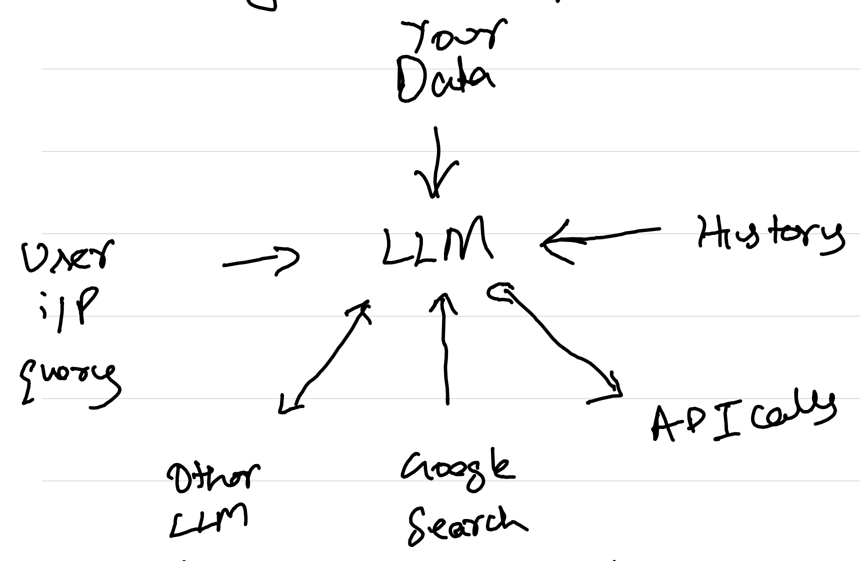
LANGCHAIN is a simplified framework that simplifies the process of building LLM powered Apps

Langchain has
→ tools
→ Abstraction

→ Building an LLM App isn't straightforward



→ To build an intelligent App like above, you will need LangChain.

→ LangChain has same interface across all LLM vendors.

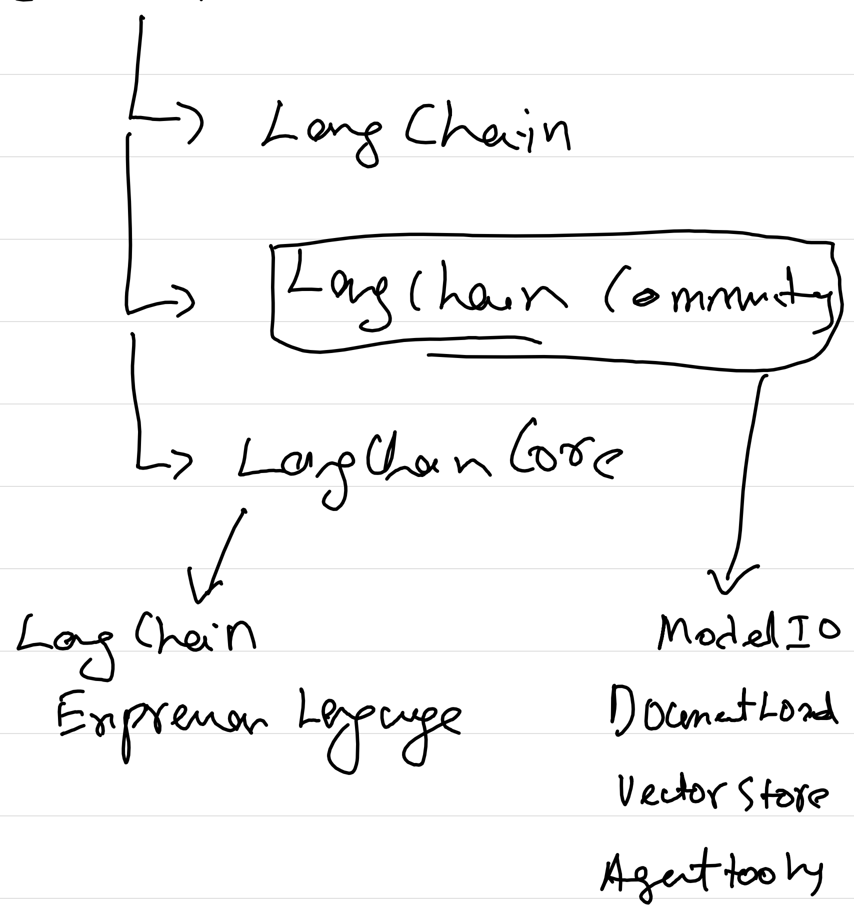
→ LangChain (LC) have prompt management, optimization, serialization

→ LC also has document loaders (.pdf, word, emails, etc)

→ LC supports tool calling, etc

→ LC creates abstraction "chains".

