Building and Testing Reliable Agents

LangChain
128K subscribers

Subscribe

552 Share Clip Save

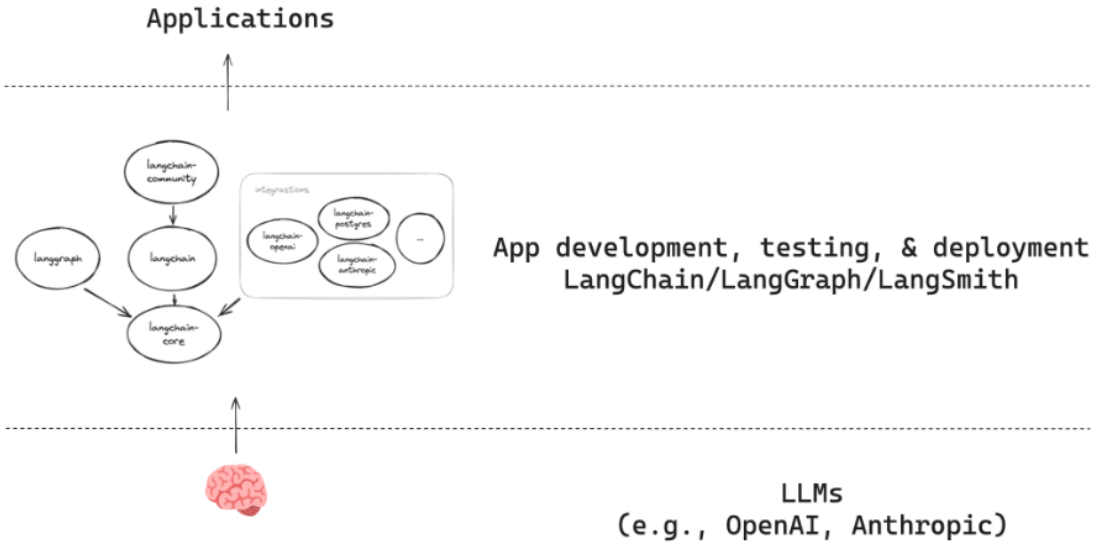
18,333 views Jun 28, 2024

This talk was given as a workshop at the AI Engineering World's Fair on June, 24 2024. LLM-powered agents hold tremendous promise for autonomously performing tasks, but reliability is often a barrier for deployment and productionisation. Here, we'll show how to design and build reliable agents using LangGraph. We'll cover ways to test agents using LangSmith, examining both agent's final response as well as agent tool use trajectory. We'll compare a custom LangGraph agent to a ReAct agent for RAG to showcase the reliability benefit associated with building custom agents using LangGraph.

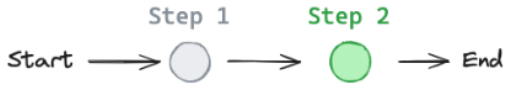
Architecting + testing reliable agents

Lance Martin
Software Engineer, LangChain
@RLanceMartin

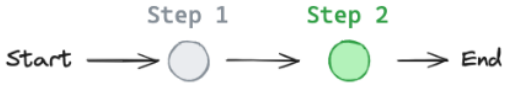
Ecosystem



LLM applications follow a control flow

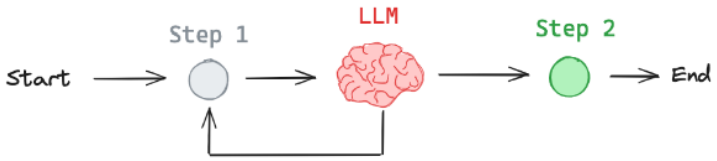


Chain = Control flow set by the developer



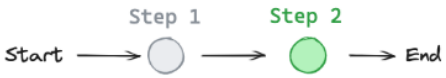
A RAG chain

Agent = Control flow set by an LLM



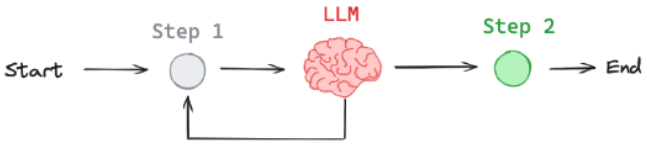
Chain

Developer defined control flow

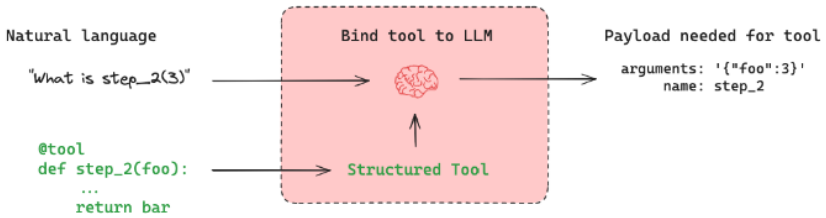


Agent

LLM defined control flow



Agents typically use tool (function) calling to execute steps



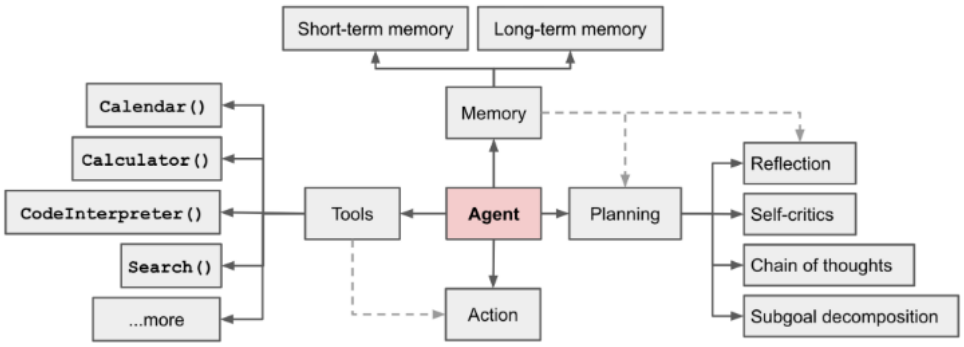
Agent System Overview

In a LLM-powered autonomous agent system, LLM functions as the agent's brain, complemented by several key components:

- **Planning**
  - Subgoal and decomposition: The agent breaks down large tasks into smaller, manageable subgoals, enabling efficient handling of complex tasks.
  - Reflection and refinement: The agent can do self-criticism and self-reflection over past actions, learn from mistakes and refine them for future steps, thereby improving the quality of final results.

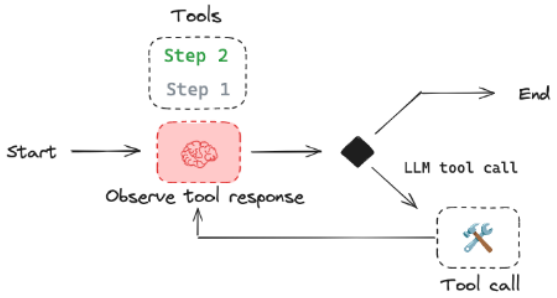
- **Memory**
  - Short-term memory: I would consider all the in-context learning (See Prompt Engineering) as utilizing short-term memory of the model to learn.
  - Long-term memory: This provides the agent with the capability to retain and recall (infinite) information over extended periods, often by leveraging an external vector store and fast retrieval.

- **Tool use**
  - The agent learns to call external APIs for extra information that is missing from the model weights (often hard to change after pre-training), including current information, code execution capability, access to proprietary information sources and more.

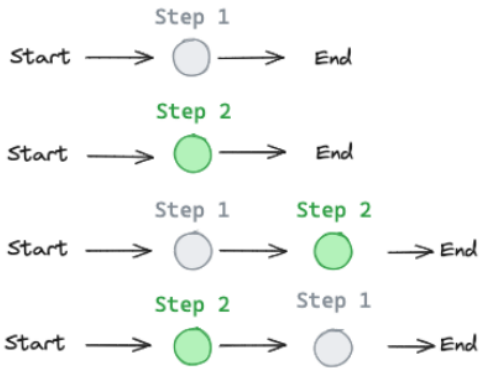


Overview of a LLM-powered autonomous agent system.

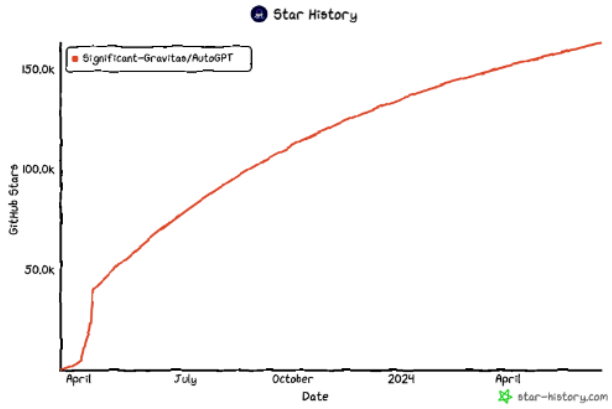
A popular approach is to just run this as a `for loop` (ReAct)



ReAct agent is flexible w/ many possible control flows

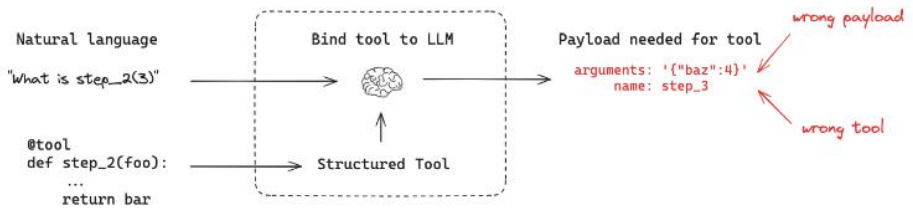


Flexible agents have promise for open-ended tasks



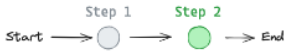
But, they can suffer from poor reliability

Often caused by LLM non-determinism and / or tool calling errors



Can we envision something in the middle?

Chain



Not flexible

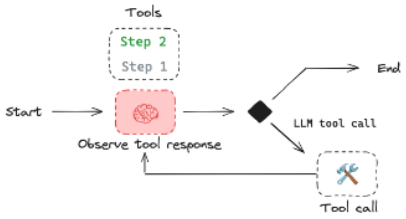
More reliable

?

Flexible

Reliable

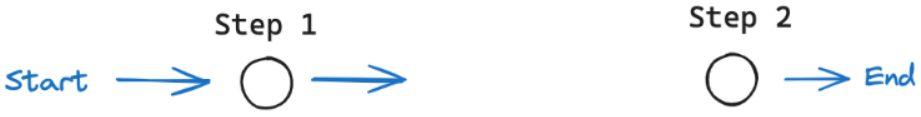
Agent (for loop)



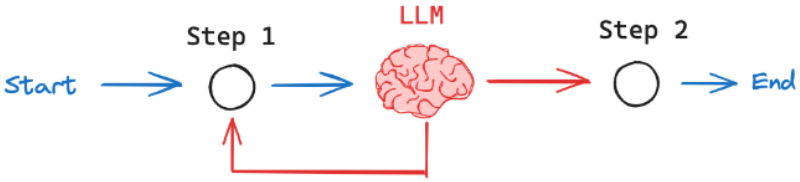
Flexible

Less reliable

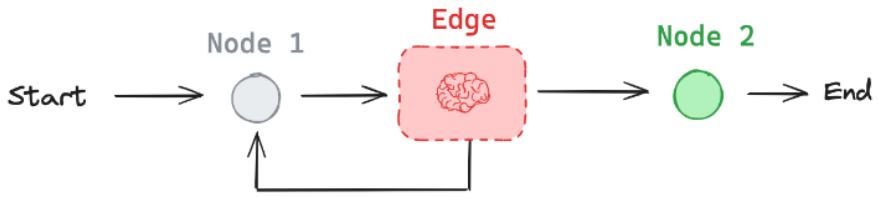
Intuition: let developer set parts of control flow (reliable)



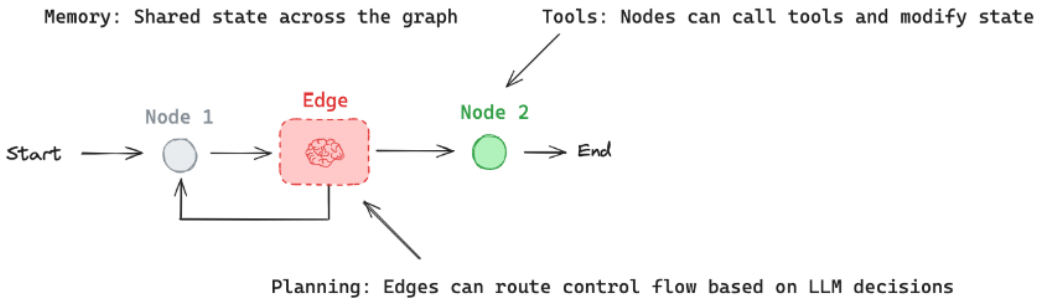
Intuition: inject LLM to make it an agent (flexible)



LangGraph: express control flows as graphs



LangGraph agent

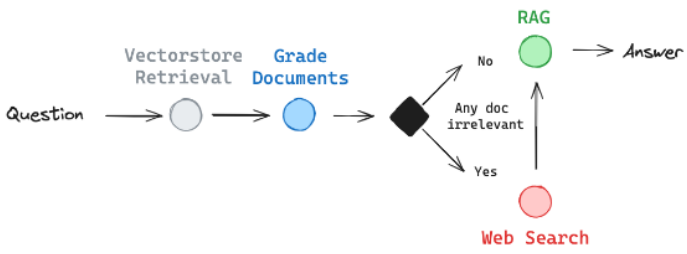


Papers use this theme for routing

Corrective Retrieval Augmented Generation

Shi-Qi Yan[1*], **Jia-Chen Gu**[2*], **Yun Zhu**[3], **Zhen-Hua Ling**[1]
[1]National Engineering Research Center of Speech and Language Information Processing,
University of Science and Technology of China, Hefei, China
[2]Department of Computer Science, University of California, Los Angeles
[3]Google Research
yansiki@mail.ustc.edu.cn, gujc@ucla.edu, yunzhu@google.com, zhling@ustc.edu.cn

Corrective RAG in LangGraph

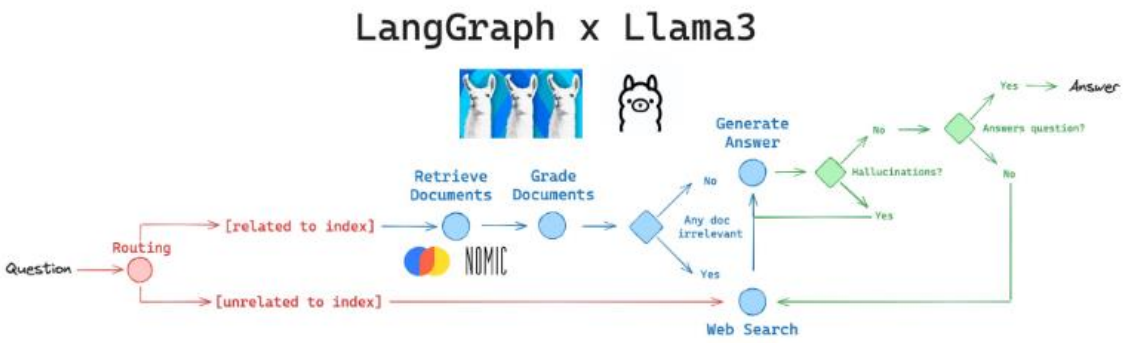


Papers use this theme for self-reflection

SELF-RAG: LEARNING TO RETRIEVE, GENERATE, AND CRITIQUE THROUGH SELF-REFLECTION

Akari Asai[†], **Zeqiu Wu**[†], **Yizhong Wang**[†§], **Avirup Sil**[‡], **Hannaneh Hajishirzi**[†§]
[†]University of Washington [§]Allen Institute for AI [‡]IBM Research AI
{akari, zeqiuwu, yizhongw, hannaneh}@cs.washington.edu, avi@us.ibm.com

Self-RAG, Corrective RAG, + Adaptive RAG in LangGraph



Papers use this theme for iterative complex problem solving

Andrej Karpathy @karpathy

Prompt engineering (or rather "Flow engineering") intensifies for code generation. Great reading and a reminder of how much alpha there is (pass@5 19% to 44%) in moving from a naive prompt:answer paradigm to a "flow" paradigm, where the answer is constructed iteratively.

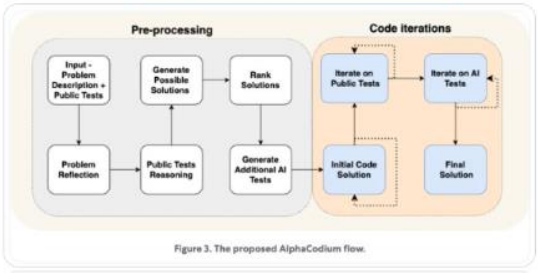


Figure 3. The proposed AlphaCodium flow.

