Carnegie Mellon University

Lecture 3:
LLM Application Development

SPRING 2026

MOHAMED FARAG

FARAG@CMU.EDU

AGAI
Applied
Generative AI

Agenda

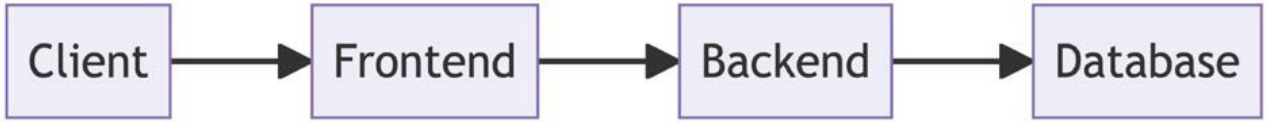
- LLM Application Design
- LLM Application Development
  - Frontend Development: Streamlit
  - Backend Development: LangGraph
- LangChain
- LLM Application Tech Stack
- LangGraph Project Structure and Code Flow
- In-class Demo

LLM Application Development

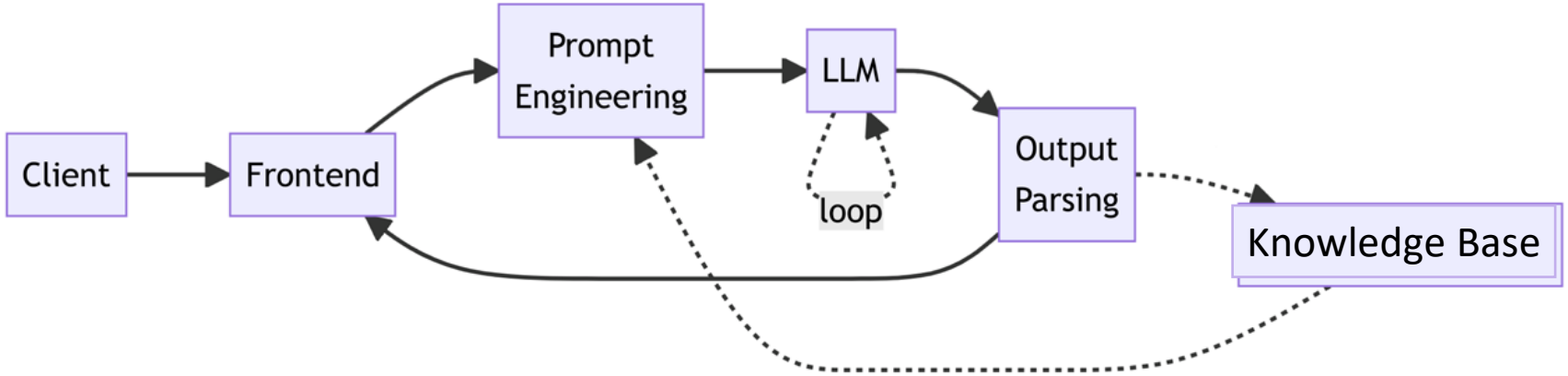
- To leverage LLM benefits and overcome LLM challenges, new kind of Apps is needed.
- This new type is called **LLM Apps.** LLM Apps offer several benefits. For example:
    - Human-like language processing without rigid rules.
    - Personalized responses using previous interactions.
    - Advanced algorithms for complex, multi-step reasoning.
    - Dynamic responses using real-time information or LLM.