→ LangChain API



→ LangChain Hub
↓
Contains a lot of prompts contributed by Community.

Python 3.11.9

Pipenv install these {
langchain
langchain-openai
langchain-community
langchainhub
black

We used langchain core version of 0.3.61

Also install python-dotenv

Deadline

PrompTemplate:

Prompt is input to LLM (instructions +data)

PromptTemplate helps to translate user inputs, parameters into instruction for LLM

$$\begin{tikzcd}
\text{Dictionary (Key : Value)} \arrow[d] \\
\text{PromptTemplate} \arrow[d] \\
\text{PromptValue} \arrow[d, "\text{i/P to LLM : grounded prompt}"] \\
\text{LLM}
\end{tikzcd}$$

ChatModels:

Provide access to LLM via ChatModels

$$\begin{tikzcd}
\text{List of Messages i/P} \arrow[d] \\
\text{ChatModel} \arrow[d] \\
\text{Message O/P}
\end{tikzcd}$$

→ They also support:
- → tool calling
- → Structured O/P
- → Multimodal i/P o/P.

→ By combining prompt templates, chat models, we can pass data to LLMs.

→ We can combine this with external APIs, pdf readers, file writers, etc in LangChain to properly make an Application

→ We can combine / Chain all these together to make an Application

→ Agent: In LLM world, an agent is a loop that repeatedly reads a conversation, decides what action or tool call to make, feeds the result back into the model until the goal is met

→ LangChain is a Python framework that gives us building blocks (prompt templates, chains, memories, callbacks, tool wrappers) to build LLM apps

→ REACT → It is a prompting pattern where the model (LLM) emits an interleaved $\boxed{\text{thought} \rightarrow \text{Action} \rightarrow \text{Observation}}$ trace

letting it reason, call tools, see result in 1 pass.

Deadline

Task:

→ We want to build a popular person information summarizer.

→ This is how you would do it using LangChain modules.

```
information = '. . . . . . . . .
        some information about a
    popular person'

Summary_template = '''
    given the information: {information}
    about a person, I want you create:
        1. A Short Summary
        2. two intersting facts'''

summary_prompt_template = PromptTemplate(
        input_variables=['information'],
        template= summary_template)
llm = ChatOpenAI(temperature=0,
                 model_name='gpt-3.5-turbo')
Chain = summary_prompt_template | llm
                                  ↓
                             pipe symbol

res = Chain.invoke(input={'information':
                          information)
                              ↓
                            query
print(res)
```

Using OpenSource Models with LangChain

- LangChain can be configured to be used with Ollama, LLAMA 3, Mixtral, etc (local models)
- However the latency will suffer as the models take forever to run.

→ pipenv install langchain-ollama

→ In your code, instead of $\boxed{\text{ChatOpenAI}}$ object instantiation use $\boxed{\text{ChatOllama}}$.

eg: $\boxed{\text{llm = ChatOllama(model='llama3')}}$

Most of the heavy lifting comes from this line

```
chain = summary_prompt_template | llm
        | StrOutputParser
```

↓

here we are using operator overloading to chain

$$\begin{tikzcd} \text{prompt} \arrow[r, "\text{call}"] & \text{LLM} \arrow[d, "\text{format O/P}"] \\ & \text{StrOutput Parser} \end{tikzcd}$$

→ The ice breaker application
  - → gathers information on a given individual (twitter, LinkedIn)
  - → Summarizes it
  - → display to user

↳ Building this App using:

1) Chains
2) Agents
3) Custom Agents
4) Tools, toolkits
5) Output Parsers

Scrapin io signup
Service API
to
scrape LinkedIn profile and
data

→ Get data of linkedIn profiles
as JSON.
→ Pass to LLM if necessary
, make decisions / generate
O/P based on that profile
data.