Code Generation with AlphaCodium: From Prompt Engineering to Flow Engineering

Tal Ridnik, Dedy Kredo, Itamar Friedman
CodiumAI
{tal.r, dedy.k, itamar.f}@codium.ai

Abstract

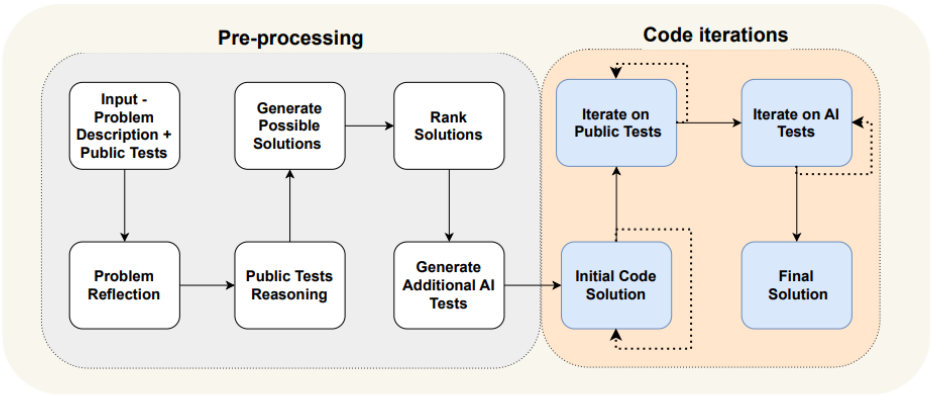
Code generation problems differ from common natural language problems - they require matching the exact syntax of the target language, identifying happy paths and edge cases, paying attention to numerous small details in the problem spec, and addressing other code-specific issues and requirements. Hence, many of the optimizations and tricks that have been successful in natural language generation may not be effective for code tasks. In this work, we propose a new approach to code generation by LLMs, which we call AlphaCodium - a test-based, multi-stage, code-oriented iterative flow, that improves the performances of LLMs on code problems. We tested AlphaCodium on a challenging code generation dataset called CodeContests, which includes competitive programming problems from platforms such as Codeforces. The proposed flow consistently and significantly improves results. On the validation set, for example, GPT-4 accuracy (pass@5) increased from 19% with a single well-designed direct prompt to 44% with the AlphaCodium flow. Many of the principles and best practices acquired in this work, we believe, are broadly applicable to general code generation tasks.

Full implementation is available at: https://github.com/Codium-ai/AlphaCodium

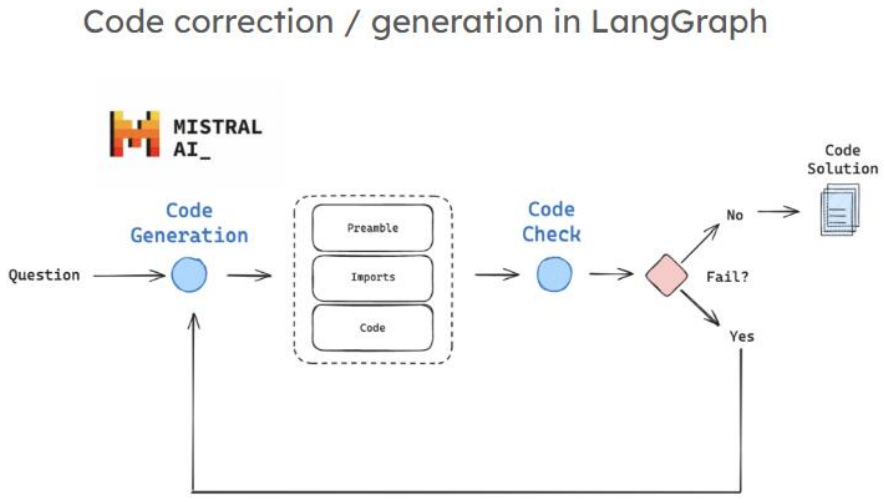
els [12] have successfully generated code that solves simple programming tasks [2, 1]. However, real-world code problems are often different in nature - they are more nuanced, and can be defined by a long natural language task description (i.e., spec), that contains multiple details and rules that the solution code must address.

The introduction of CodeContests [8], a dataset curated from competitive programming platforms such as Codeforces [9], enabled the evaluation of models and flows on more challenging code problems, which usually include a lengthy problem description. A private test set, with more than 200 unseen tests per problem, enables to evaluate the generated code comprehensively, and to reduce false positive rates to a minimum.

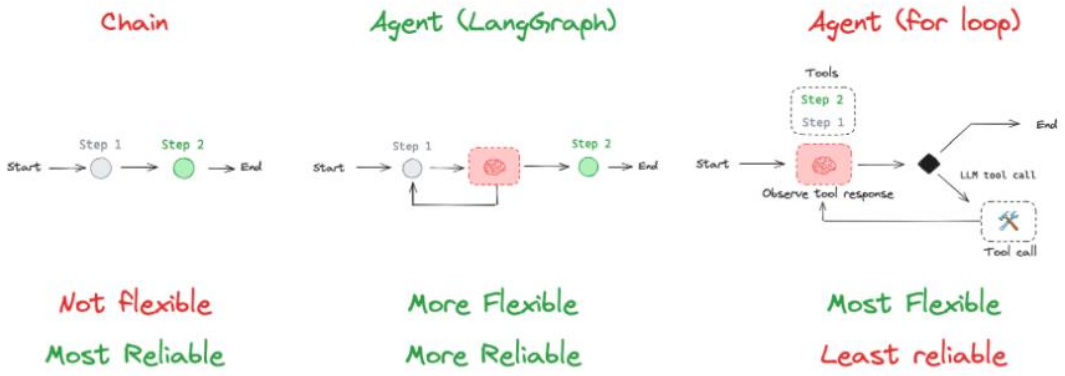
The primary work addressing the CodeContests dataset was AlphaCode [8], a code generation system developed by DeepMind, that utilizes a fine-tuned network specifically for competitive programming tasks. AlphaCode generates a very large number of possible solutions (up to 1M), that are then processed and clustered, and among them a small number ($\sim$ 10) is chosen and submitted. While the results of AlphaCode are impressive, the need to fine-tune a model specifically for code-oriented tasks, and the heavy computational brute-force-like load, makes it impractical for most real-life usages. CodeChain [7] is another work to tackle



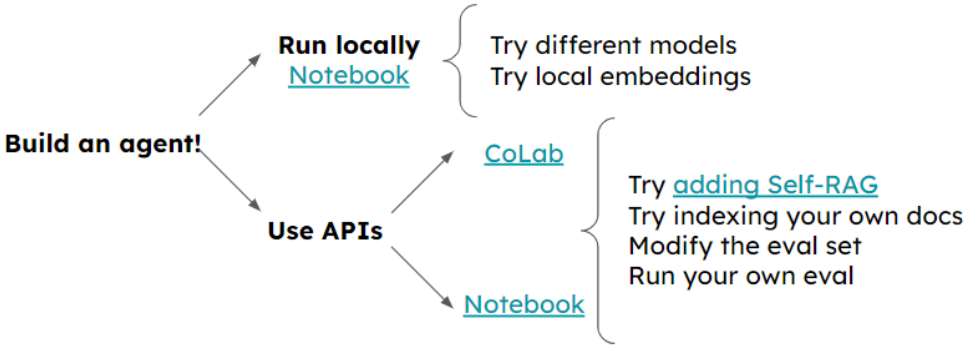
(a) The proposed AlphaCodium flow.

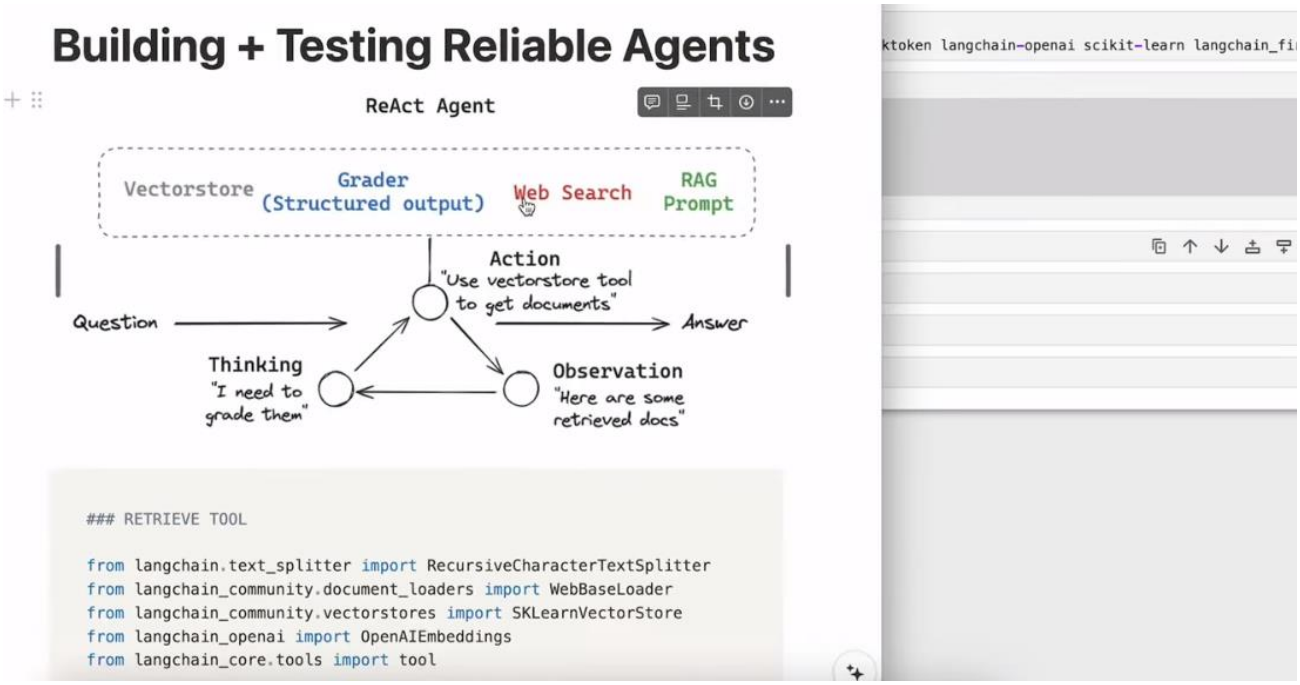


LangGraph allows for developer + LLM-defined control flow



Choose your own adventure! (All notebooks have optional testing / eval)





Python

Copy Caption

```
### RETRIEVE TOOL

from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.document_loaders import WebBaseLoader
from langchain_community.vectorstores import SKLearnVectorStore
from langchain_openai import OpenAIEmbeddings
from langchain_core.tools import tool

# List of URLs to load documents from
urls = [
    "https://lilianweng.github.io/posts/2023-06-23-agent/",
    "https://lilianweng.github.io/posts/2023-03-15-prompt-engineerin
g/",
    "https://lilianweng.github.io/posts/2023-10-25-adv-attack-llm/",
]

# Load documents from the URLs
docs = [WebBaseLoader(url).load() for url in urls]
docs_list = [item for sublist in docs for item in sublist]

# Initialize a text splitter with specified chunk size and overlap
text_splitter = RecursiveCharacterTextSplitter.from_tiktoken_encoder(
    chunk_size=250, chunk_overlap=0
)

# Split the documents into chunks
doc_splits = text_splitter.split_documents(docs_list)

# Add the document chunks to the "vector store" using OpenAIEmbeddings
vectorstore = SKLearnVectorStore.from_documents(
```

ktoken langchain-openai scikit-learn langchain_firewo

Trusted

JupyterLab Python 3 (ipykernel)

ktoken langchain-openai scikit-learn langchain_firewo


```
# Load documents from the URLs
docs = [WebBaseLoader(url).load() for url in urls]
docs_list = [item for sublist in docs for item in sublist]

# Initialize a text splitter with specified chunk size and overlap
text_splitter = RecursiveCharacterTextSplitter.from_tiktoken_encoder(
    chunk_size=250, chunk_overlap=0
)

# Split the documents into chunks
doc_splits = text_splitter.split_documents(docs_list)

# Add the document chunks to the "vector store" using OpenAIEmbeddings
vectorstore = SKLearnVectorStore.from_documents(
    documents=doc_splits,
    embedding=OpenAIEmbeddings(),
)
retriever = vectorstore.as_retriever(k=4)

# Define a tool, which we will connect to our agent
@tool
def retrieve_documents(query: str) -> list:
    """Retrieve documents from the vector store based on the query."""
    return retriever.invoke(query)

### GRADE TOOL

@tool
def grade_document_retrieval(step_by_step_reasoning: str, score: int) -
> str:
    """You are a teacher grading a quiz. You will be given:
    1/ a QUESTION
    2/ a set of comma separated FACTS provided by the student
```

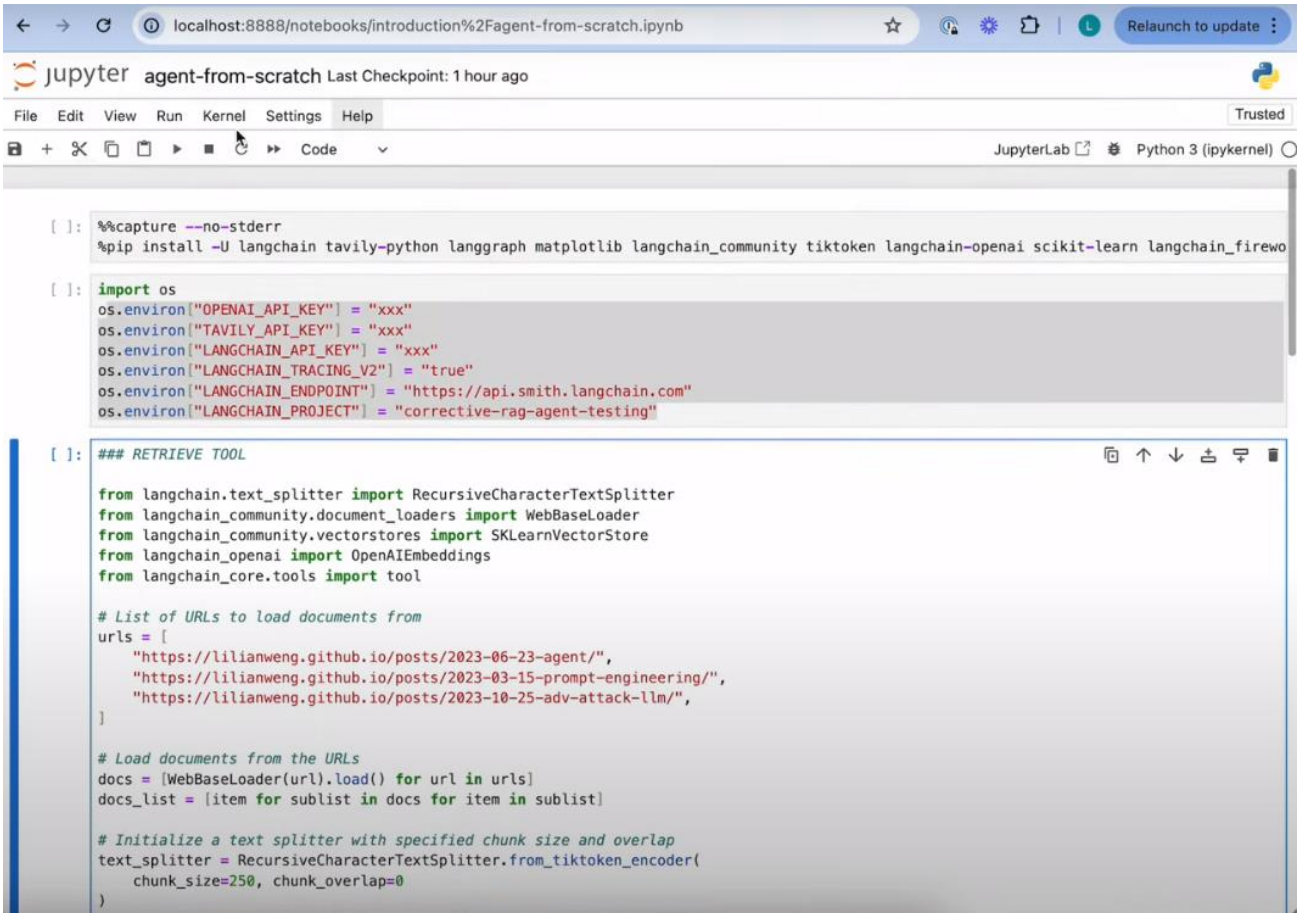
```python
# Define a tool, which we will connect to our agent
@tool
def retrieve_documents(query: str) -> list:
    """Retrieve documents from the vector store based on the query."""
    return retriever.invoke(query)

### GRADE TOOL

@tool
def grade_document_retrieval(step_by_step_reasoning: str, score: int) -
> str:
    """You are a teacher grading a quiz. You will be given:
    1/ a QUESTION
    2/ a set of comma separated FACTS provided by the student

    You are grading RELEVANCE RECALL:
    A score of 1 means that ANY of the FACTS are relevant to the QUESTI
ON.
    A score of 0 means that NONE of the FACTS are relevant to the QUEST
ION.

    If your score is 1: then call a tool to generate the answer, genera
te_answer
    If your score is 0: then call a tool to perform web search, web_sea
rch."""
    if score == 1:
        return "Docs are relevant. Generate the answer to the questio
n."
    return "Docs are not relevant. Use web search to find more document
s."

### WEB SEARCH TOOL
```