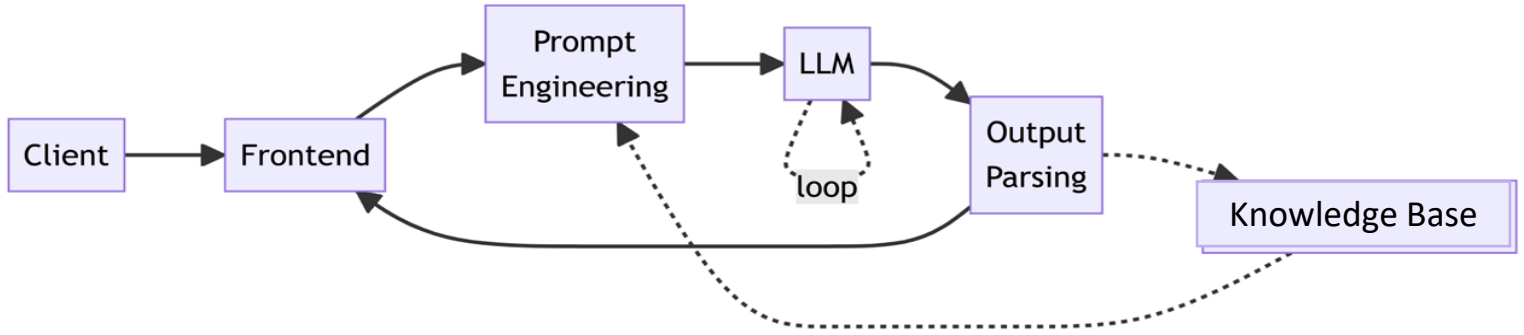
Designing LLM Applications – The General Idea

- **Traditional Apps:**



- **LLM Apps:**





- A **frontend layer** to collect user input as text queries or decisions.
- A **prompt engineering** layer to construct prompts that guide the LLM. Prompt engineering may include prompt parsing as well.
- An **LLM backend** to analyze prompts and produce relevant responses.
- An **output parsing** layer to interpret LLM responses for the application interface.
- Optional integration with **knowledge bases (internal or external)** via APIs, data stores, and reasoning algorithms to augment the LLM's capabilities.

LLM Application Examples

1. Chatbots and virtual assistants
2. Intelligent search engines
3. Automated content creation
4. Question answering
5. Sentiment analysis
6. Text summarization
7. Data analysis & Code Generation

Frontend: Streamlit

What is Streamlit?

- Streamlit is a web application framework for Python web applications.
- Provides built-in methods for handling user inputs like text and dates.
- Enables displaying interactive graphs with popular Python graphing libraries.

Why Streamlit?

- Static visualizations are limited for complex analyses requiring user input.
- Word documents or exported Jupyter notebooks lack user interaction and challenge reproducibility.
- Building a web application with Flask or Django and deploying on the Cloud is complex and time-consuming.
- Many traditional methods are slow, lack user input handling, or aren't optimal for decision-making.

More on Streamlit

- Streamlit runs our Python files from top to down as a script, so we can perform data manipulation with libraries such as pandas in the same way that we might do it in a Jupyter notebook or a regular Python script.
- Develop in Streamlit and use st.write() as a debugger.
- Explore in Jupyter and then copy to Streamlit.
- If you face issues running Streamlit on your M1/M2 machines, you can use a Virtual Machine.

Streamlit cheat sheet

streamlit.io

This cheat sheet is a summary of the docs

I also recommend streamlitopedia

How to install and import

```
$ pip install streamlit
```

Import convention
```
>>> import streamlit as st
```

Add widgets to sidebar

```
st.sidebar.<widget>

>>> my_val = st.sidebar.text_input('I:')
```

Command line

```
$ streamlit --help
$ streamlit run your_script.py
$ streamlit hello
$ streamlit config show
$ streamlit cache clear
$ streamlit docs
$ streamlit --version
```

Pre-release features

To access beta and experimental features

```
pip uninstall streamlit
pip install streamlit-nightly --upgrade
```

Magic commands

Magic commands allow you to implicitly `st.write()`

```
 ''' _This_ is some __Markdown__ '''

a=3
'a', a

'dataframe:', data
```

Display text

```
st.text('Fixed width text')
st.markdown('_Markdown_') # see *
st.latex(r''' e^{i\pi} + 1 = 0 ''')
st.write('Most objects') # df, err, func, keras!
st.write(['st', 'is <', 3]) # see *
st.title('My title')
st.header(My header')
st.subheader('My sub')
st.code('for i in range(8): foo()')
```

* optional kwarg unsafe_allow_html = True

Display data

```
st.dataframe(data)
st.table(data.iloc[0:10]
st.json({'foo':'bar','fu':'ba'})
```