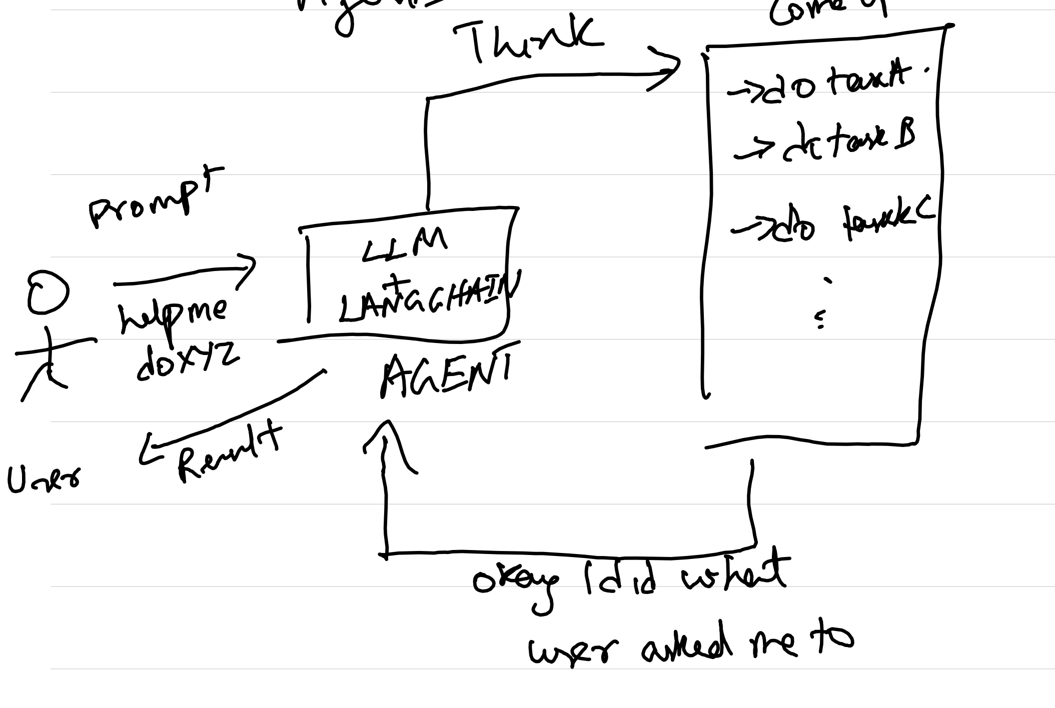
☆
LLM isn't connected to the Web.
They can be considered as a repo
of static information

→ Agent/bot can connect to both
internet through tools.

→ tools are scripts or 3rd party
Services

→ LLM with a prompt can breakdown
big problem into smaller problems
achieve those tasks (using tool
calls).
→ These actions can be then chained
to one another.

→ Agents



→ Chain of Thought is a prompt
engg technique that helps
LLM to think and answer questions