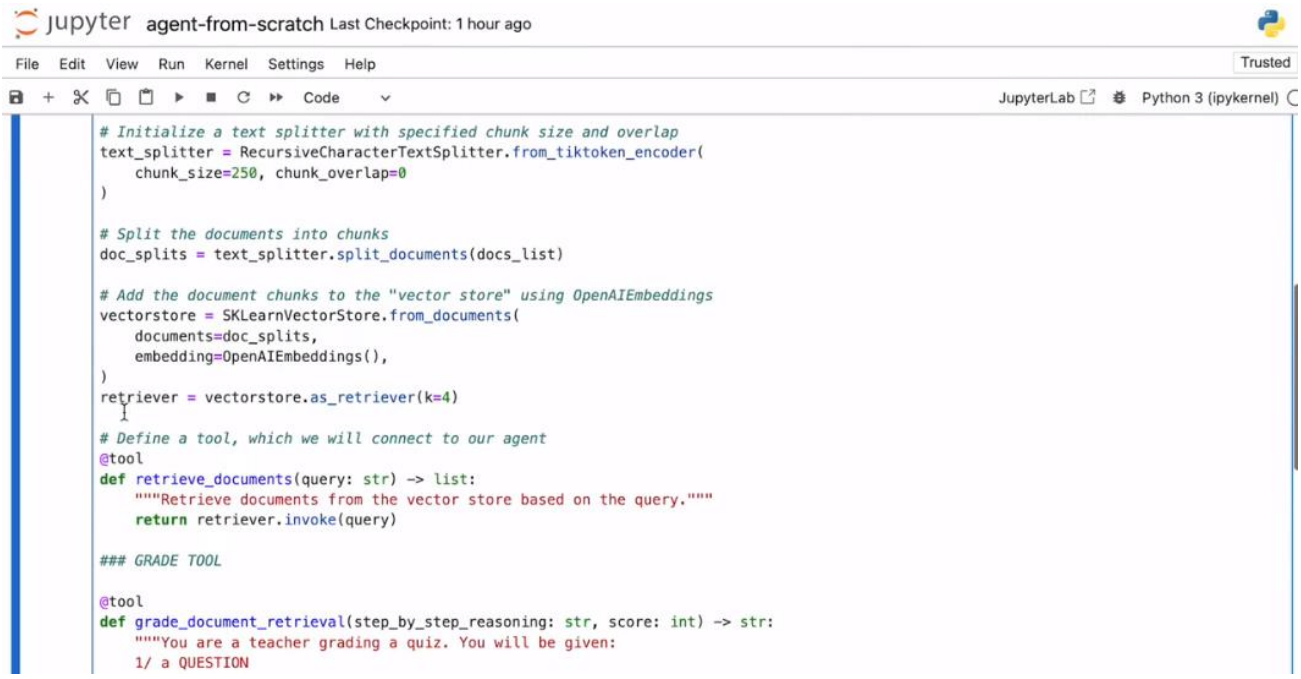

```python
### WEB SEARCH TOOL

from langchain.schema import Document
from langchain_community.tools.tavily_search import TavilySearchResults

web_search_tool = TavilySearchResults()

@tool
def web_search(query: str) -> str:
    """Run web search on the question."""
    web_results = web_search_tool.invoke({"query": query})
    return [
        Document(page_content=d["content"], metadata={"url": d["url"]})
        for d in web_results
    ]

### GENERATE TOOL

@tool
def generate_answer(answer: str) -> str:
    """You are an assistant for question-answering tasks.
    Use the retrieved documents to answer the user question.
    If you don't know the answer, just say that you don't know.
    Use three sentences maximum and keep the answer concise"""
    return f"Here is the answer to the user question: {answer}"
```

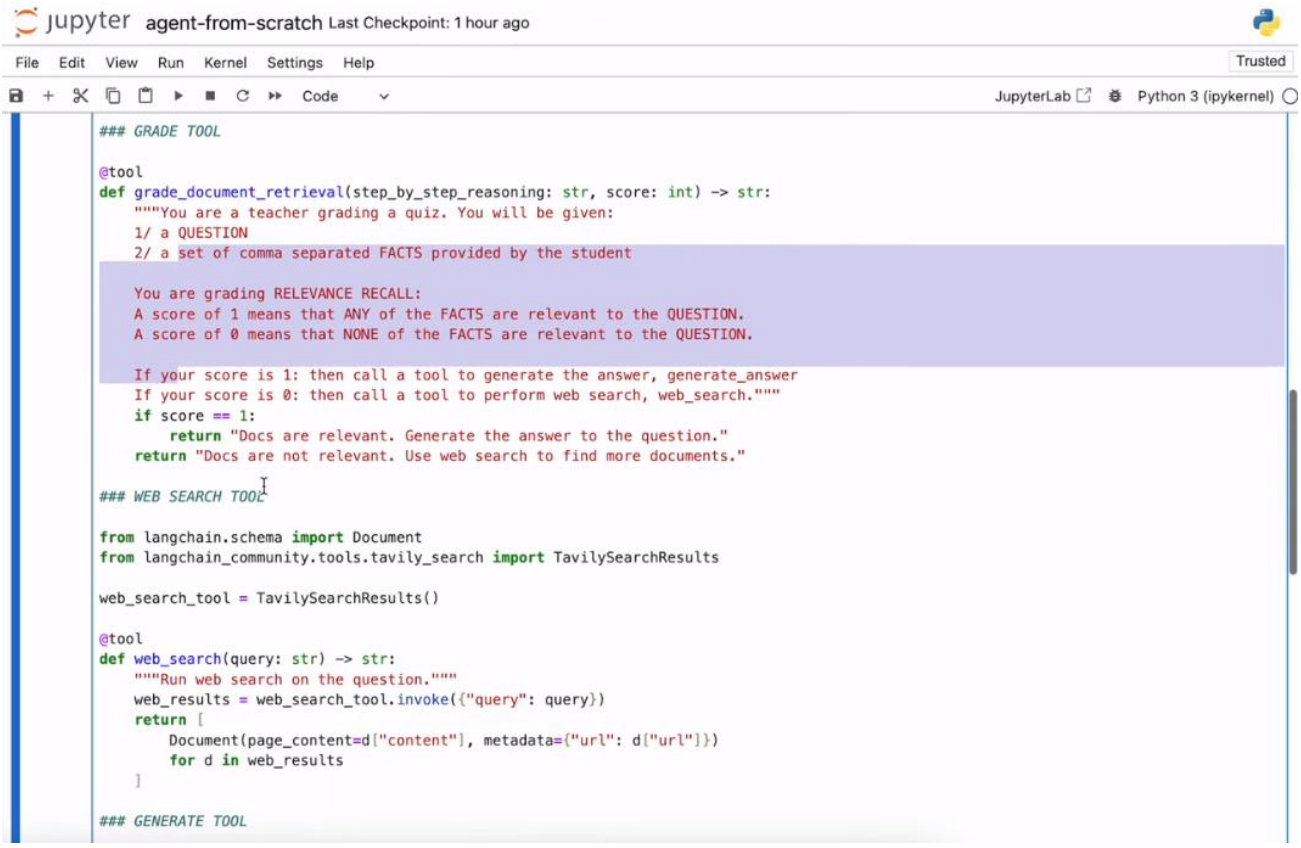
```
### GENERATE TOOL

@tool
def generate_answer(answer: str) -> str:
    """You are an assistant for question-answering tasks.
    Use the retrieved documents to answer the user question.
    If you don't know the answer, just say that you don't know.
    Use three sentences maximum and keep the answer concise"""
    return f"Here is the answer to the user question: {answer}"

tools = [retrieve_documents, grade_document_retrieval, web_search, gene
rate_answer]
```

```
### Assistant

from typing import Annotated, List
from typing_extensions import TypedDict
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.runnables import Runnable, RunnableConfig
from langchain_openai import ChatOpenAI
from langgraph.graph.message import AnyMessage, add_messages

# LLM
model_tested = "gpt-4o"
metadata = "CRAG, gpt-4o"
llm = ChatOpenAI(model_name=model_tested, temperature=0)

# State
class State(TypedDict):
    messages: Annotated[list[AnyMessage], add_messages]
```

```
ktoken langchain-openai scikit-learn langchain_firewo
```




jupyter agent-from-scratch Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
# Initialize a text splitter with specified chunk size and overlap
text_splitter = RecursiveCharacterTextSplitter.from_tiktoken_encoder(
chunk_size=250, chunk_overlap=0
)
# Split the documents into chunks
doc_splits = text_splitter.split_documents(docs_list)
# Add the document chunks to the "vector store" using OpenAIEmbeddings
vectorstore = SKLearnVectorStore.from_documents(
documents=doc_splits,
embedding=OpenAIEmbeddings(),
)
retriever = vectorstore.as_retriever(k=4)
# Define a tool, which we will connect to our agent
@tool
def retrieve_documents(query: str) -> list:
"""Retrieve documents from the vector store based on the query."""
return retriever.invoke(query)
GRADE TOOL
@tool
def grade_document_retrieval(step_by_step_reasoning: str, score: int) -> str:
"""You are a teacher grading a quiz. You will be given:
1/ a QUESTION


jupyter agent-from-scratch Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
GRADE TOOL
@tool
def grade_document_retrieval(step_by_step_reasoning: str, score: int) -> str:
"""You are a teacher grading a quiz. You will be given:
1/ a QUESTION
2/ a set of comma separated FACTS provided by the student
You are grading RELEVANCE RECALL:
A score of 1 means that ANY of the FACTS are relevant to the QUESTION.
A score of 0 means that NONE of the FACTS are relevant to the QUESTION.
If your score is 1: then call a tool to generate the answer, generate_answer
If your score is 0: then call a tool to perform web search, web_search."""
if score == 1:
return "Docs are relevant. Generate the answer to the question."
return "Docs are not relevant. Use web search to find more documents."
WEB SEARCH TOOL
from langchain.schema import Document
from langchain_community.tools.tavily_search import TavilySearchResults
web_search_tool = TavilySearchResults()
@tool
def web_search(query: str) -> str:
"""Run web search on the question."""
web_results = web_search_tool.invoke({"query": query})
return [
Document(page_content=d["content"], metadata={"url": d["url"]})
for d in web_results
]
GENERATE TOOL

localhost:8888/notebooks/introduction%2Fagent-from-scratch.ipynb

Relaunch to update

jupyter agent-from-scratch Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab

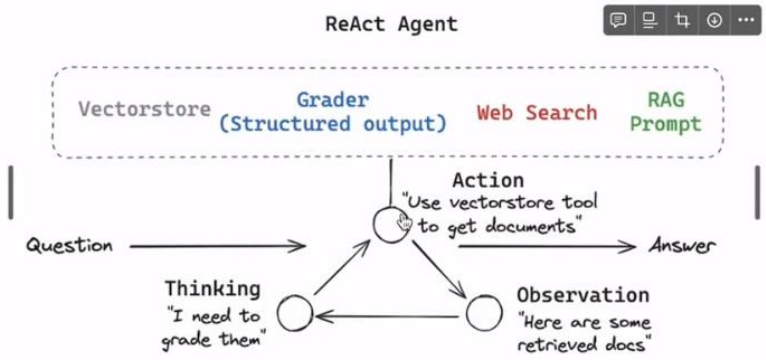
Python 3 (ipykernel)

```
    if score == 1:
        return "Docs are relevant. Generate the answer to the question."
    return "Docs are not relevant. Use web search to find more documents."

### WEB SEARCH TOOL

from langchain.schema import Document
from langchain_community.tools.tavily_search import TavilySearchResults

web_search_tool = TavilySearchResults()

@tool
def web_search(query: str) -> str:
    """Run web search on the question."""
    web_results = web_search_tool.invoke({"query": query})
    return [
        Document(page_content=d["content"], metadata={"url": d["url"]})
        for d in web_results
    ]

### GENERATE TOOL

@tool
def generate_answer(answer: str) -> str:
    """You are an assistant for question-answering tasks.
    Use the retrieved documents to answer the user question.
    If you don't know the answer, just say that you don't know.
    Use three sentences maximum and keep the answer concise"""
    return f"Here is the answer to the user question: {answer}"

tools = [retrieve_documents, grade_document_retrieval, web_search, generate_answer]
```

```
USER_AGENT environment variable not set, consider setting it to identify your requests.
```

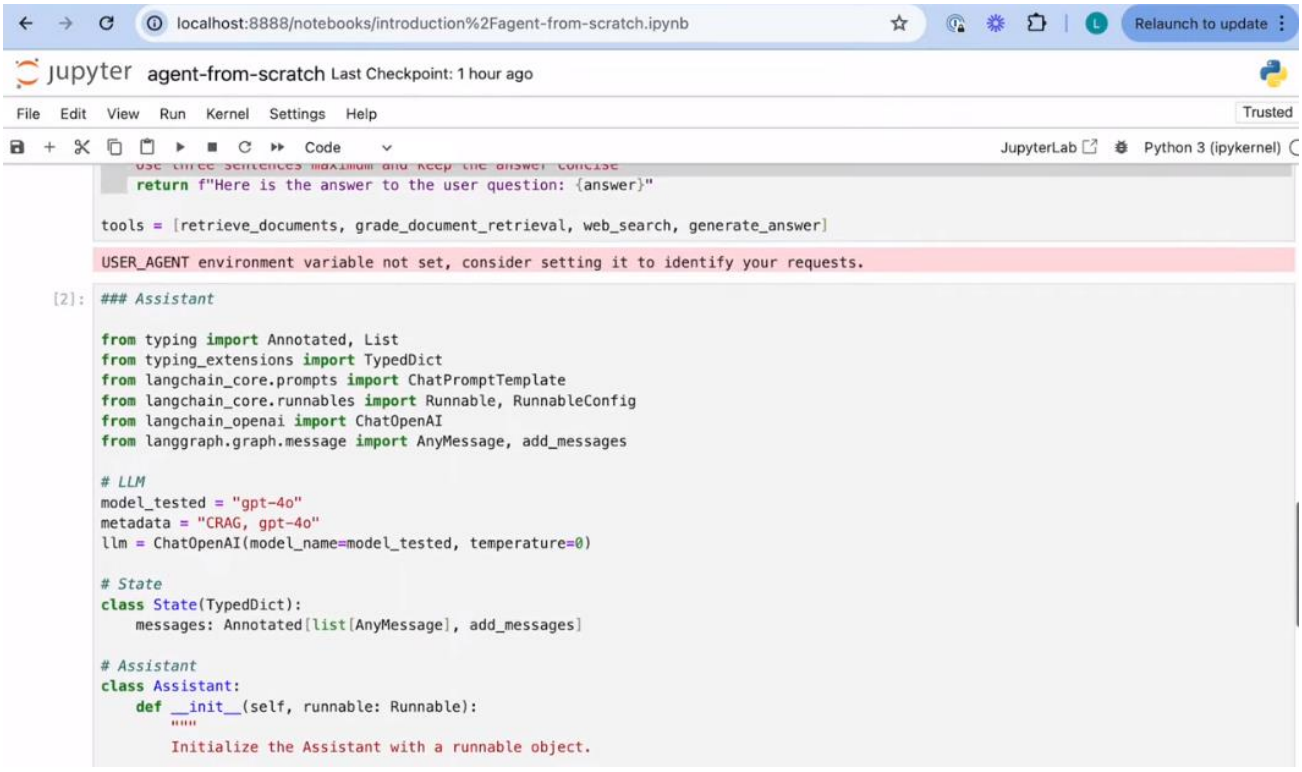
[ ]:

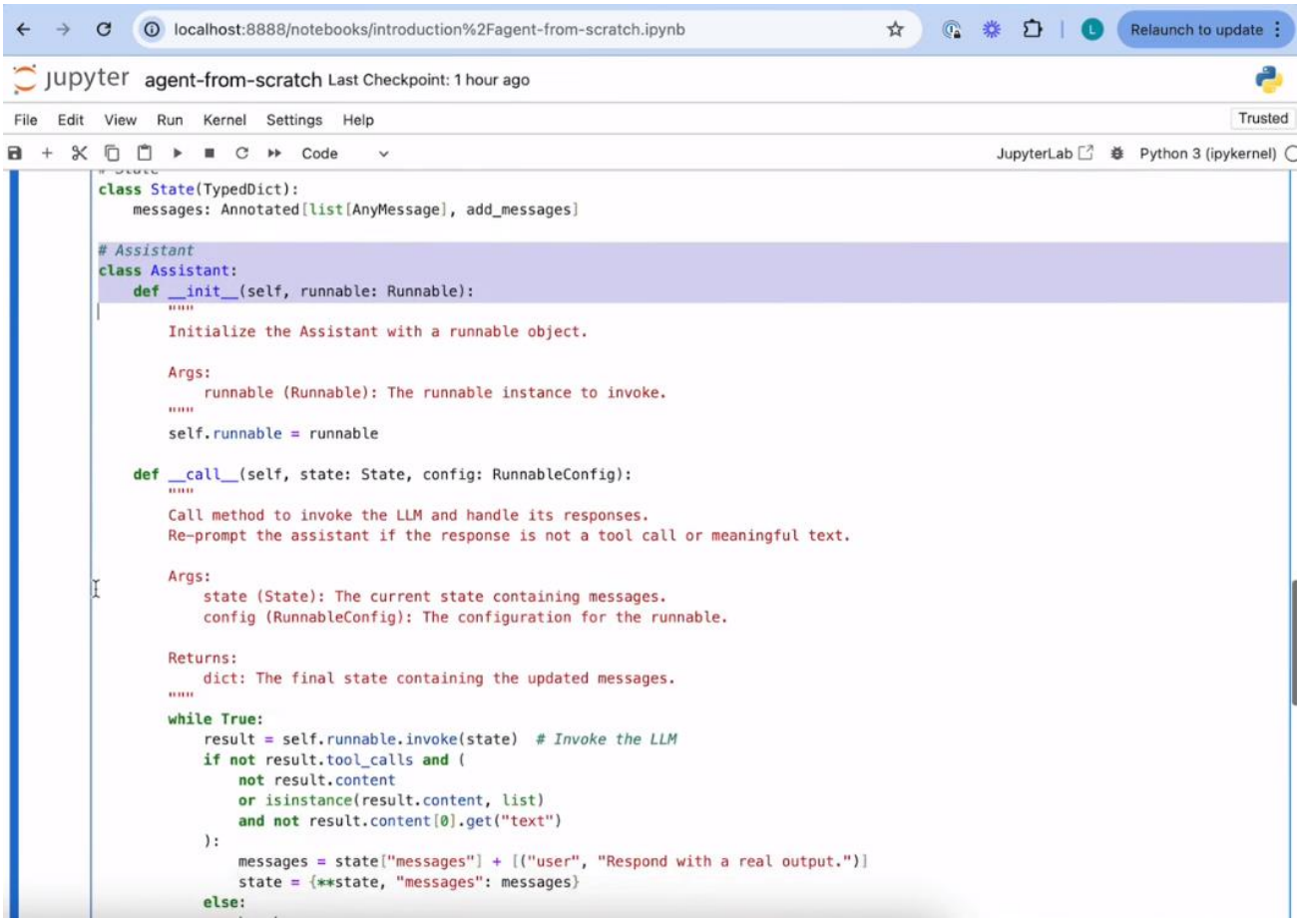
Building + Testing Reliable Agents

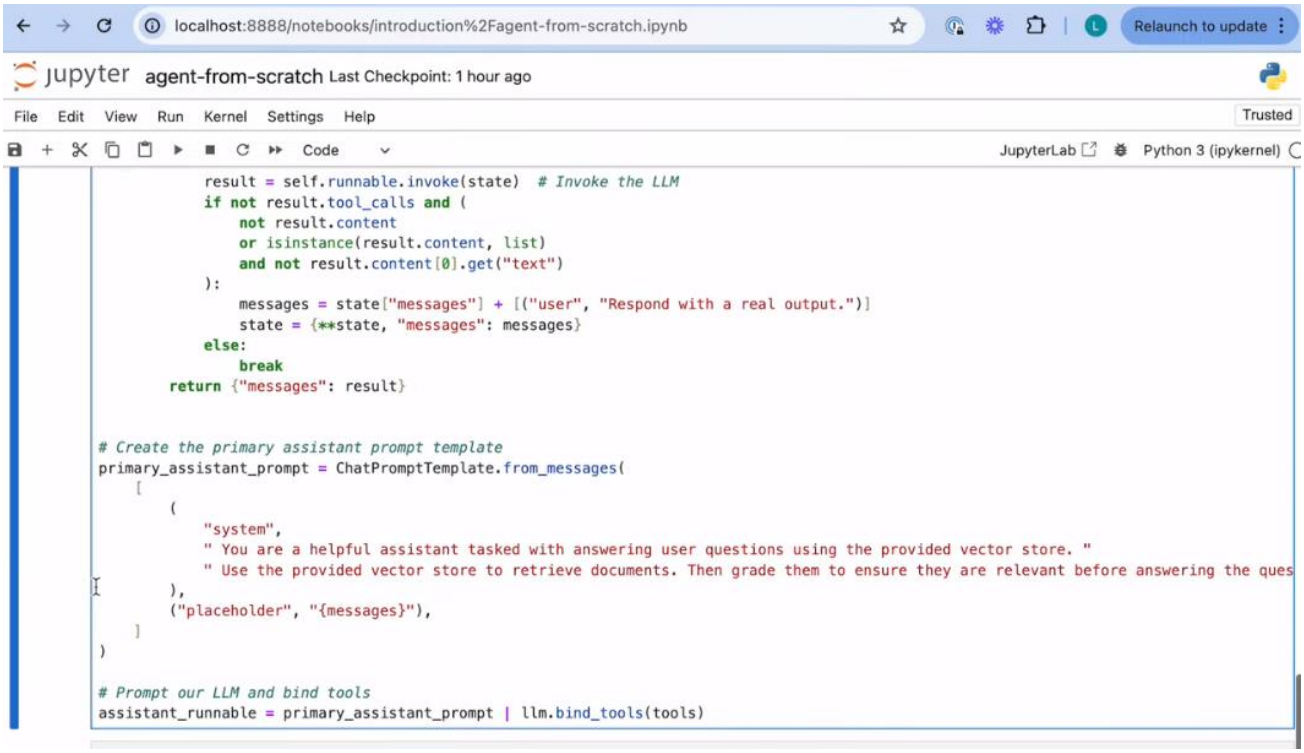


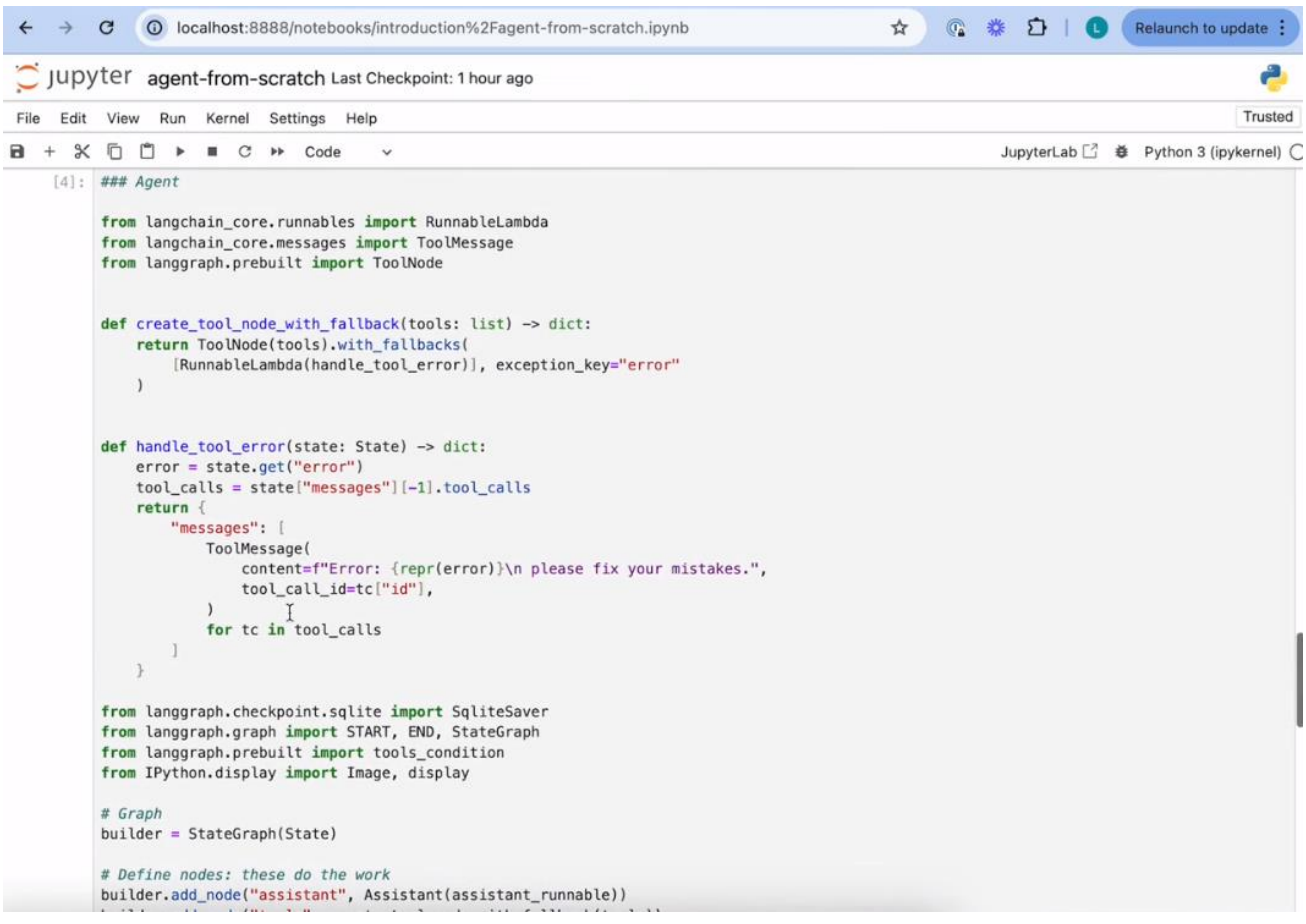
Code

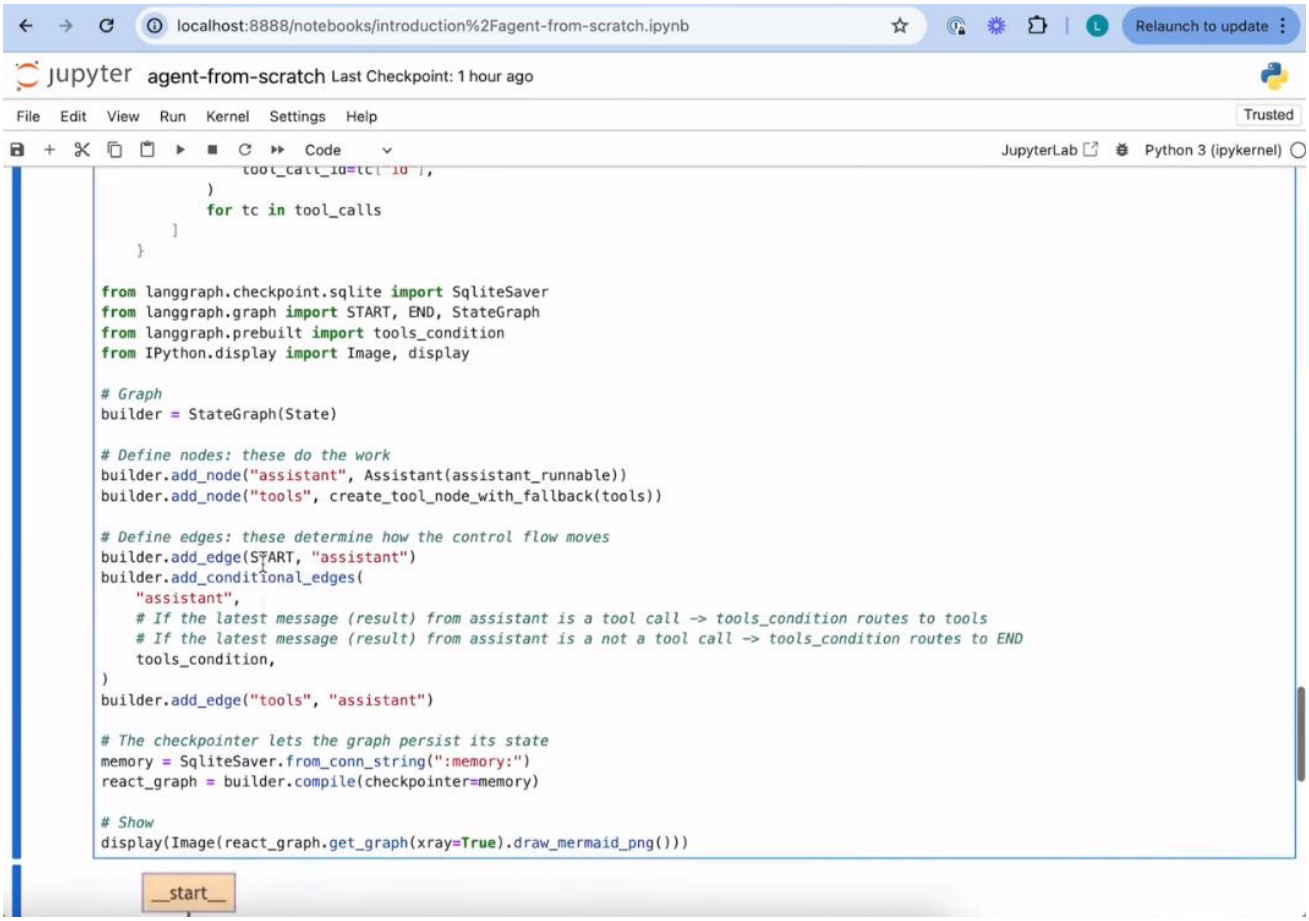
```
### RETRIEVE TOOL

from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.document_loaders import WebBaseLoader
from langchain_community.vectorstores import SKLearnVectorStore
from langchain_openai import OpenAIEmbeddings
```

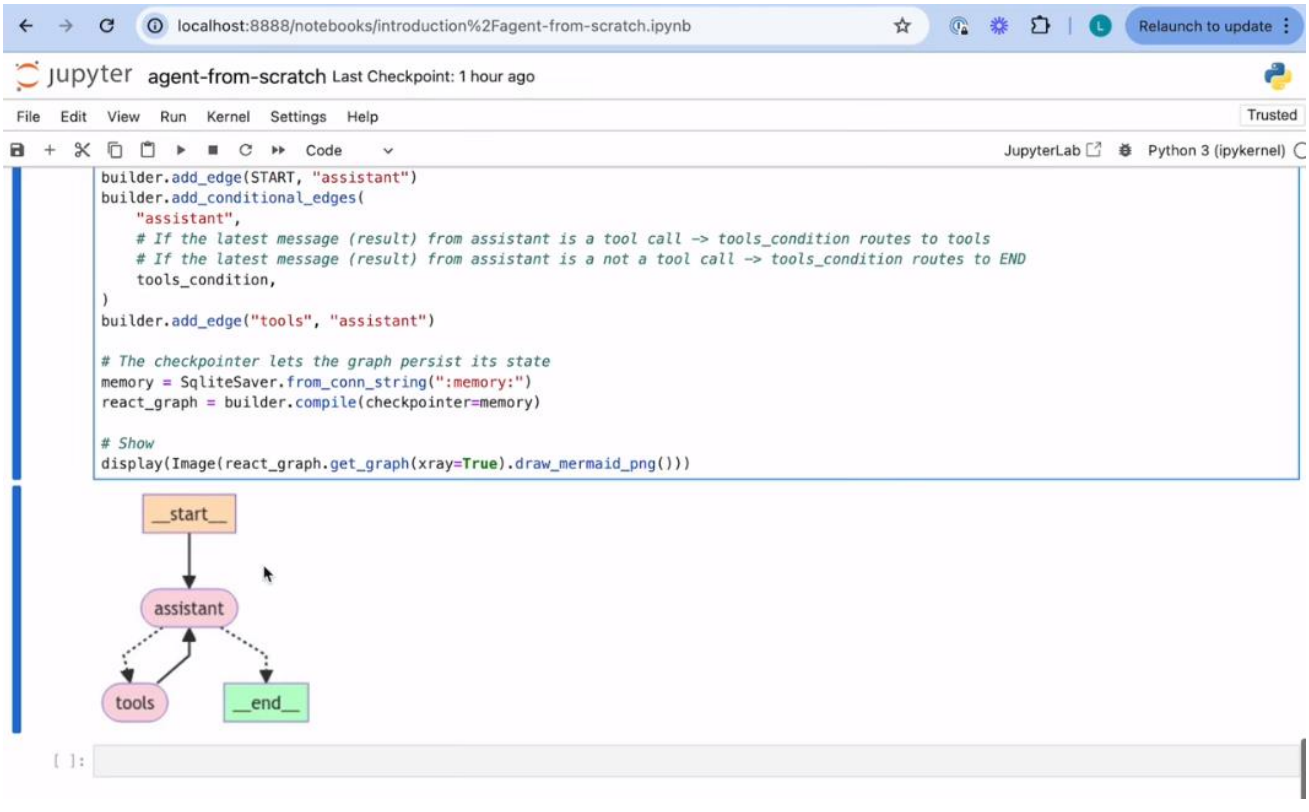

localhost:8888/notebooks/introduction%2Fagent-from-scratch.ipynb
Relaunch to update
jupyter agent-from-scratch Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
return f"Here is the answer to the user question: {answer}"
tools = [retrieve_documents, grade_document_retrieval, web_search, generate_answer]
USER_AGENT environment variable not set, consider setting it to identify your requests.
[2]:
Assistant
from typing import Annotated, List
from typing_extensions import TypedDict
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.runnables import Runnable, RunnableConfig
from langchain_openai import ChatOpenAI
from langgraph.graph.message import AnyMessage, add_messages
# LLM
model_tested = "gpt-4o"
metadata = "CRAG, gpt-4o"
llm = ChatOpenAI(model_name=model_tested, temperature=0)
# State
class State(TypedDict):
messages: Annotated[list[AnyMessage], add_messages]
# Assistant
class Assistant:
def __init__(self, runnable: Runnable):
"""
Initialize the Assistant with a runnable object.

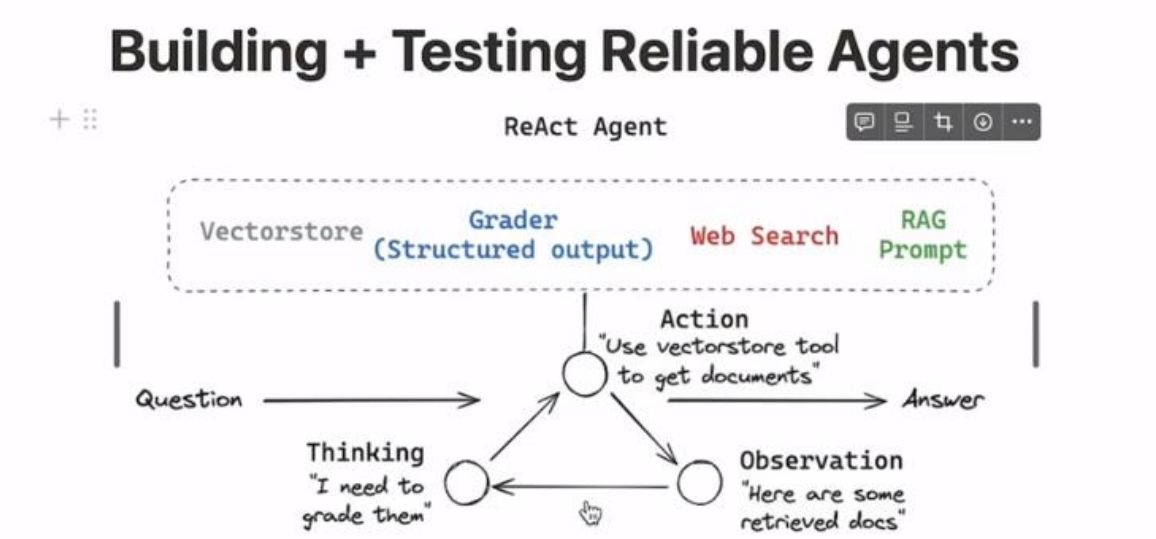

localhost:8888/notebooks/introduction%2Fagent-from-scratch.ipynb
Relaunch to update
jupyter agent-from-scratch Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
class State(TypedDict):
messages: Annotated[list[AnyMessage], add_messages]
# Assistant
class Assistant:
def __init__(self, runnable: Runnable):
"""
Initialize the Assistant with a runnable object.
Args:
runnable (Runnable): The runnable instance to invoke.
"""
self.runnable = runnable
def __call__(self, state: State, config: RunnableConfig):
"""
Call method to invoke the LLM and handle its responses.
Re-prompt the assistant if the response is not a tool call or meaningful text.
Args:
state (State): The current state containing messages.
config (RunnableConfig): The configuration for the runnable.
Returns:
dict: The final state containing the updated messages.
"""
while True:
result = self.runnable.invoke(state) # Invoke the LLM
if not result.tool_calls and (
not result.content
or isinstance(result.content, list)
and not result.content[0].get("text")
):
messages = state["messages"] + [("user", "Respond with a real output.")]
state = {**state, "messages": messages}
else:

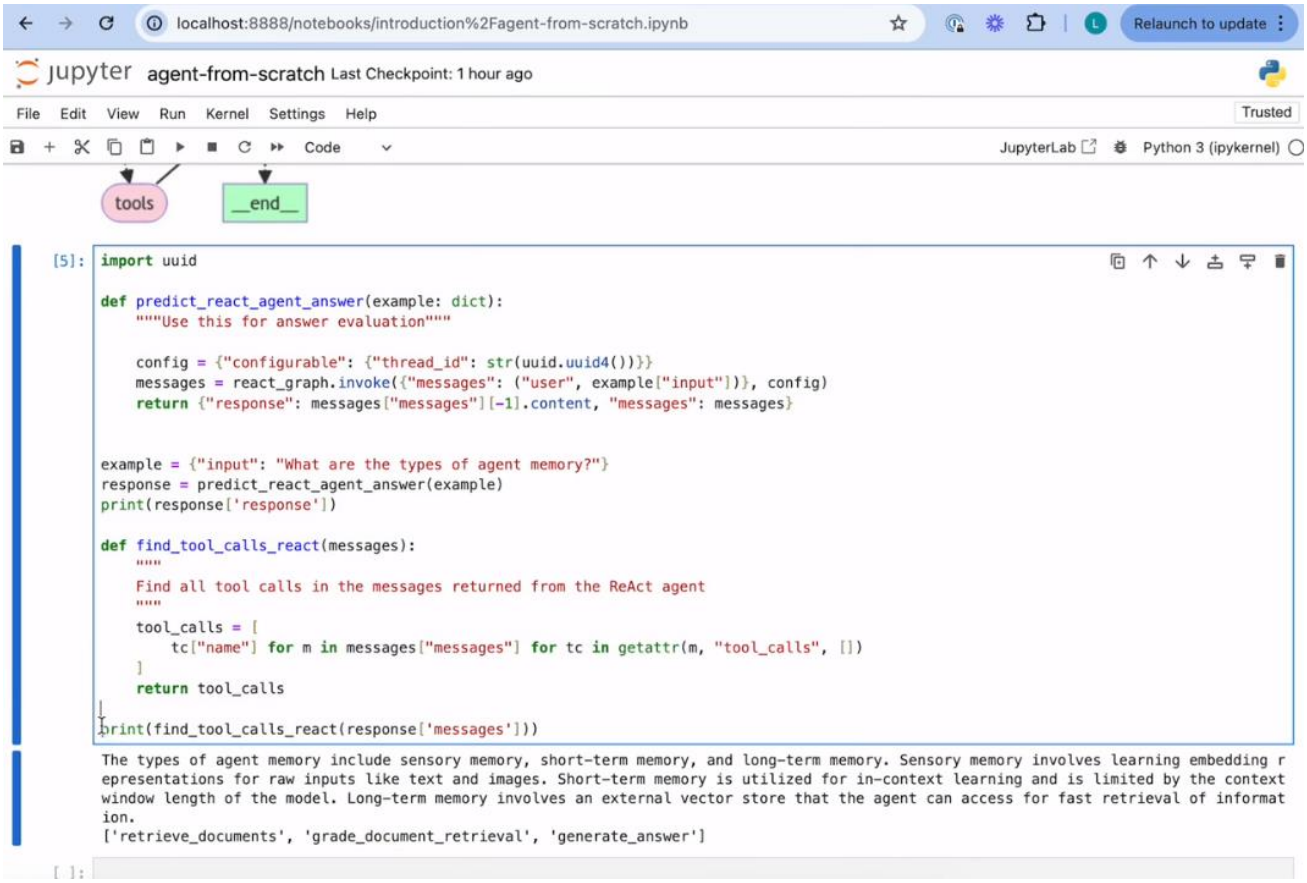

localhost:8888/notebooks/introduction%2Fagent-from-scratch.ipynb
Relaunch to update
jupyter agent-from-scratch Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
result = self.runnable.invoke(state) # Invoke the LLM
if not result.tool_calls and (
not result.content
or isinstance(result.content, list)
and not result.content[0].get("text")
):
messages = state["messages"] + [("user", "Respond with a real output.")]
state = {**state, "messages": messages}
else:
break
return {"messages": result}
# Create the primary assistant prompt template
primary_assistant_prompt = ChatPromptTemplate.from_messages(
[
(
"system",
" You are a helpful assistant tasked with answering user questions using the provided vector store. "
" Use the provided vector store to retrieve documents. Then grade them to ensure they are relevant before answering the ques
),
("placeholder", "{messages}"),
]
)
# Prompt our LLM and bind tools
assistant_runnable = primary_assistant_prompt | llm.bind_tools(tools)


localhost:8888/notebooks/introduction%2Fagent-from-scratch.ipynb
Relaunch to update
jupyter agent-from-scratch Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
[4]:
Agent
from langchain_core.runnables import RunnableLambda
from langchain_core.messages import ToolMessage
from langgraph.prebuilt import ToolNode
def create_tool_node_with_fallback(tools: list) -> dict:
return ToolNode(tools).with_fallbacks(
[RunnableLambda(handle_tool_error)], exception_key="error"
)
def handle_tool_error(state: State) -> dict:
error = state.get("error")
tool_calls = state["messages"][-1].tool_calls
return {
"messages": [
ToolMessage(
content=f"Error: {repr(error)}\n please fix your mistakes.",
tool_call_id=tc["id"],
)
for tc in tool_calls
]
}
from langgraph.checkpoint.sqlite import SqliteSaver
from langgraph.graph import START, END, StateGraph
from langgraph.prebuilt import tools_condition
from IPython.display import Image, display
# Graph
builder = StateGraph(State)
# Define nodes: these do the work
builder.add_node("assistant", Assistant(assistant_runnable))

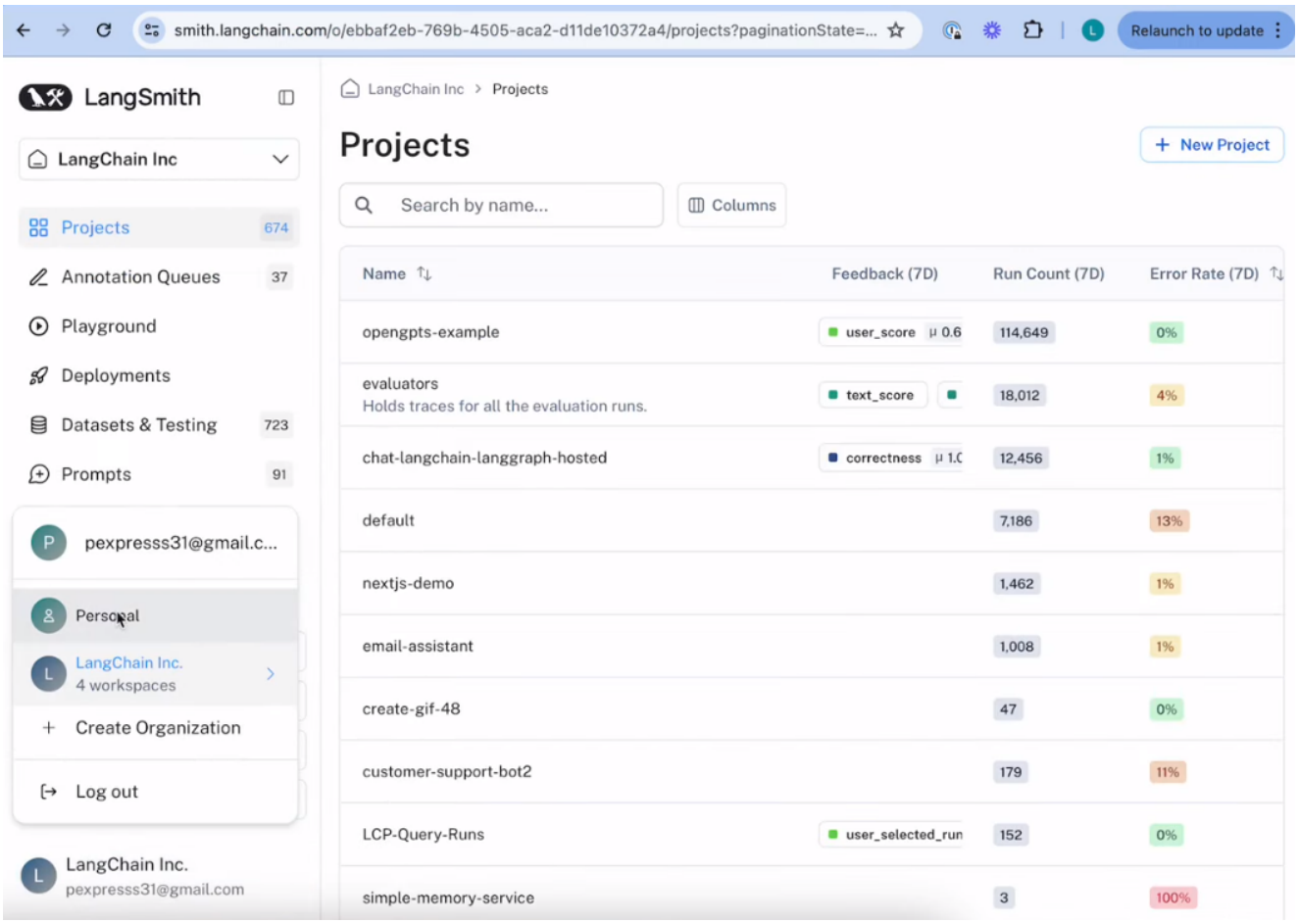

localhost:8888/notebooks/introduction%2Fagent-from-scratch.ipynb
Relaunch to update
jupyter agent-from-scratch Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
tool_call_id=tc["id"],
)
for tc in tool_calls
]
}
from langgraph.checkpoint.sqlite import SqliteSaver
from langgraph.graph import START, END, StateGraph
from langgraph.prebuilt import tools_condition
from IPython.display import Image, display
# Graph
builder = StateGraph(State)
# Define nodes: these do the work
builder.add_node("assistant", Assistant(assistant_runnable))
builder.add_node("tools", create_tool_node_with_fallback(tools))
# Define edges: these determine how the control flow moves
builder.add_edge(START, "assistant")
builder.add_conditional_edges(
"assistant",
If the latest message (result) from assistant is a tool call -> tools_condition routes to tools
If the latest message (result) from assistant is a not a tool call -> tools_condition routes to END
tools_condition,
)
builder.add_edge("tools", "assistant")
# The checkpointer lets the graph persist its state
memory = SqliteSaver.from_conn_string(":memory:")
react_graph = builder.compile(checkpointer=memory)
# Show
display(Image(react_graph.get_graph(xray=True).draw_mermaid_png()))
__start__

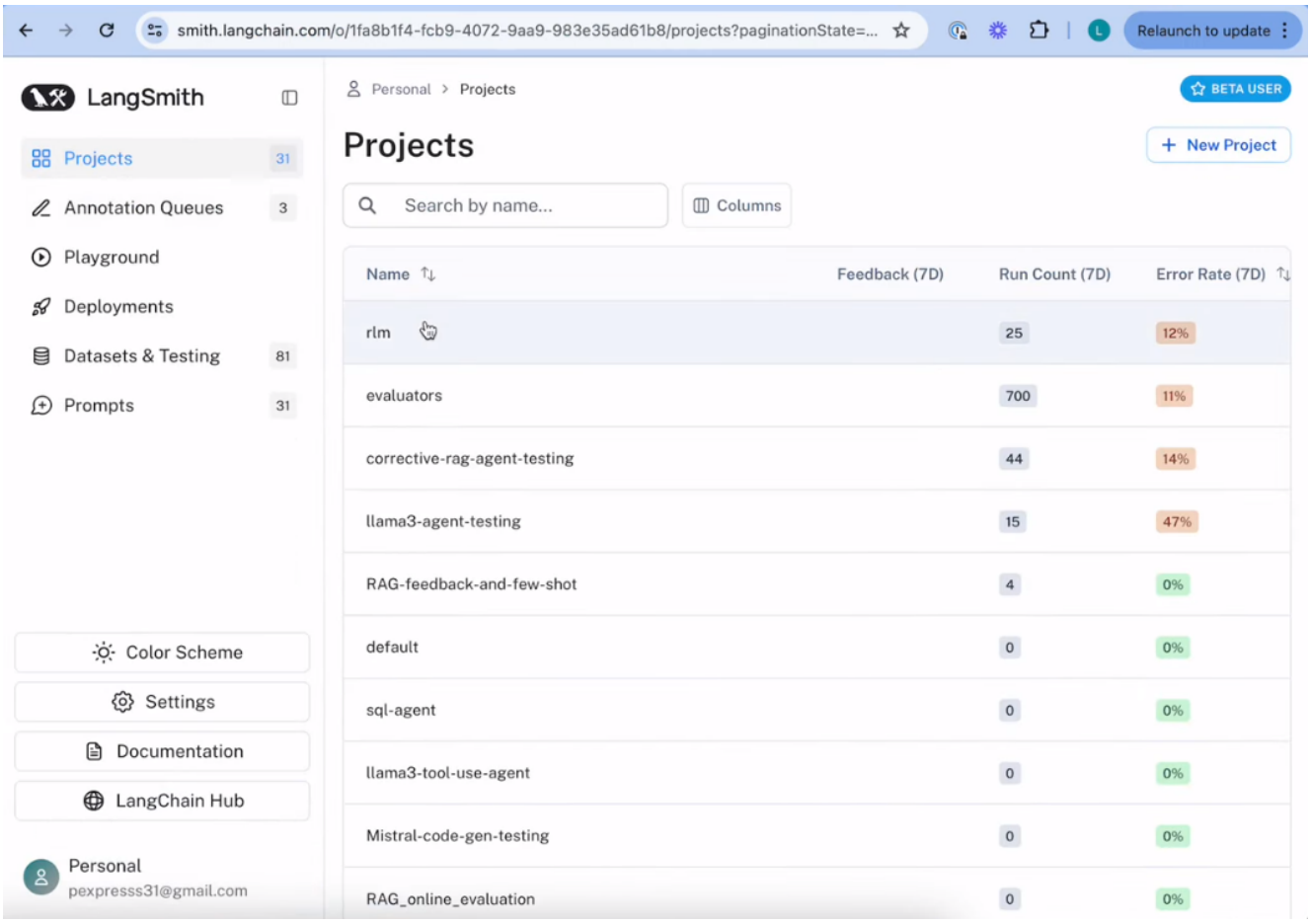

localhost:8888/notebooks/introduction%2Fagent-from-scratch.ipynb
Relaunch to update
jupyter agent-from-scratch Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
builder.add_edge(START, "assistant")
builder.add_conditional_edges(
"assistant",
If the latest message (result) from assistant is a tool call -> tools_condition routes to tools
If the latest message (result) from assistant is a not a tool call -> tools_condition routes to END
tools_condition,
)
builder.add_edge("tools", "assistant")
# The checkpointer lets the graph persist its state
memory = SqliteSaver.from_conn_string(":memory:")
react_graph = builder.compile(checkpointer=memory)
# Show
display(Image(react_graph.get_graph(xray=True).draw_mermaid_png()))
__start__
assistant
tools
__end__
[ ]:

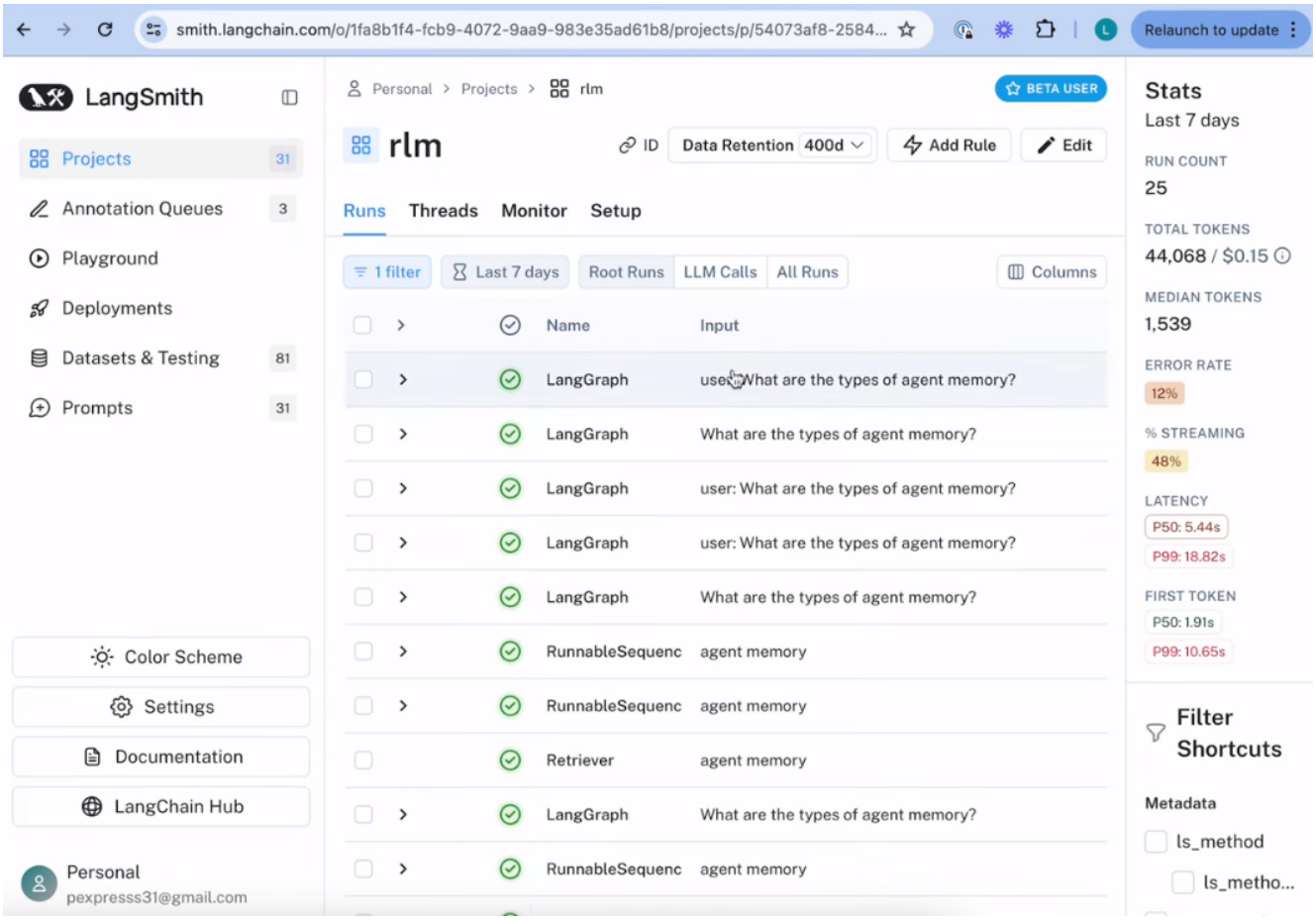


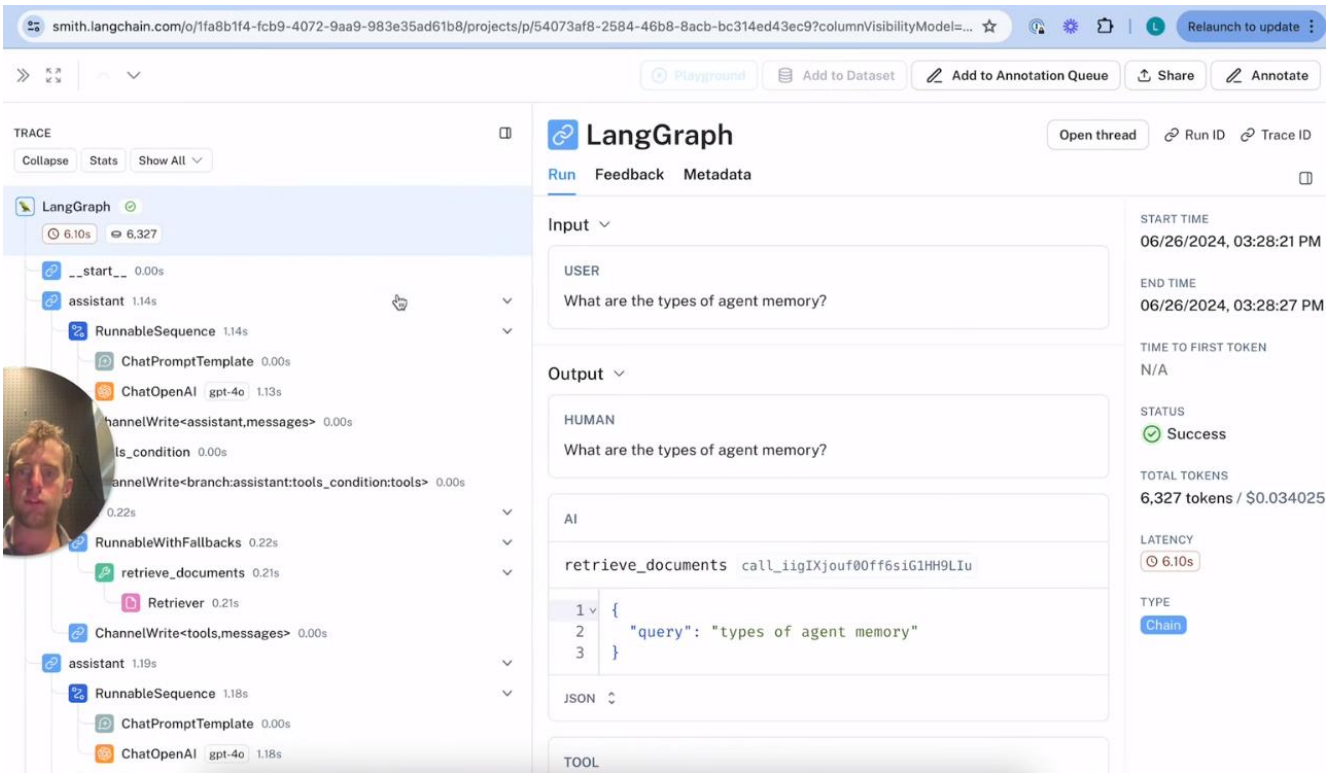


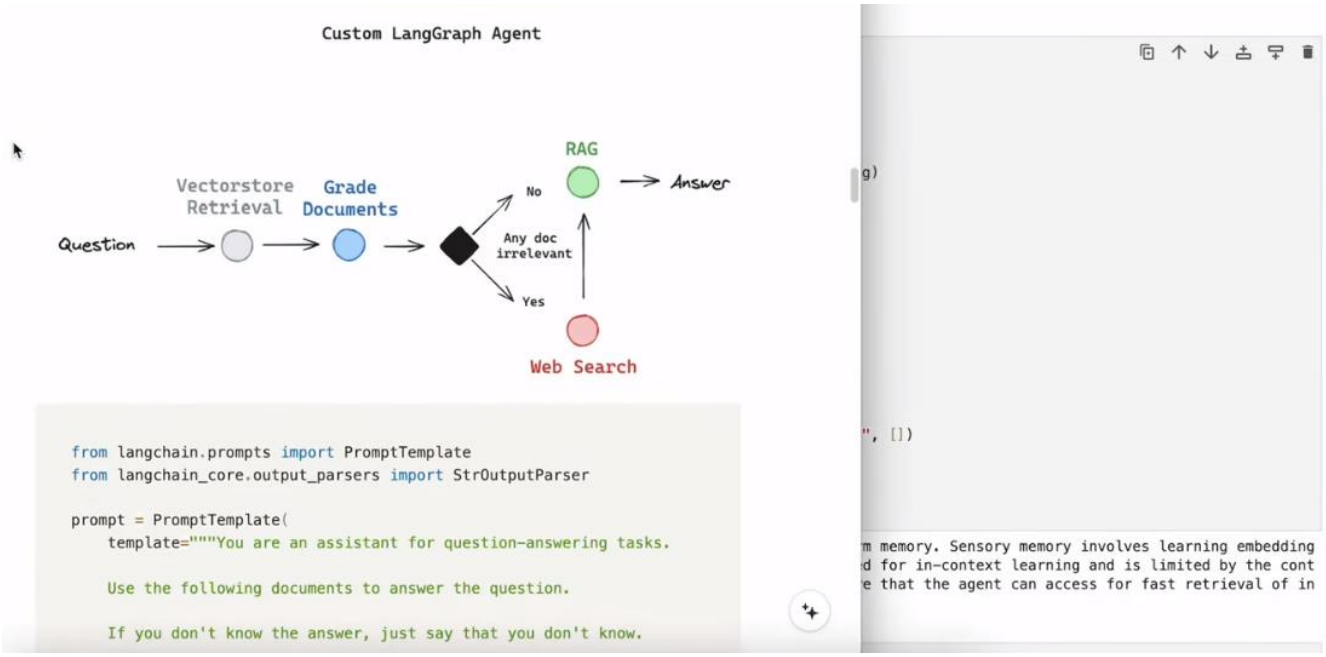
We then run our **ReAct agent** and can see the response and the tool call list as above.


smith.langchain.com/o/ebbaf2eb-769b-4505-aca2-d11de10372a4/projects?paginationState=...
Relaunch to update
LangSmith
LangChain Inc
Projects 674
Annotation Queues 37
Playground
Deployments
Datasets & Testing 723
Prompts 91
pexpresss31@gmail.c...
Personal
LangChain Inc.
4 workspaces
Create Organization
Log out
LangChain Inc.
pexpresss31@gmail.com
LangChain Inc > Projects
Projects
New Project
Search by name...
Columns
Name
Feedback (7D)
Run Count (7D)
Error Rate (7D)
opengpts-example
user_score μ 0.6
114,649
0%
evaluators
Holds traces for all the evaluation runs.
text_score
18,012
4%
chat-langchain-langgraph-hosted
correctness μ 1.0
12,456
1%
default
7,186
13%
nextjs-demo
1,462
1%
email-assistant
1,008
1%
create-gif-48
47
0%
customer-support-bot2
179
11%
LCP-Query-Runs
user_selected_run
152
0%
simple-memory-service
3
100%


smith.langchain.com/o/1fa8b1f4-fcb9-4072-9aa9-983e35ad61b8/projects?paginationState=...
Relaunch to update
LangSmith
Projects 31
Annotation Queues 3
Playground
Deployments
Datasets & Testing 81
Prompts 31
Color Scheme
Settings
Documentation
LangChain Hub
Personal
pexpresss31@gmail.com
Personal > Projects
BETA USER
Projects
New Project
Search by name...
Columns
Name
Feedback (7D)
Run Count (7D)
Error Rate (7D)
rlm
25
12%
evaluators
700
11%
corrective-rag-agent-testing
44
14%
llama3-agent-testing
15
47%
RAG-feedback-and-few-shot
4
0%
default
0
0%
sql-agent
0
0%
llama3-tool-use-agent
0
0%
Mistral-code-gen-testing
0
0%
RAG_online_evaluation
0
0%


smith.langchain.com/o/1fa8b1f4-fcb9-4072-9aa9-983e35ad61b8/projects/p/54073af8-2584...
Relaunch to update
LangSmith
Projects 31
Annotation Queues 3
Playground
Deployments
Datasets & Testing 81
Prompts 31
Color Scheme
Settings
Documentation
LangChain Hub
Personal
pexpresss31@gmail.com
Personal > Projects > rlm
BETA USER
rlm
ID
Data Retention 400d
Add Rule
Edit
Runs Threads Monitor Setup
1 filter
Last 7 days
Root Runs LLM Calls All Runs
Columns
Name Input
LangGraph user: What are the types of agent memory?
LangGraph What are the types of agent memory?
LangGraph user: What are the types of agent memory?
LangGraph user: What are the types of agent memory?
LangGraph What are the types of agent memory?
RunnableSequenc agent memory
RunnableSequenc agent memory
Retriever agent memory
LangGraph What are the types of agent memory?
RunnableSequenc agent memory
Stats
Last 7 days
RUN COUNT
25
TOTAL TOKENS
44,068 / $0.15
MEDIAN TOKENS
1,539
ERROR RATE
12%
% STREAMING
48%
LATENCY
P50: 5.44s
P99: 18.82s
FIRST TOKEN
P50: 1.91s
P99: 10.65s
Filter Shortcuts
Metadata
ls_method
ls_metho...


smith.langchain.com/o/1fa8b1f4-fcb9-4072-9aa9-983e35ad61b8/projects/p/54073af8-2584-46b8-8acb-bc314ed43ec9?columnVisibilityModel=...
Relaunch to update
Playground
Add to Dataset
Add to Annotation Queue
Share
Annotate
TRACE
Collapse Stats Show All
LangGraph 6.10s 6,327
__start__ 0.00s
assistant 1.14s
RunnableSequence 1.14s
ChatPromptTemplate 0.00s
ChatOpenAI gpt-4o 1.13s
hannelWrite<assistant,messages> 0.00s
ls_condition 0.00s
annelWrite<branch:assistant:tools_condition:tools> 0.00s
0.22s
RunnableWithFallbacks 0.22s
retrieve_documents 0.21s
Retriever 0.21s
ChannelWrite<tools,messages> 0.00s
assistant 1.19s
RunnableSequence 1.18s
ChatPromptTemplate 0.00s
ChatOpenAI gpt-4o 1.18s
LangGraph
Open thread
Run ID
Trace ID
Run Feedback Metadata
Input
USER
What are the types of agent memory?
Output
HUMAN
What are the types of agent memory?
AI
retrieve_documents call_iigIXjouf0Off6siG1HH9LIu
{
"query": "types of agent memory"
}
JSON
TOOL
START TIME
06/26/2024, 03:28:21 PM
END TIME
06/26/2024, 03:28:27 PM
TIME TO FIRST TOKEN
N/A
STATUS
Success
TOTAL TOKENS
6,327 tokens / $0.034025
LATENCY
6.10s
TYPE
Chain




```
from langchain.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser

prompt = PromptTemplate(
    template="""You are an assistant for question-answering tasks.

    Use the following documents to answer the question.

    If you don't know the answer, just say that you don't know.
```

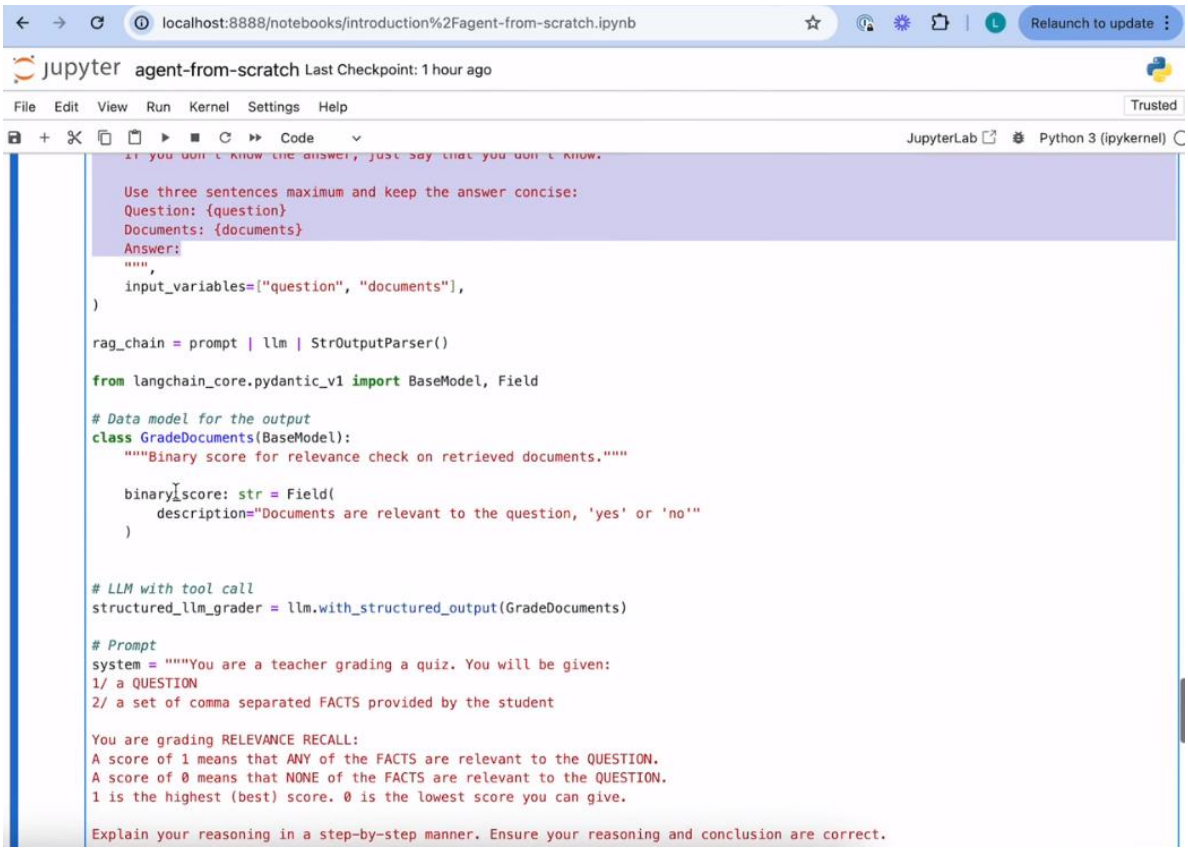
g)

", [])

m memory. Sensory memory involves learning embedding
d for in-context learning and is limited by the cont
e that the agent can access for fast retrieval of in

[6]:

```
from langchain.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser

prompt = PromptTemplate(
    template="""You are an assistant for question-answering tasks.

    Use the following documents to answer the question.

    If you don't know the answer, just say that you don't know.

    Use three sentences maximum and keep the answer concise:
    Question: {question}
    Documents: {documents}
    Answer:
    """,
    input_variables=["question", "documents"],
)

rag_chain = prompt | llm | StrOutputParser()

from langchain_core.pydantic_v1 import BaseModel, Field

# Data model for the output
class GradeDocuments(BaseModel):
    """Binary score for relevance check on retrieved documents."""

    binary_score: str = Field(
```




```
# LLM with tool call
structured_llm_grader = llm.with_structured_output(GradeDocuments)

# Prompt
system = """You are a teacher grading a quiz. You will be given:
1/ a QUESTION
2/ a set of comma separated FACTS provided by the student

You are grading RELEVANCE RECALL:
A score of 1 means that ANY of the FACTS are relevant to the QUESTION.
A score of 0 means that NONE of the FACTS are relevant to the QUESTION.
1 is the highest (best) score. 0 is the lowest score you can give.

Explain your reasoning in a step-by-step manner. Ensure your reasoning and conclusion are correct.

Avoid simply stating the correct answer at the outset."""

grade_prompt = ChatPromptTemplate.from_messages(
    [
        ("system", system),
        ("human", "FACTS: \n\n {documents} \n\n QUESTION: {question}"),
    ]
)

retrieval_grader = grade_prompt | structured_llm_grader
```

```
[ ]:
```

```
retrieval_grader = grade_prompt | structured_llm_grader
```