Display charts

```
st.line_chart(data)
st.area_chart(data)
st.bar_chart(data)
st.pyplot(fig)
st.altair_chart(data)
st.vega_lite_chart(data)
st.plotly_chart(data)
st.bokeh_chart(data)
st.pydeck_chart(data)
st.deck_gl_chart(data)
st.graphviz_chart(data)
st.map(data)
```

Display media

```
st.image('./header.png')
st.audio(data)
st.video(data)
```

Display interactive widgets

```
st.button('Hit me')
st.checkbox('Check me out')
st.radio('Radio', [1,2,3])
st.selectbox('Select', [1,2,3])
st.multiselect('Multiselect', [1,2,3])
st.slider('Slide me', min_value=0, max_value=10)
st.text_input('Enter some text')
st.number_input('Enter a number')
st.text_area('Area for textual entry')
st.date_input('Date input')
st.time_input('Time entry')
st.file_uploader('File uploader')
st.beta_color_picker('Pick a color')
```

Use widgets' returned values in variables:

```
>>> for i in range(int(st.number_input('Num:'))): foo()
>>> if st.sidebar.selectbox('I:',['f']) == 'f': b()
>>> my_slider_val = st.slider('Quinn Mallory', 1, 88)
>>> st.write(slider_val)
```

Control flow

```
st.stop()
```

Display code

```
st.echo()

>>> with st.echo():
>>>     # Code below both executed and printed
>>>     foo = 'bar'
>>>     st.write(foo)
```

Display progress and status

```
st.progress(progress__variable_1_to_100)

st.spinner()

>>> with st.spinner(text='In progress'):
>>>     time.sleep(5)
>>>     st.success('Done')

st.balloons()
st.error('Error message')
st.warning('Warning message')
st.info('Info message')
st.success('Success message')
st.exception(e)
```

Placeholders, help, and options

```
st.empty()

>>> my_placeholder = st.empty()
>>> my_placeholder.text('Replaced!')

st.help(pandas.DataFrame)

st.get_option(key)
st.set_option(key)

st.beta_set_page_config(layout='wide')
```

Mutate data

```
DeltaGenerator.add_rows(data)

>>> my_table = st.table(df1)
>>> my_table.add_rows(df2)

>>> my_chart = st.line_chart(df1)
>>> my_chart.add_rows(df2)
```

