arize

AI Agents Masterclass:

Part One

Jerry Liu
LlamaIndex
Co-Founder and CEO

Jason Lopatecki
Arize AI
Co-Founder and CEO

Our Upcoming Agent Series

- Session 6: **Agent Masterclass #2** (10/15 @ 12pm PT)
  - Featuring Chi Wang from Autogen

🏅 **Agents Certification** 🏅

https://bit.ly/agents-course

Agenda

- What are workflows?
- How does workflows compare to other approaches?
- Why the event-driven architecture?
- Context vs state
- How do multi-agent architectures fit in?

What Are Agents?

LLM AGENT IS... using an LLM to iterate on a task, make decisions, do analysis or control execution flow of your application.

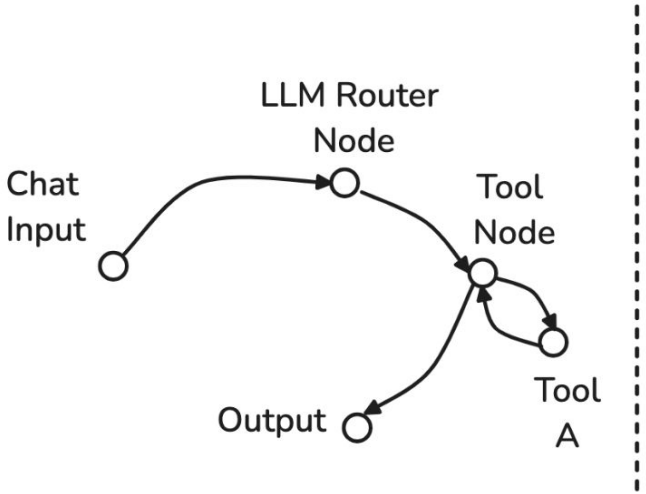
Simplest form of Agent:

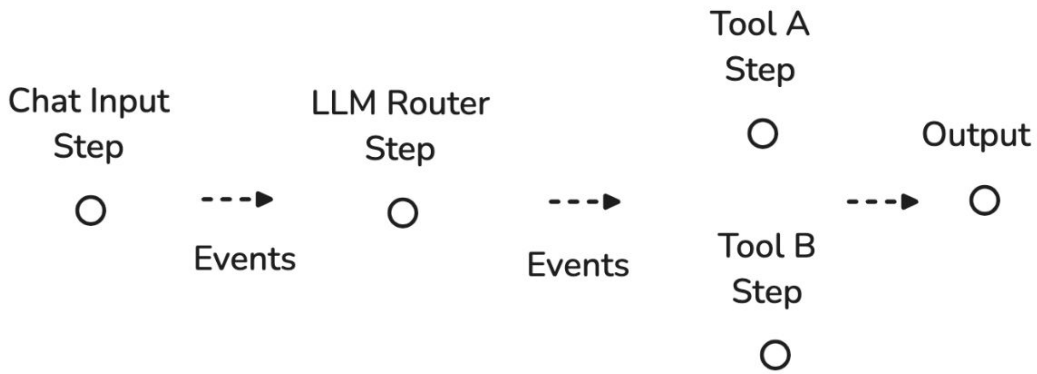
- Using an LLM router to determine what to do based on input data & user intent
- Using an LLM to accomplish a skill or a task
- Combining the above for an iterative workflow

What is not an agent:

- *A single LLM Call*

Graph vs Event Based Agent Architecture

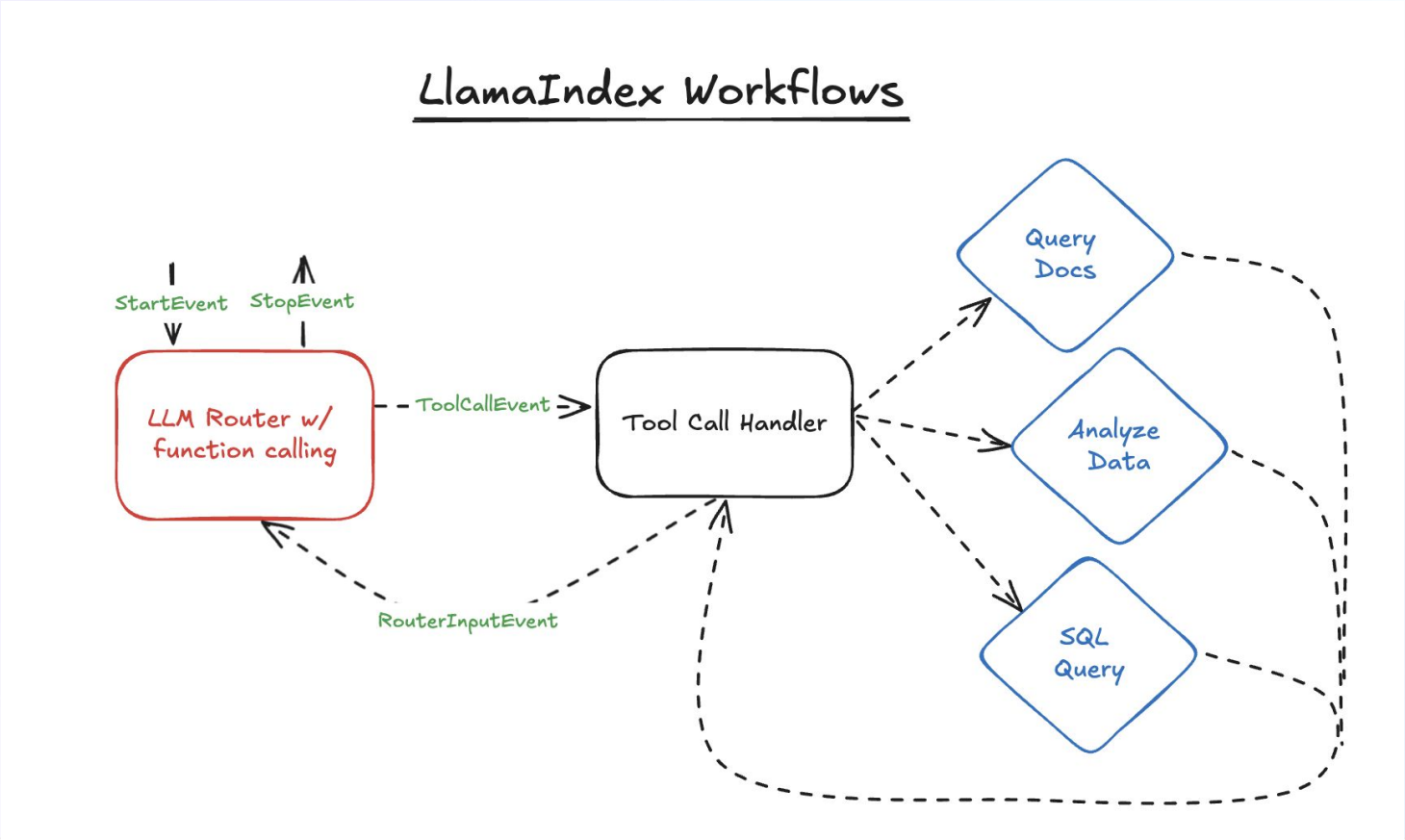




Agent with Graph

Workflows

Workflows Example - Data Analysis Agent



LlamaIndex Workflows - Setup the Workflow

```python
class AgentFlow(Workflow):
    def __init__(self, llm, timeout=300):
        super().__init__(timeout=timeout)
        self.llm = llm
        self.memory = ChatMemoryBuffer(token_limit=1000).from_defaults(llm=llm)
        self.tools = []
        for func in skill_map.get_function_list():
            self.tools.append(FunctionTool(
                skill_map.get_function_callable_by_name(func),
                metadata=ToolMetadata(name=func, description=skill_map.get_function_description_by_name(func))))

    @step
    async def prepare_agent(self, ev: StartEvent) -> RouterInputEvent:
        user_input = ev.input
        user_msg = ChatMessage(role="user", content=user_input)
        self.memory.put(user_msg)

        chat_history = self.memory.get()
        return RouterInputEvent(input=chat_history)
```

LlamaIndex Workflows - Setup LLM Router

```python
@step
async def router(self, ev: RouterInputEvent) -> ToolCallEvent | StopEvent:
    messages = ev.input

    if not any(isinstance(message, dict) and message.get("role") == "system" for message in messages):
        system_prompt = ChatMessage(role="system", content=SYSTEM_PROMPT)
        messages.insert(0, system_prompt)

    with using_prompt_template(template=SYSTEM_PROMPT, version="v0.1"):
        response = await self.llm.achat_with_tools(
            model="gpt-4o",
            messages=messages,
            tools=self.tools,
        )

    self.memory.put(response.message)

    tool_calls = self.llm.get_tool_calls_from_response(
        response, error_on_no_tool_call=False
    )
    if tool_calls:
        return ToolCallEvent(tool_calls=tool_calls)
    else:
        return StopEvent(result=response.message.content)
```

Highlights:

1. Asynchronous LLM call
2. Event-based traversal

```python
class ToolCallEvent(Event):
    tool_calls: list[ToolSelection]

class RouterInputEvent(Event):
    input: list[ChatMessage]
```

LlamaIndex Workflows - Setup Tool Call Handler

```python
@step
async def tool_call_handler(self, ev: ToolCallEvent) -> RouterInputEvent:
    tool_calls = ev.tool_calls

    for tool_call in tool_calls:
        function_name = tool_call.tool_name
        arguments = tool_call.tool_kwargs
        if "input" in arguments:
            arguments["prompt"] = arguments.pop("input")

        try:
            function_callable = skill_map.get_function_callable_by_name(function_name)
        except KeyError:
            function_result = "Error: Unknown function call"

        function_result = function_callable(arguments)
        message = ChatMessage(role="tool",
                              content=function_result,
                              additional_kwargs={"tool_call_id": tool_call.tool_id})

        self.memory.put(message)

    return RouterInputEvent(input=self.memory.get())
```

