Theory + Hands-on

Assignments

Mini project

## Basics

gen AI

generate

learn patterns from huge datasets

80s

expert systems

“if this, then that” logic

2000s

ML DL

2010s

LLM

flexible, creative, conversational

2020s

Agentic AI

* planning
* memory

autonomy

Why now?

* maturity of LLMs
* tool integration
* business demand
* infrastructure
* user expectations

## ML/DL

Tokenisation (turn text → tokens)

Vectorisation with TF-IDF (turn tokens → numbers )

token → weight

TF x IDF

L2 normalize

TF

IDF (inverse document frequency)

cosine similarity

search()

kind: preference episodic semantic

doc A: “apples apples bananas”

doc B: “bananas carrots”

df(apples) = 1

df(bananas) = 2

## Core components

### Perception/Input

“table for two”

“Koramangala”

“tommorrow evening”

understanding

### Reasoning/Planning

sequence

to-do

adaptive (instead of reactive)

### Action Execution & Tools

#### tools

APIs, databases, search engines, custom functions

logging

### Feedback loops

tool outputs

error codes

user confirmation

### Memory

## Guardrails

## Memory

ConversationMemory

ConversationSummaryMemory

ConversationBufferWindowMemory

vector DBs:

Chroma/FAISS/Pinecone

## Architectures

### Reactive

### Delibrative

### Hybrid

ReAct

Tree of Thoughts

reflexion

## Flow Control (Lifecycle)

order (goal) → plan recipe → cook (actions/tools) → taste/adjust (feedback) → write notes (memory)

tools → safety → memory →planning

lab1

project

* agent /
  + \_\_init\_\_.py
  + [tools.py](http://tools.py) # tools (mock) + registry
  + [core.py](http://core.py) # router + arg extraction + orchestrator
* apps /
  + \_\_init\_\_.py
  + demo\_lab1.py

minimum skeleton

hear a goal

decide

extract the inputs

act

respond

lab 2

what:

seatbelt (guardrail)

dashboard (logging)

retry

Why:

Trust & debugging (observability)

Real-world reliability (retry)

Safety by default (gaurdrails)

better user experience (graceful fallbacks)

logging

who: tool\_name

with: sanitize args

when:

what happened: status

CALL/OK/RETRY/FAIL

query: weather in Paris

INFO CALL weather args={‘city’:’paris”}

INFO OK

WARN RETRY weather attempt 1: transient timeout

query: delete database

WARN BLOCKED

argument validation (not a retry case)

query: check weather

router : picks weather

arg extractor city = None

Reply:

lab 3

why:

continuity

personalisation

quality & efficiency

foundation for planning

ShortMemory:

deque   
LongMemory:

vector store

tokenizes text

computes naive TF-IDF-ish vectors

cosine similarity

Safety policies for storage:

redact PII

is\_storable rule

integration points

Write

Read

lab 4

understand

build a plan

execute the steps

keep a trace

use memory

execute\_the\_plan()

dependency passing

step B depends on A

inject upstream=outputs[“A”] into B’s argument

lab 4B (online fetch)

add a live tool

* live\_weather(city) → open-met
* wiki\_summary(topic) →

domains:

allow\_listed (ALLOWED\_HOSTS)

lab 4C (Open AI LLM)

llm\_openai(prompt) → OpenAI Chat Completions via requests

OPENAI\_API\_KEY

default model gpt-40-mini

call\_with\_policies()

## LLM

predicts

1. OPENAI

gpt-4o

1. DeepMind

gemini (PaLM/Bard)

1. Anthropic

Claude

backed by amazon, google

1. Meta

LLaMA

1. Mistral

mistral

1. xAI

grok

1. Cohere

command R

1. Alibaba baidu Huawei (china)

## LangChain

framework for building LLM-powerd apps

prompt management

external tool

memory

decision making

user input

prompt construction

reasoning / decision

tool/memory

output

lab5

LLM + PromptTemplate + LLMChain

input → reasoning → tool/memory → output

llm = ChatOpenAI(model="gpt-3.5-turbo", temperature=0.7)

temperature:

probabilities

“paris” → 0.9

“marseille” → 0.03

2+2

T = 0.1 (low)