Optimize performance

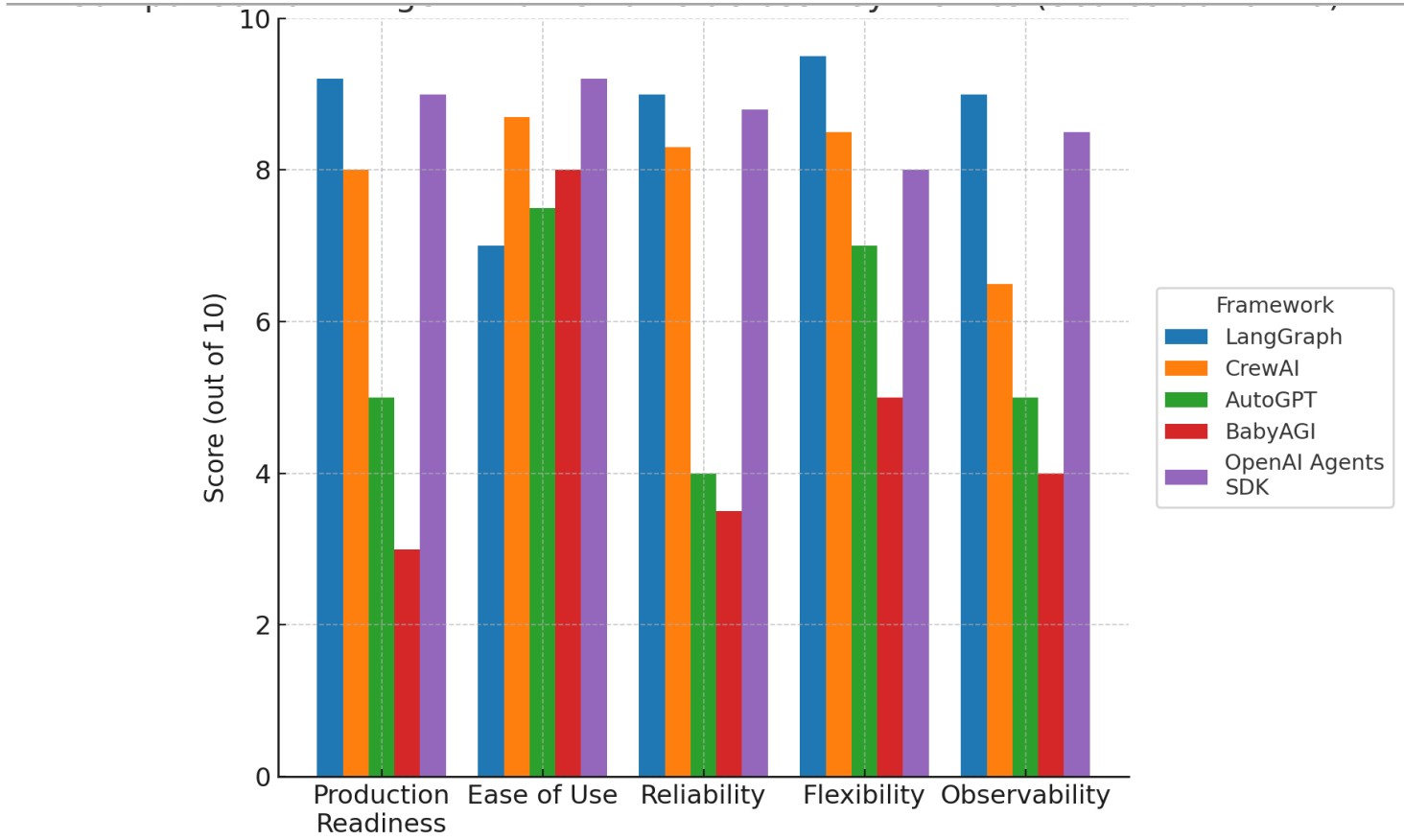
```
@st.cache

>>> @st.cache
... def foo(bar):
...     # Mutate bar
...     return data
...
>>> d1 = foo(ref1)
>>> # Executes as first time
>>>
>>> d2 = foo(ref1)
>>> # Does not execute; returns cached value, d1==d2
>>>
>>> d3 = foo(ref2)
>>> # Different arg, so function executes
```

Streamlit cheat sheet v1.0 | August 2020
https://github.com/daniellewisDL/streamlit-cheat-sheet

Backend: LangGraph

AI Agent Framework Comparison



References:
1. https://kitemetric.com/blogs/ai-agent-frameworks-showdown-openai-agents-sdk-langgraph-autogen-and-crewai
2. https://aiblogfirst.com/autogpt-vs-babyagi-vs-godmode/
3. https://agixtech.com/toolformer-vs-autogpt-vs-babyagi-reliable-agent-architectures/
4. https://medium.com/@maheshus007/autogpt-vs-crewai-which-agentic-framework-is-better-28782ad37c56