Deadline

→ Agents use LLM + tools to solve what user is asking the agent to solve

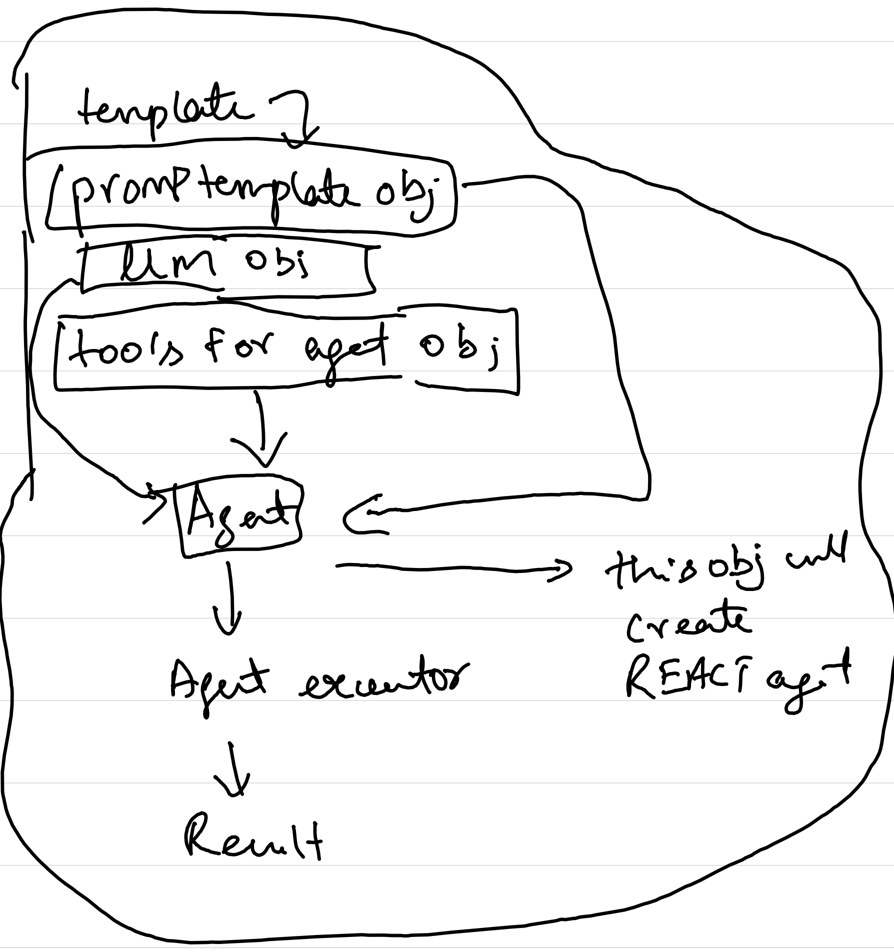
→ Core idea of Agents is to use language model to choose a sequence of tasks to take.

→ Agents use LLM as a reasoning engine to determine which action to take.

→ Build a search agent.
  ↳ Get name from text box
  ↳ search name (REACT agent)
  ↳ get linkedin profile based on URL

We build a search tool
  ↳ linkedin_lookup_tool.py

→ Import from langchain libs:
- prompt template
- tool
- create react agent
- Agent executor
- hub → hub contains premade prompts.



→ Main idea behind icebreaker App
→ Lookup linkedin user name based on user description (return URL)
→ Pull Linkedin profile and Summarize content for icebreaker

Steps:

1 ↳ Search API → List of users in JSON

2 ↳ REACT PROMPT → LLM → filters the list of JSO N
↓
URL

3 ↳ Scrape API → LinkedIn URL → JSON

4 ↳ Summarize JSON using a prompt
↓
→ Brief intro
→ 2 interesting facts

→ Output Parsers:
- LLM usually O/P text
- Output Parsers take text and transform into JSON, CSV, etc

→ PyDantic is a package that helps in schema and data validation.

Icebreaker O/P

→ Summary → str
2 Facts → [str]

↓

If LLM O/P a String with JSON content
↓
the Output Parser is going to wrap it in a Custom result object
↳ Summary.
↳ facts

res.summary → str

res.facts → List[str]

Deadline

→ Creating a full Stack App:
Using FLASK

→ WebPage
→ I/P User name
→ Run button
→ Gives { Summary, Facts, IceBreakers } Shows upon Web page

→ Icebreaker return Summary object rather than Str.

→ index.html
  ↳ text box —i/p→ Python function
  ↳ Button click action
    ↳ Python function
      ↓ call
      icebreaker

→ Response is rendered to UI

→ LangSmith: Service by LangChain offers Observability and monitoring LLMops

↓

After setting up LANGCHAIN_API_KEY

↓ Inside the App we can see

Every LLM call, cost, latency, i/P, O/P of LLM is rendered in the LangSmith Service URL

① What is an Agent ?

Answer: Given a task, agent executes a loop repeatedly until the goal is met { to read state
Pick and run tool
Read results
decide nextsteps

② Explain core components of LangChain REACT agent(LLM, tools, Prompt Structure)

-> LLM: Reasoning engine behind any REACT agent
-> takes instructions in text
-> responds in text
-> Produces thought -> Action -> Observation trace in each step.

-> tools: - List of functions or Callables.
- Agent Picks one of available tools to perform a given action
- Ips are passed by Langchain and O/P observation is returned

-> Prompt Struture:
Set of instructions
↳ { Main task
Available tools
Scratch pad
Few Shot examples
↳ thought
↳ Action
↳ Observation

-> LLM decides
tool Acts
Prompt + Scratchpad } guide the conversation to keep reasoning chain transparent, iterate

Deadline

→ whatere we building?

REACT AgentExecutor from Scratch?