T = 1.0 (medium)

most 4, few 3,5

T = 2.0 (high)

banana

| Lab 4 component | LangChain |
| --- | --- |
| [tools.py](http://tools.py) | Tools ( @tool ) |
| manual prompts | PromptTemplate |
| agent.py | Agents (ReAct, Conversational, Tool-using) |
| memory.py | Memory (ConversationBuffermemory, ConversationSummaryMemory) |
| planner.py | Chains( SequentialChain, LLMChain, AgentExecutor) |
| raw openAI API calls | ChatOpenAI/LLM wrappers  ChatOpenAI, ChatAnthropic |
| Orchestrator | LangChain Orchestrator (Agents + Chains + Memory) |

lab 6

Tools & Tool-calling Agents

### Agent Styles

1. Tool-Calling Agent (OpenAi function)
2. ReAct agent (Thought → Action → Observation)

### ReAct Agent

1. Reasons
2. Act choose tool
3. Observes tool’s output
4. Repeat reason & act until final answer

#### Agent Types

1. OPENAI\_FUNCTIONS / OPENAI\_MULTI\_FUNCTIONS
2. zero shot react description

default ReAct agent

general purpose agent

1. chat zero shot react description

prompt is more like a chat

1. structured chat zero shot react description

prompt is more like a chat

structured output (JSON)

1. conversational react description

chat history

1. self ask with search

### prompt engineering

lab8

1. Baseline
2. JSON Policy

split after ‘Final Answer:’

regex-extracts { … }

json.loads

1. Cost aware

business constraints

1. Math
2. Forbid

tool blacklisting

Flow lab\_6

Prompt 1

Trace

* enter new AgentExecutor chain

Zero\_shot react

ReAct prints

Thought model’s private reasoning

Action

Observation

Repeat

Trace walkthrough

lab7

tool-Calling vs ReAct vs conversational agents

single-input tools for ReAct compatibility

ConversationBufferMemory

“single input tools” : ReAct constraint

lab8

prompt engineering

{

‘city’: “chicago”

“weather”

### for execution:

virtual environment

python -m venv venv

bash

source venv/bin/activate

export OPEN\_API\_KEY=”your key”

windows

python -m venv venv

venv\Scripts\activate

setx OPEN\_API\_KEY ”your key”

## LangGraph

AI workflow modeled state machine

state

nodes

(LLM call, tool, router, DB step)

edges

(decides what runs next)

conditional

checkpointers

resume, rollback

interrupts

human approval

concurrency & loops

branch, join, loop

shared state

scenario 1

Single Agent with Tools

A [user input ] —> S (State)

S → P [Planner (LLM) ]

R → yes : search → T1 [Tool, web search]

R → yes : calc → T2 [Tool, cacli]

T1 → M [merge results]

T2 → P

F → O (answer)

lab9

|  | lab8 | lab9 (runtime arch) |
| --- | --- | --- |
| control style | prompt govern behavior | Graph governs behaviour |
| Tool calls | Tool-calling agent | model bound to tools + tool router node |
| reliability |  | deterministic flow |
| extensibility | add/change prompts | add nodes/edges |

scenario 2

conditional loop (goal oriented flow)

scenario 3

multi agent with supervisor

## Multi-Agent Systems

(often LLMs with distinct roles)

shared goal

### Planner & executor

### 



Planner :

{“done”: false, “next\_step:

{“tool” : “get\_weather

{“done”: true, "final\_answer”: “.....”}

Executor : bound to tools

EXEC\_RESULT

LangGraph: handles routing + memory (same thread\_id)

Orchestrator (LangGraph)

### Blackboard (decentralized)

Agent post messages → shared messages

pick up task

## Labs

### day 1

#### lab 1 Reactive chatbot

see the baseline

pure/simple LLM reply

## one API call (not external)

no state

#### lab 2 tool using agent

tool

real data

perception

#### lab 3 planning agent

#### lab 4 memory enabled agent

embeddings

PII patterns

SSN

phone number

email ID

credit card

call me at 331 800 1299

### day 2

#### lab 1 core components & control flow

tools , router

project

* agent /
  + \_\_init\_\_.py
  + [tools.py](http://tools.py) # tools (mock) + registry
  + [core.py](http://core.py) # router + arg extraction + orchestrator
* apps /
  + \_\_init\_\_.py
  + demo\_lab1.py

lifecycle

perceive → route → extract args → act → reply

#### lab 2

lifecycle (with policies)

perceive → route → extract args → guardrails + log + retry → act → observe (logs) → reply

#### lab 3

lifecycle (with policies)

perceive → update short memory → write LongMemory → Retrieve LongMemory → route → extract args → guardrails + log + retry → act → observe (logs) → reply → UpdateShortMemory

#### lab 4

## Parked doubts

internal working of OPEANAI\_FUNCTION (reference lab\_6)