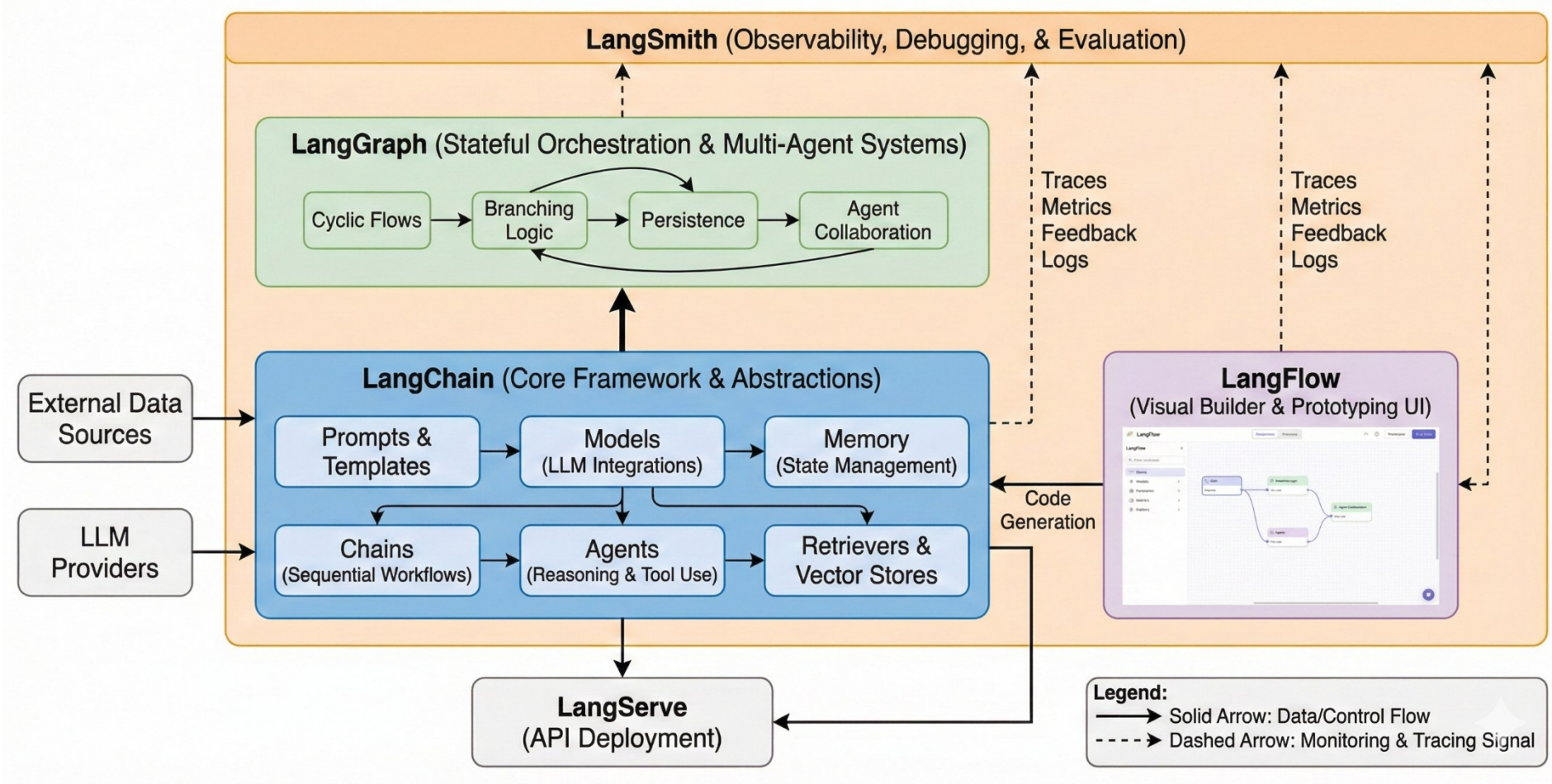
Is LangGraph Self-Sufficient?

- Using LangGraph typically requires access to LLMs and the specification of relevant tools
- The recommended way to do this is to use **LangChain**.
- LangChain provides extensive interoperability with the broader AI ecosystem using 1000+ integrations encompassing embedding models, tools, toolkits, document loaders, and vector stores.
- For more information, check this page: https://docs.langchain.com/oss/python/integrations/providers/overview

Introduction to LangChain

- LangChain is an open-source framework designed to facilitate the development of applications that involve language models.
- LangChain was created in 2022 by Harrison Chase, a researcher in the field of AI.
- LangChain aims to make apps able to reason and are context-aware by creating layers of custom components between LLMs and the end user.
- LangChain is available in Python and JavaScript.

