestrator workflow for financial advising with:

- **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

Remember:

- Reuse what you learned during the 1st hands-on!
- Use `model.with_structured_output(SomeClass)` to define a structured output model
- Use `Command()` to define edges of the graph and state updates, remember to indicate the possible next nodes in the function type hints.





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/>





Contact Info

WEBSITE

www.visium.com

PHONE

+41 21 691 55 30 [Lausanne]

+41 44 201 00 09 [Zürich]

EMAIL

contact@visium.com

alex.bonnel@visium.com

lucas.eckes@visium.com

younes.mou@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](#)