[7]:
```
### Graph

from IPython.display import Image, display


class GraphState(TypedDict):
    """
    Represents the state of our graph.

    Attributes:
        question: question
        generation: LLM generation
        search: whether to add search
        documents: list of documents
    """

    question: str
    generation: str
    search: str
    documents: List[str]
    steps: List[str]


def retrieve(state):
    """
    Retrieve documents
```

```
def retrieve(state):
    """
    Retrieve documents

    Args:
        state (dict): The current graph state

    Returns:
        state (dict): New key added to state, documents, that contains retrieved documents
    """
```

```
        state (dict): New key added to state, documents, that contains retrieved documents
    """
    question = state["question"]
    documents = retriever.invoke(question)
    steps = state["steps"]
    steps.append("retrieve_documents")
    return {"documents": documents, "question": question, "steps": steps}


def generate(state):
    """
    Generate answer

    Args:
        state (dict): The current graph state

    Returns:
        state (dict): New key added to state, generation, that contains LLM generation
    """

    question = state["question"]
    documents = state["documents"]
    generation = rag_chain.invoke({"documents": documents, "question": question})
    steps = state["steps"]
    steps.append("generate_answer")
    return {
        "documents": documents,
        "question": question,
        "generation": generation,
        "steps": steps,
    }


def grade_documents(state):
```

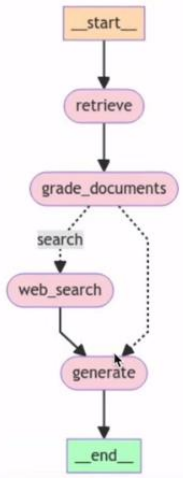
```
def grade_documents(state):
    """
    Determines whether the retrieved documents are relevant to the question.

    Args:
        state (dict): The current graph state

    Returns:
        state (dict): Updates documents key with only filtered relevant documents
    """

    question = state["question"]
    documents = state["documents"]
    steps = state["steps"]
    steps.append("grade_document_retrieval")
    filtered_docs = []
    search = "No"
    for d in documents:
        score = retrieval_grader.invoke(
            {"question": question, "documents": d.page_content}
        )
        grade = score.binary_score
        if grade == "yes":
            filtered_docs.append(d)
        else:
            search = "Yes"
            continue
```

```
        )
        grade = score.binary_score
        if grade == "yes":
            filtered_docs.append(d)
        else:
            search = "Yes"
            continue
    return {
        "documents": filtered_docs,
        "question": question,
        "search": search,
        "steps": steps,
    }


def web_search(state):
    """
    Web search based on the re-phrased question.

    Args:
        state (dict): The current graph state

    Returns:
        state (dict): Updates documents key with appended web results
    """

    question = state["question"]
    documents = state.get("documents", [])
    steps = state["steps"]
```

```
    question = state["question"]
    documents = state.get("documents", [])
    steps = state["steps"]
    steps.append("web_search")
    web_results = web_search_tool.invoke({"query": question})
    documents.extend(
        [
            Document(page_content=d["content"], metadata={"url": d["url"]})
            for d in web_results
        ]
    )
    return {"documents": documents, "question": question, "steps": steps}


def decide_to_generate(state):
    """
    Determines whether to generate an answer, or re-generate a question.

    Args:
        state (dict): The current graph state

    Returns:
        str: Binary decision for next node to call
    """
    search = state["search"]
    if search == "Yes":
        return "search"
    else:
        return "generate"


# Graph
workflow = StateGraph(GraphState)
```

```
# Graph
workflow = StateGraph(GraphState)

# Define the nodes
workflow.add_node("retrieve", retrieve)  # retrieve
workflow.add_node("grade_documents", grade_documents)  # grade documents
workflow.add_node("generate", generate)  # generatae
workflow.add_node("web_search", web_search)  # web search

# Build graph
workflow.set_entry_point("retrieve")
workflow.add_edge("retrieve", "grade_documents")
workflow.add_conditional_edges(
    "grade_documents",
    decide_to_generate,
    {
        "search": "web_search",
        "generate": "generate",
    },
)
workflow.add_edge("web_search", "generate")
```

```
)
workflow.add_edge("web_search", "generate")
workflow.add_edge("generate", END)

custom_graph = workflow.compile()

display(Image(custom_graph.get_graph(xray=True).draw_mermaid_png()))
```



This is what our **custom Langgraph agent** looks like. The sequence of calls is stated much clearly. Let's run it as below

```
[8]: def predict_custom_agent_answer(example: dict):
         config = {"configurable": {"thread_id": str(uuid.uuid4())}}
         state_dict = custom_graph.invoke(
             {"question": example["input"], "steps": []}, config
         )
         return {"response": state_dict["generation"], "steps": state_dict["steps"]}


     example = {"input": "What are the types of agent memory?"}
     response = predict_custom_agent_answer(example)
     response
```

```
[8]: {'response': 'The types of agent memory are short-term memory and long-term memory. Short-term memory involves in-context l
     earning using prompt templates and relevant variables. Long-term memory allows the agent to retain and recall information o
     ver extended periods, often leveraging an external vector store for fast retrieval.',
      'steps': ['retrieve_documents',
       'grade_document_retrieval',
       'web_search',
       'generate_answer']}
```

```
[ ]:
```

Next, let us see how to do simple **evaluations** of the ReAct and Custom Langgraph agents as below


localhost:8888/notebooks/introduction%2Fagent-from-scratch.ipynb
Relaunch to update
jupyter agent-from-scratch Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
[ ]:
from langsmith import Client
client = Client()
DATASET
# Create a dataset
examples = [
("How does the ReAct agent use self-reflection? ",
"ReAct integrates reasoning and acting, performing actions - such tools like Wikipedia search API - and then observ
),
("What are the types of biases that can arise with few-shot prompting?",
"The biases that can arise with few-shot prompting include (1) Majority label bias, (2) Recency bias, and (3) Commo
),
("What are five types of adversarial attacks?",
"Five types of adversarial attacks are (1) Token manipulation, (2) Gradient based attack, (3) Jailbreak prompting,
),
("Who did the Chicago Bears draft first in the 2024 NFL draft"?",
"The Chicago Bears drafted Caleb Williams first in the 2024 NFL draft.",
),
("Who won the 2024 NBA finals?", "The Boston Celtics on the 2024 NBA finals"),
]
# Save it
dataset_name = "Corrective RAG Agent QA"
# Save it
dataset_name = "Corrective RAG Agent QA"
if not client.has_dataset(dataset_name=dataset_name):
dataset = client.create_dataset(dataset_name=dataset_name)
inputs, outputs = zip(
*[({"input": text}, {"output": label}) for text, label in examples]
)
client.create_examples(inputs=inputs, outputs=outputs, dataset_id=dataset.id)
from langchain import hub
from langchain_openai import ChatOpenAI
ANSWER EVAL
# Grade prompt
grade_prompt_answer_accuracy = hub.pull("langchain-ai/rag-answer-vs-reference")
# Grade prompt
grade_prompt_answer_accuracy = hub.pull("langchain-ai/rag-answer-vs-reference")
def answer_evaluator(run, example) -> dict:
"""
A simple evaluator for RAG answer accuracy
"""
# Get the question, the ground truth reference answer, RAG chain answer prediction
input_question = example.inputs["input"]
reference = example.outputs["output"]
prediction = run.outputs["response"]
# Define an LLM grader
llm = ChatOpenAI(model="gpt-4o", temperature=0)
answer_grader = grade_prompt_answer_accuracy | llm
# Run evaluator
score = answer_grader.invoke(
{
"question": input_question,
"correct_answer": reference,
"student_answer": prediction,
}
)
score = score["Score"]
return {"key": "answer_v_reference_score", "score": score}
from langsmith.schemas import Example, Run
REASONING

```
### REASONING

# Reasoning traces that we expect the agents to take
expected_trajectory_1 = [
    "retrieve_documents",
    "grade_document_retrieval",
    "web_search",
    "generate_answer",
]
expected_trajectory_2 = [
    "retrieve_documents",
    "grade_document_retrieval",
    "generate_answer",
]

def check_trajectory_react(root_run: Run, example: Example) -> dict:
    """
    Check if all expected tools are called in exact order and without any additional tool calls.
    """
    messages = root_run.outputs["messages"]
    tool_calls = find_tool_calls_react(messages)
    print(f"Tool calls ReAct agent: {tool_calls}")
    if tool_calls == expected_trajectory_1 or tool_calls == expected_trajectory_2:
        score = 1
    else:
        score = 0

    return {"score": int(score), "key": "tool_calls_in_exact_order"}


def check_trajectory_custom(root_run: Run, example: Example) -> dict:
```

```
def check_trajectory_react(root_run: Run, example: Example) -> dict:
    """
    Check if all expected tools are called in exact order and without any additional tool calls.
    """
    messages = root_run.outputs["messages"]
    tool_calls = find_tool_calls_react(messages)
    print(f"Tool calls ReAct agent: {tool_calls}")
    if tool_calls == expected_trajectory_1 or tool_calls == expected_trajectory_2:
        score = 1
    else:
        score = 0

    return {"score": int(score), "key": "tool_calls_in_exact_order"}

def check_trajectory_custom(root_run: Run, example: Example) -> dict:
    """
    Check if all expected tools are called in exact order and without any additional tool calls.
    """
    tool_calls = root_run.outputs["steps"]
    print(f"Tool calls custom agent: {tool_calls}")
    if tool_calls == expected_trajectory_1 or tool_calls == expected_trajectory_2:
        score = 1
    else:
        score = 0

    return {"score": int(score), "key": "tool_calls_in_exact_order"}
```

We can then run our evaluations using the code below

How can we test our agent performance?

```python
from langsmith.evaluation import evaluate

experiment_prefix = f"react-agent-{model_tested}"
experiment_results = evaluate(
    predict_react_agent_answer,
    data=dataset_name,
    evaluators=[answer_evaluator, check_trajectory_react],
    experiment_prefix=experiment_prefix + "-answer-and-tool-use",
    num_repetitions=3,
    metadata={"version": metadata},
)

experiment_prefix = f"custom-agent-{model_tested}"
experiment_results = evaluate(
    predict_custom_agent_answer,
    data=dataset_name,
    evaluators=[answer_evaluator, check_trajectory_custom],
    experiment_prefix=experiment_prefix + "-answer-and-tool-use",
    num_repetitions=3,
    metadata={"version": metadata},
)
```

```
View the evaluation results for experiment: 'react-agent-gpt-4o-answer-and-tool-use-53a6181f' at:
https://smith.langchain.com/o/1fa8b1f4-fcb9-4072-9aa9-983e35ad61b8/datasets/17a04a67-8279-4b0c-a11c-e63f21955559/compare?selectedSessions=4c26213d-3228-4980-853e-2013fb05da7e

0/? [00:00<?, ?it/s]
```

localhost:8888/notebooks/introduction%2Fagent-from-scratch.ipynb

jupyter agent-from-scratch Last Checkpoint: 1 hour ago

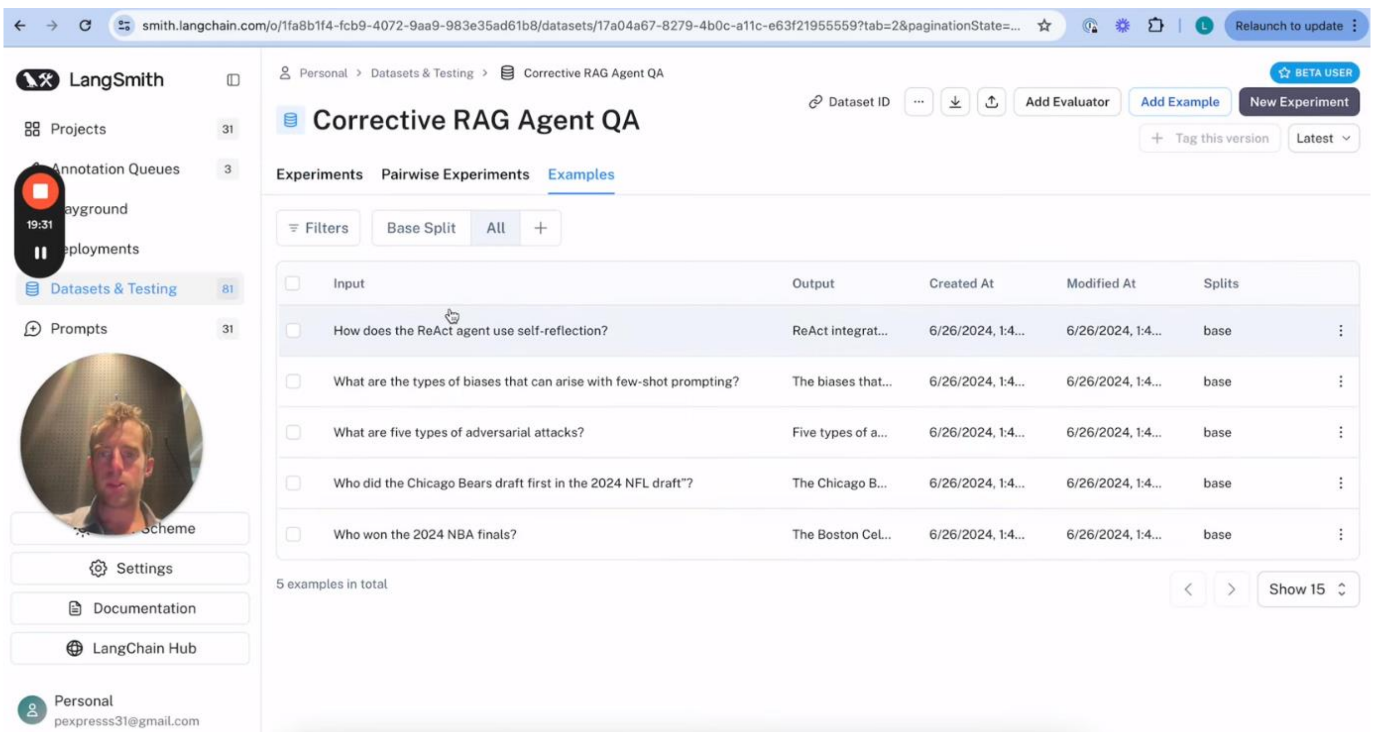
File Edit View Run Kernel Settings Help Trusted

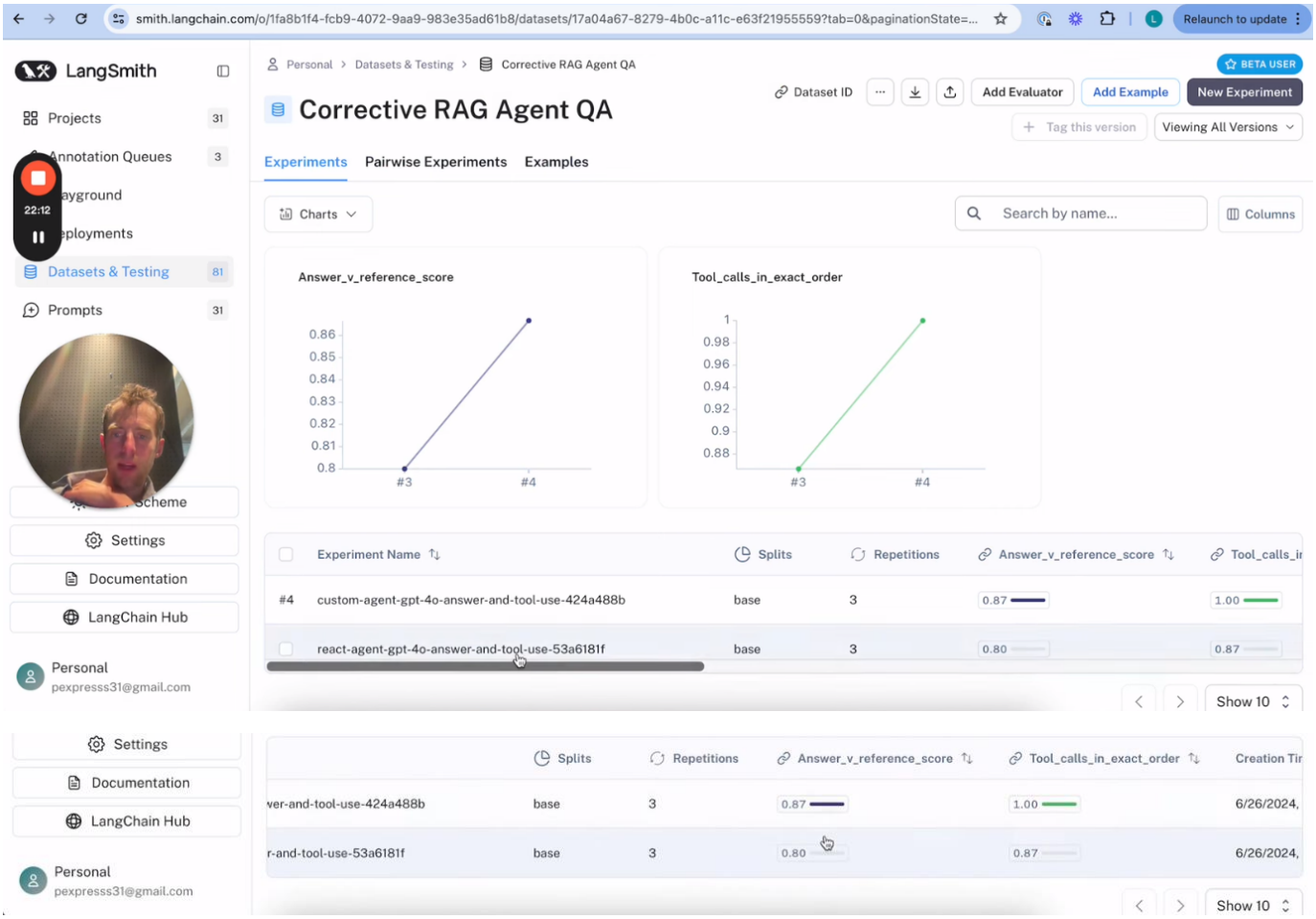
Code JupyterLab Python 3 (ipykernel)