LangChain Ecosysytem

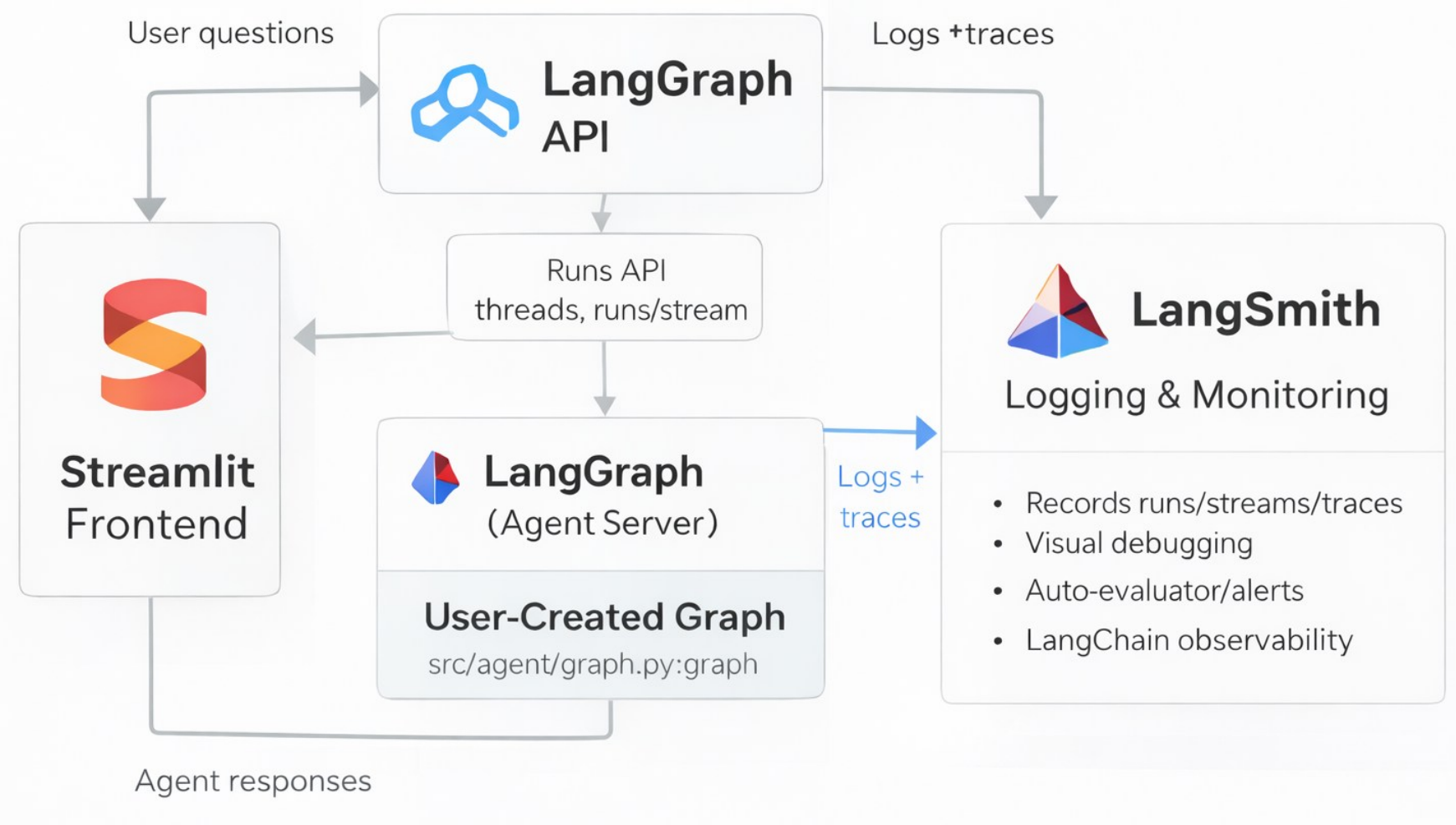


University

LangGraph Quickstart Guide

1. Follow https://docs.langchain.com/oss/python/langgraph/install
2. Sign-up for a LangSmith account: https://smith.langchain.com/

