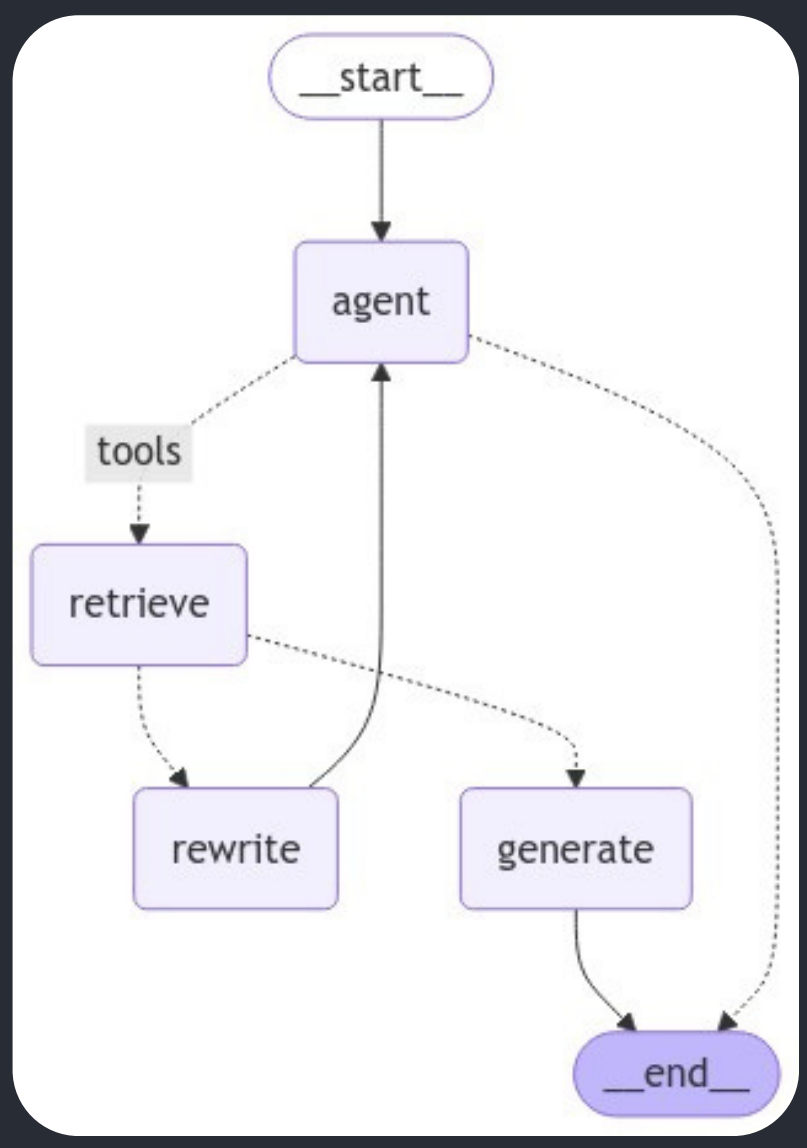
# LangGraph

Building language agents as graphs

## 1. Agentic **RAG**

```python
from langgraph.graph import END, StateGraph, START
from langgraph.prebuilt import ToolNode

workflow = StateGraph(AgentState)

workflow.add_node("agent", agent)  # agent
retrieve = ToolNode([retriever_tool])
workflow.add_node("retrieve", retrieve)  # retrieval
workflow.add_node("rewrite", rewrite)
workflow.add_node(
    "generate", generate
)

workflow.add_edge(START, "agent")

# Decide whether to retrieve
workflow.add_conditional_edges(
    "agent",
    # Assess agent decision
    tools_condition,
    {
        "tools": "retrieve",
        END: END,
    },
)

workflow.add_conditional_edges(
    "retrieve",
    # Assess agent decision
    grade_documents,
)
workflow.add_edge("generate", END)
workflow.add_edge("rewrite", "agent")

graph = workflow.compile()
```



# LangGraph

Building language agents as graphs

## 2. **Multi-Agent** Network

```
workflow = StateGraph(AgentState)

workflow.add_node("Researcher", research_node)
workflow.add_node("chart_generator", chart_node)
workflow.add_node("call_tool", tool_node)

workflow.add_conditional_edges(
    "Researcher",
    router,
    {"continue": "chart_generator", "call_tool": "call_tool", END: END},
)
workflow.add_conditional_edges(
    "chart_generator",
    router,
    {"continue": "Researcher", "call_tool": "call_tool", END: END},
)

workflow.add_conditional_edges(
    "call_tool",
    lambda x: x["sender"],
    {
        "Researcher": "Researcher",
        "chart_generator": "chart_generator",
    },
)
workflow.add_edge(START, "Researcher")
graph = workflow.compile()
```

# LangGraph

Building language agents as graphs

## 3. Storing **Memories**

```python
from langgraph.store.memory import InMemoryStore

store = InMemoryStore()
user_id = "my-user"
application_context = "chitchat"
namespace = (user_id, application_context)
store.put(namespace, "a-memory", {"rules": ["User likes short, direct language", "User only speaks
English & python"], "my-key": "my-value"})
# get the "memory" by ID
item = store.get(namespace, "a-memory")
# list "memories" within this namespace, filtering on content equivalence
items = store.search(namespace, filter={"my-key": "my-value"})
```