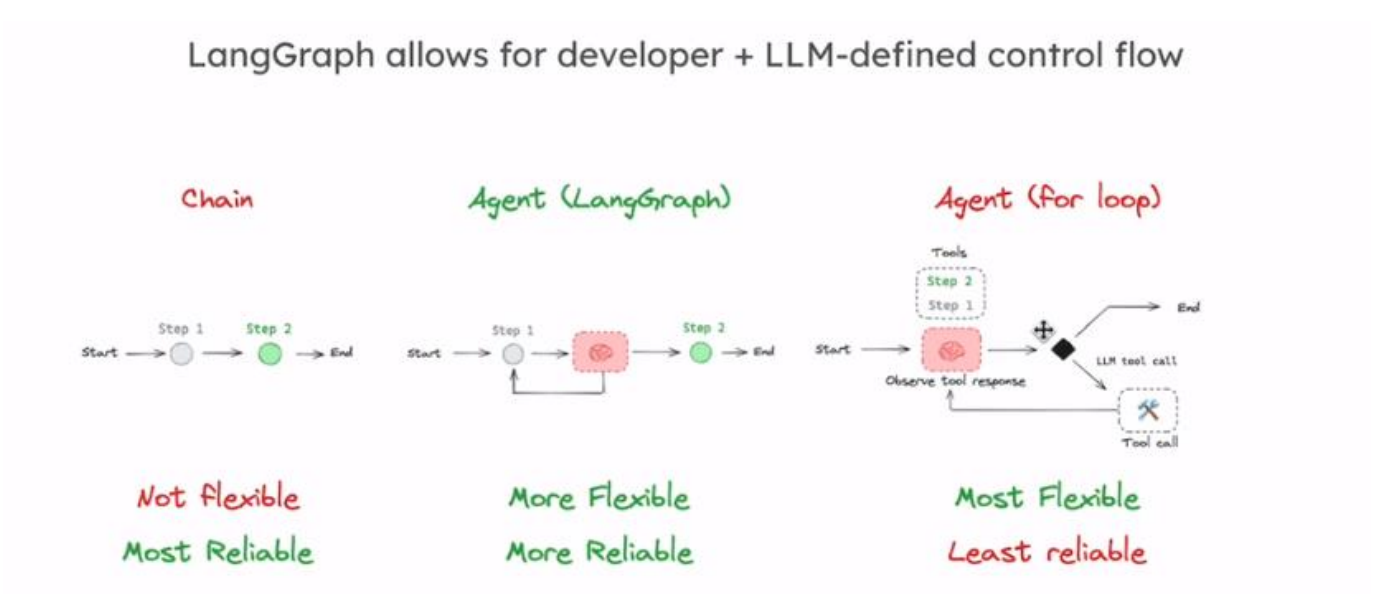
```
https://smith.langchain.com/o/1fa8b1f4-fcb9-4072-9aa9-983e35ad61b8/datasets/17a04a67-8279-4b0c-a11c-e63f21955559/compare?selectedSessions=4c26213d-3228-4980-853e-2013fb05da7e

15/? [00:11<00:00, 4.83it/s]
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'web_search', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'web_search', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'generate_answer']
View the evaluation results for experiment: 'custom-agent-gpt-4o-answer-and-tool-use-424a488b' at:
https://smith.langchain.com/o/1fa8b1f4-fcb9-4072-9aa9-983e35ad61b8/datasets/17a04a67-8279-4b0c-a11c-e63f21955559/compare?selectedSessions=df118d7a-1b9a-4311-bf22-ef7d265650c8

15/? [00:09<00:00, 5.59it/s]
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
```

```
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'web_search', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls ReAct agent: ['retrieve_documents', 'grade_document_retrieval', 'generate_answer']
View the evaluation results for experiment: 'custom-agent-gpt-4o-answer-and-tool-use-424a488b' at:
https://smith.langchain.com/o/1fa8b1f4-fcb9-4072-9aa9-983e35ad61b8/datasets/17a04a67-8279-4b0c-a11c-e63f21955559/compare?selectedSessions=df1
18d7a-1b9a-4311-bf22-ef7d265650c8

15/? [00:09<00:00,  5.59it/s]
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
Tool calls custom agent: ['retrieve_documents', 'grade_document_retrieval', 'web_search', 'generate_answer']
```

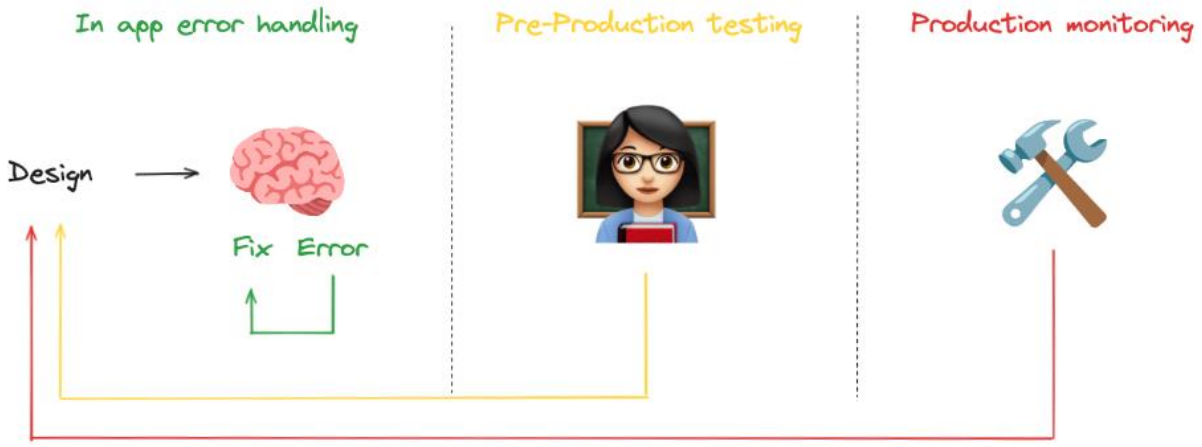
We can see that the bottom custom Langgraph agent control flow always followed the sequence unlike the ReAct agent




smith.langchain.com/o/1fa8b1f4-fcb9-4072-9aa9-983e35ad61b8/datasets/17a04a67-8279-4b0c-a11c-e63f21955559?tab=0&paginationState=...
Relaunch to update
LangSmith
Projects 31
Annotation Queues 3
Datasets & Testing 81
Prompts 31
Settings
Documentation
LangChain Hub
Personal
pexpresss31@gmail.com
Personal > Datasets & Testing > Corrective RAG Agent QA
BETA USER
Corrective RAG Agent QA
Dataset ID
Add Evaluator
Add Example
New Experiment
Tag this version
Viewing All Versions
Experiments
Pairwise Experiments
Examples
Charts
Search by name...
Columns
Answer_v_reference_score
Tool_calls_in_exact_order
Experiment Name
Splits
Repetitions
Answer_v_reference_score
Tool_calls_ir
#4 custom-agent-gpt-4o-answer-and-tool-use-424a488b base 3 0.87 1.00
react-agent-gpt-4o-answer-and-tool-use-53a6181f base 3 0.80 0.87
Show 10
Creation Tir
ver-and-tool-use-424a488b base 3 0.87 1.00 6/26/2024,
r-and-tool-use-53a6181f base 3 0.80 0.87 6/26/2024,

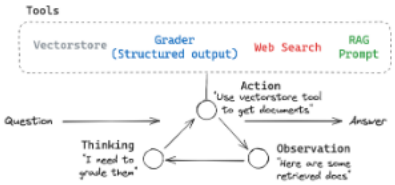

LangGraph allows for developer + LLM-defined control flow
Chain
Agent (LangGraph)
Agent (for loop)
Start
Step 1
Step 2
End
Tools
LLM tool call
Observe tool response
Tool call
Not flexible
Most Reliable
More Flexible
More Reliable
Most Flexible
Least reliable

Three types of general testing loops

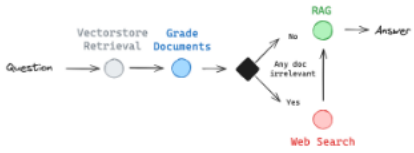


Let's build Corrective RAG agent in two different ways

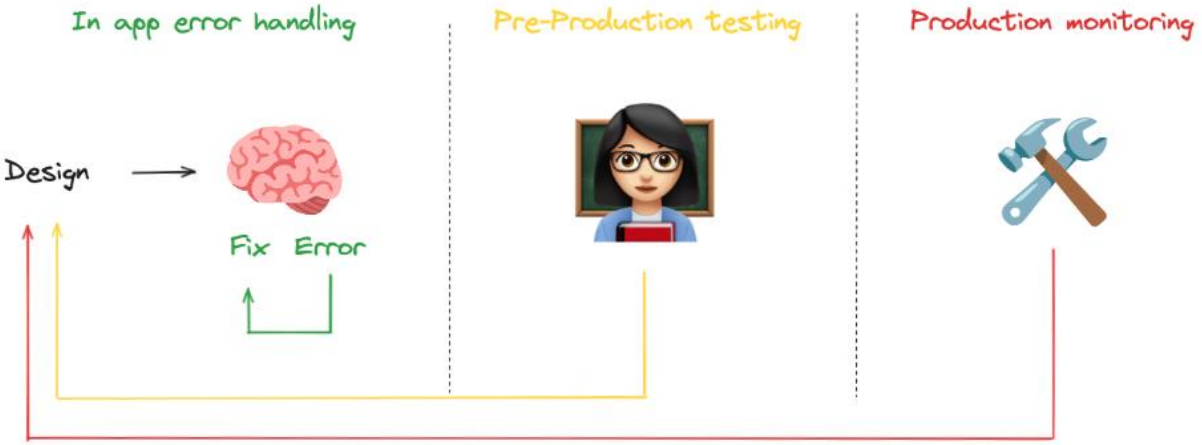
ReAct Agent



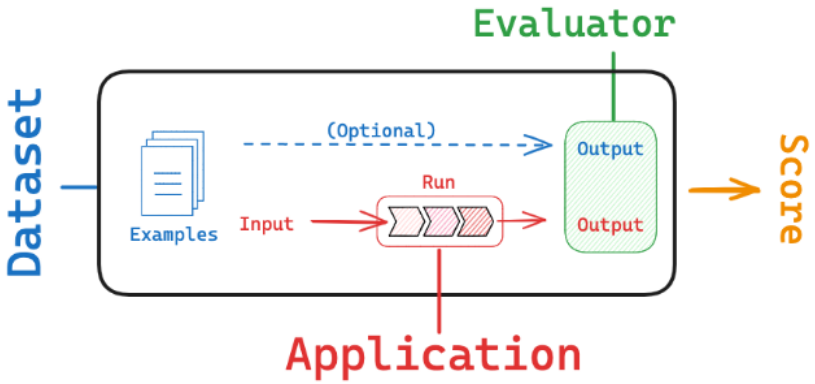
Custom LangGraph Agent



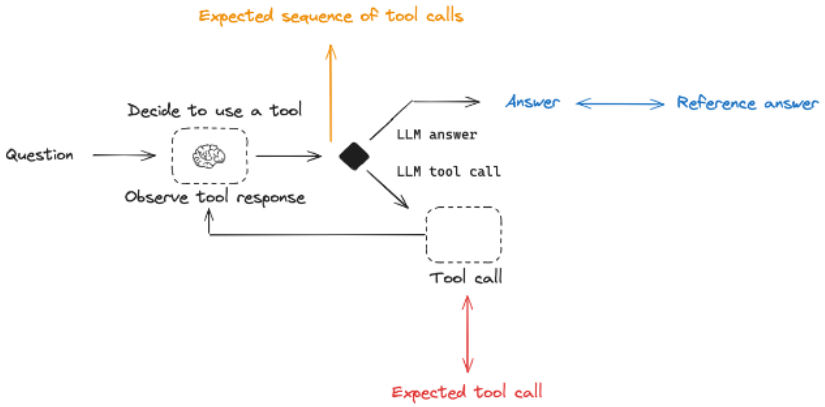
Now, let's use LangSmith to test both of our agents



Build systems to catch errors before deployment: LangSmith



LangSmith testing use-cases: Agents



LangSmith allows you to compare tests across experiments

Experiment Name	Splits	Repetitions	Answer: Answer_v_reference_score	Tool use: Tool_calls_in_exact_order
#40 custom-agent-llama3-8b-answer-and-tool-use-d6006159	base	3	0.60	1.00
#39 custom-agent-firefunction-v2-answer-and-tool-use-9ff2990a	base	3	0.80	1.00
#38 react-agent-firefunction-v2-answer-and-tool-use-8d22a2d1	base	3	0.60	0.40
#37 custom-agent-gpt-4o-answer-and-tool-use-124beb12	base	3	0.80	1.00
#36 react-agent-gpt-4o-answer-and-tool-use-b217ea0a	base	3	0.80	0.67

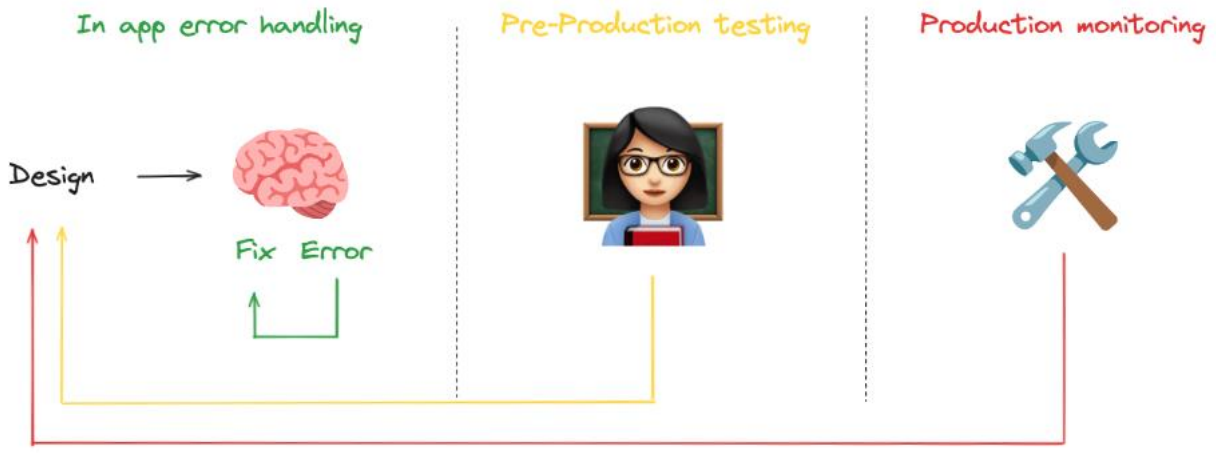
Insight: Custom LangGraph agent has consistent reasoning, even locally!

	Experiment Name	Splits	Repetitions	Answer_v_reference_score	Tool_calls_in_exact_order
#40	custom-agent-llama3-8b-answer-and-tool-use-d6006159	base	3	0.60	1.00
#39	custom-agent-firefunction-v2-answer-and-tool-use-9ff2990a	base	3	0.80	1.00
#38	react-agent-firefunction-v2-answer-and-tool-use-8d22a2d1	base	3	0.60	0.40
#37	custom-agent-gpt-4o-answer-and-tool-use-124beb12	base	3	0.80	1.00
#36	react-agent-gpt-4o-answer-and-tool-use-b217ea0a	base	3	0.80	0.67

Insight: ReAct is less predictable! And degraded going gpt4-o to llama3-70b

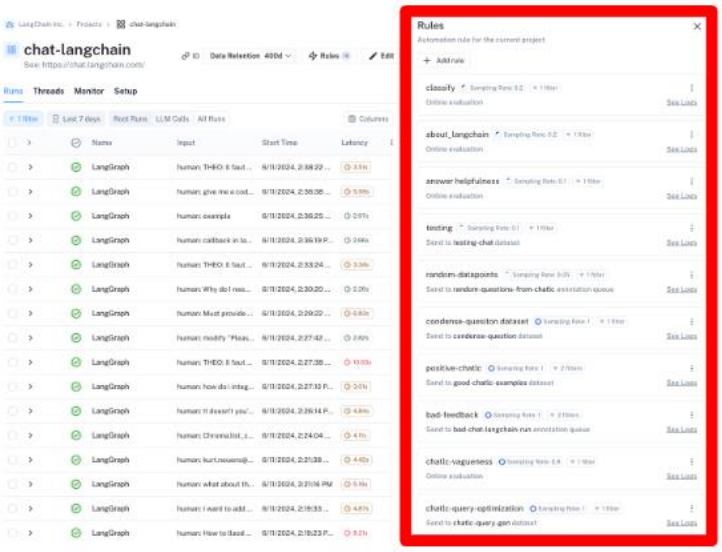
	Experiment Name	Splits	Repetitions	Answer_v_reference_score	Tool_calls_in_exact_order
#40	custom-agent-llama3-8b-answer-and-tool-use-d6006159	base	3	0.60	1.00
#39	custom-agent-firefunction-v2-answer-and-tool-use-9ff2990a	base	3	0.80	1.00
#38	react-agent-firefunction-v2-answer-and-tool-use-8d22a2d1	base	3	0.60	0.40
#37	custom-agent-gpt-4o-answer-and-tool-use-124beb12	base	3	0.80	1.00
#36	react-agent-gpt-4o-answer-and-tool-use-b217ea0a	base	3	0.80	0.67

Now, we deploy!



LangGraph Cloud

Build systems to catch errors in deployment: LangSmith



The model is not the moat

The model isn't the product; the system around it is

For teams that aren't building models, the rapid pace of innovation is a boon as they migrate from one SOTA model to the next, chasing gains in context size, reasoning capability, and price-to-value to build better and better products.

This progress is as exciting as it is predictable. Taken together, this means models are likely to be the least durable component in the system.

Instead, focus your efforts on what's going to provide lasting value, such as:

- Evaluation chassis: To reliably measure performance on your task across models
- Guardrails: To prevent undesired outputs no matter the model
- Caching: To reduce latency and cost by avoiding the model altogether
- Data flywheel: To power the iterative improvement of everything above

These components create a thicker moat of product quality than raw model capabilities.

https://www.oreilly.com/radar/what-we-learned-from-a-year-of-building-with-llms-part-iii-strategy/

Three types of feedback loops to speed up development