LLM Application Technology Stack



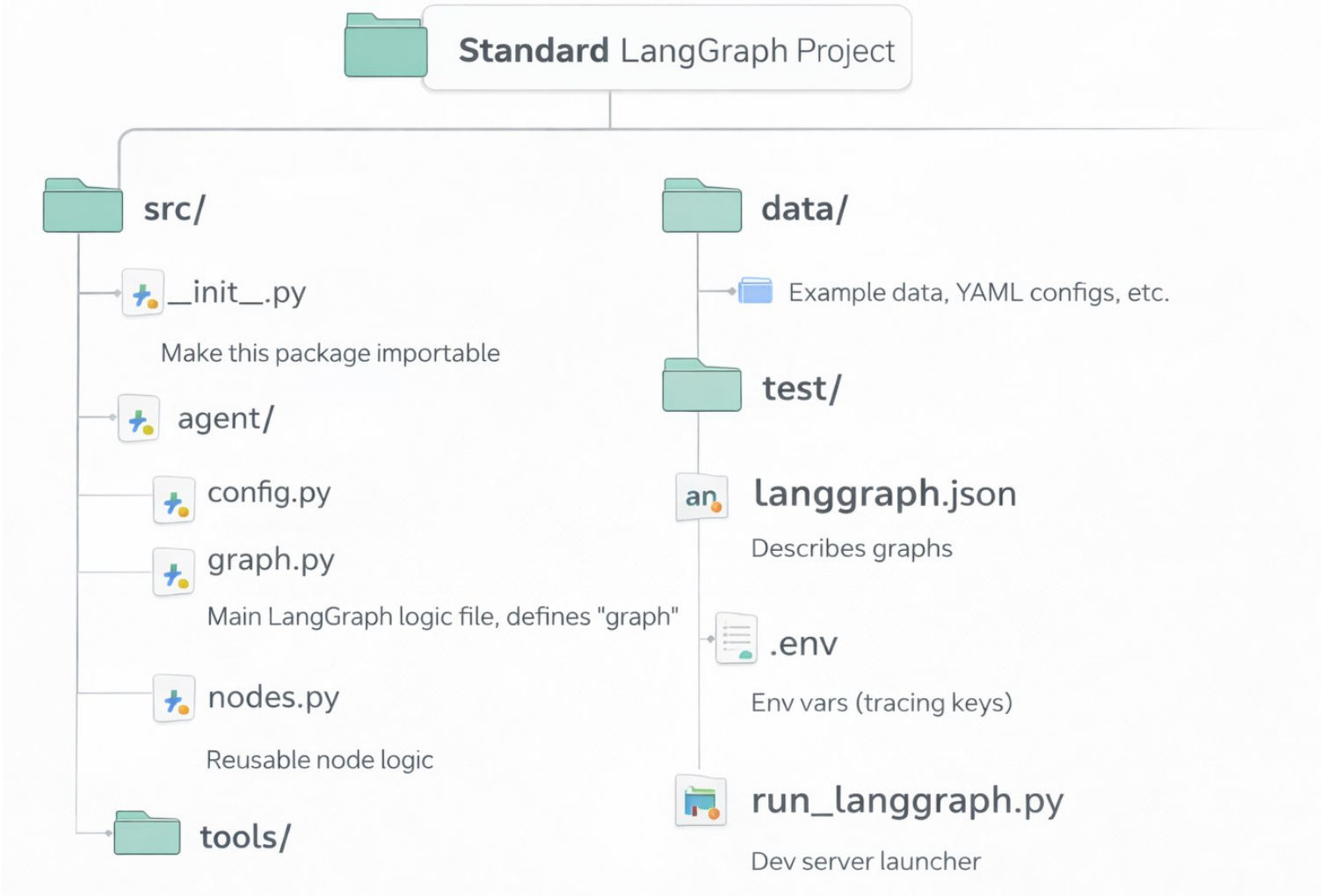
Prerequisites for Lecture Code

- Have Python and VS Code Installed on Your Machine.
- Preferred: Run all installations in a virtual environment.

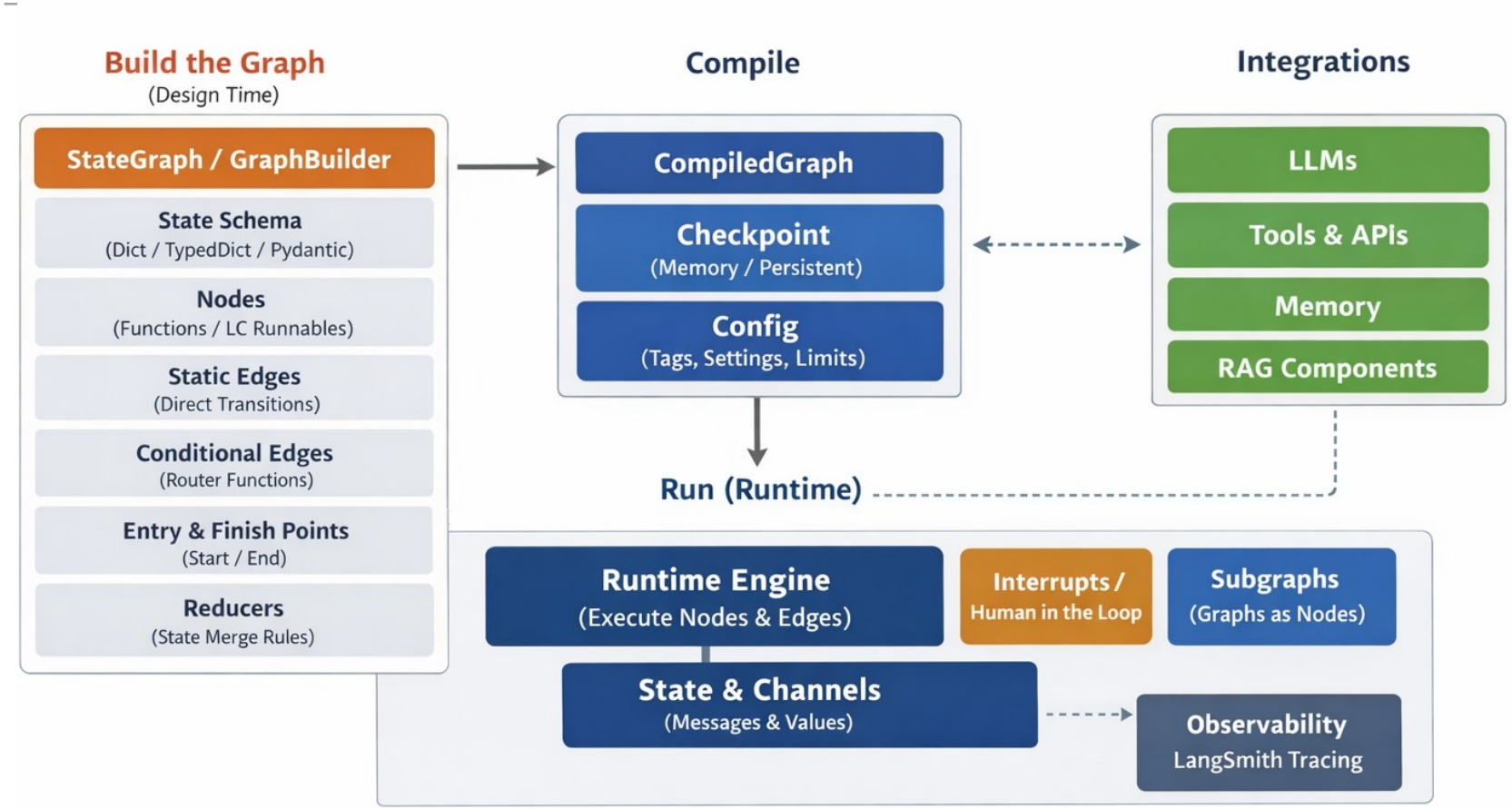
Create LangGraph Project

- Follow the “Studio quickstart” guide on LangSmith to create LangGraph project.
- Check out this GitHub repo and place *app.py* and *graph.py* in their correct locations in your LangGraph project. https://github.com/14-825-GenAI-and-LLM/langgraph-streamlit-files

LLM Application – LangGraph Project Structure



LLM Application – LangGraph Code Flow



Launch the Application

- Launch the backend using *langgraph dev* command.
- In a different terminal, launch the frontend using *streamlit run app.py*

Next Steps

- Redeem GCP Coupons.
- Install Docker Desktop: https://docs.docker.com/get-started/get-docker/