LangGraph vs Workflows

```python
tools = [generate_and_run_sql_query, data_analyzer]
model = ChatOpenAI(model="gpt-4o", temperature=0).bind_tools(tools)

def should_continue(state: MessagesState):
    messages = state["messages"]
    last_message = messages[-1]
    if last_message.tool_calls:
        return "tools"
    return END

def call_model(state: MessagesState):
    messages = state["messages"]
    response = model.invoke(messages)
    return {"messages": [response]}

def create_agent_graph():
    workflow = StateGraph(MessagesState)

    tool_node = ToolNode(tools)
    workflow.add_node("agent", call_model)
    workflow.add_node("tools", tool_node)

    workflow.add_edge(START, "agent")
    workflow.add_conditional_edges(
        "agent",
        should_continue,
    )
    workflow.add_edge("tools", "agent")

    checkpointer = MemorySaver()
    app = workflow.compile(checkpointer=checkpointer)
    return app
```

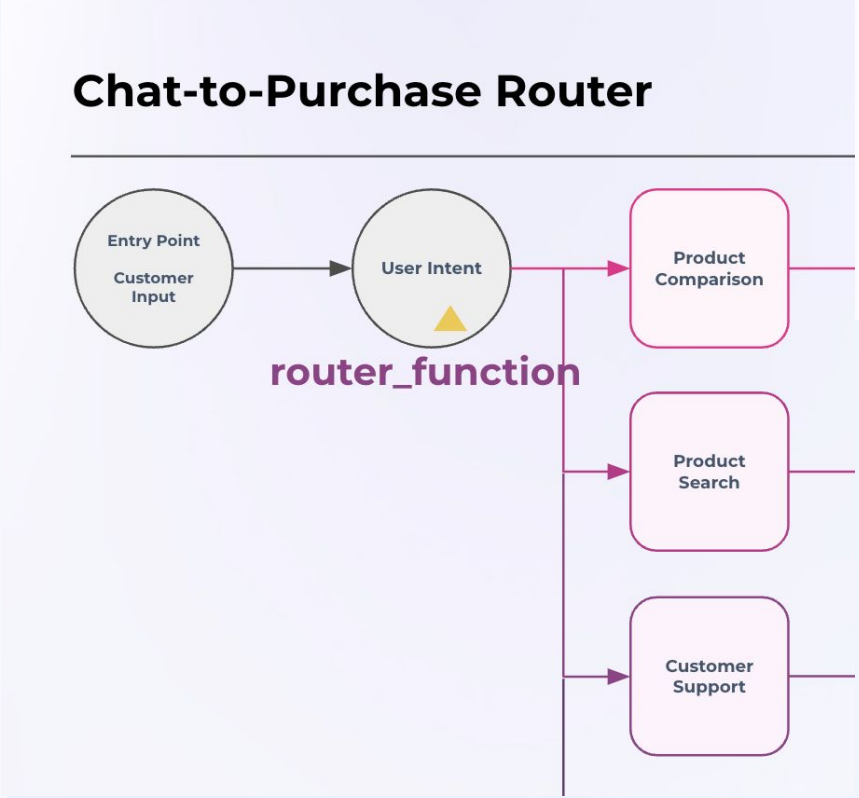
```python
class AgentFlow(Workflow):
    @step(pass_context=True)
    async def router(self, ev: StartEvent, ctx: Context) -> ToolCallEvent | StopEvent:
        messages = ev.input

        response = await self.llm.achat_with_tools(
            model="gpt-4o",
            messages=messages,
            tools=[generate_and_run_sql_query, data_analyzer],
        )

        self.ctx.put(response.message)

        tool_calls = self.llm.get_tool_calls_from_response(response, error_on_no_tool_call=False)
        if tool_calls:
            return ToolCallEvent(tool_calls=tool_calls)
        else:
            return StopEvent(result=response.message.content)

    @step(pass_context=True)
    async def tool_call_handler(self, ev: ToolCallEvent, ctx: Context) -> RouterInputEvent:
        tool_calls = ev.tool_calls

        for tool_call in tool_calls:
            function_name = tool_call.tool_name
            arguments = tool_call.tool_kwargs

            function_callable = get_function_callable_by_name(function_name)

            function_result = function_callable(arguments)
            message = ChatMessage(
                role="tool",
                content=function_result,
                additional_kwargs={"tool_call_id": tool_call.tool_id},
            )

            self.ctx.put(message)

        return RouterInputEvent(input=self.ctx.get())
```

Router V0 of an Agent

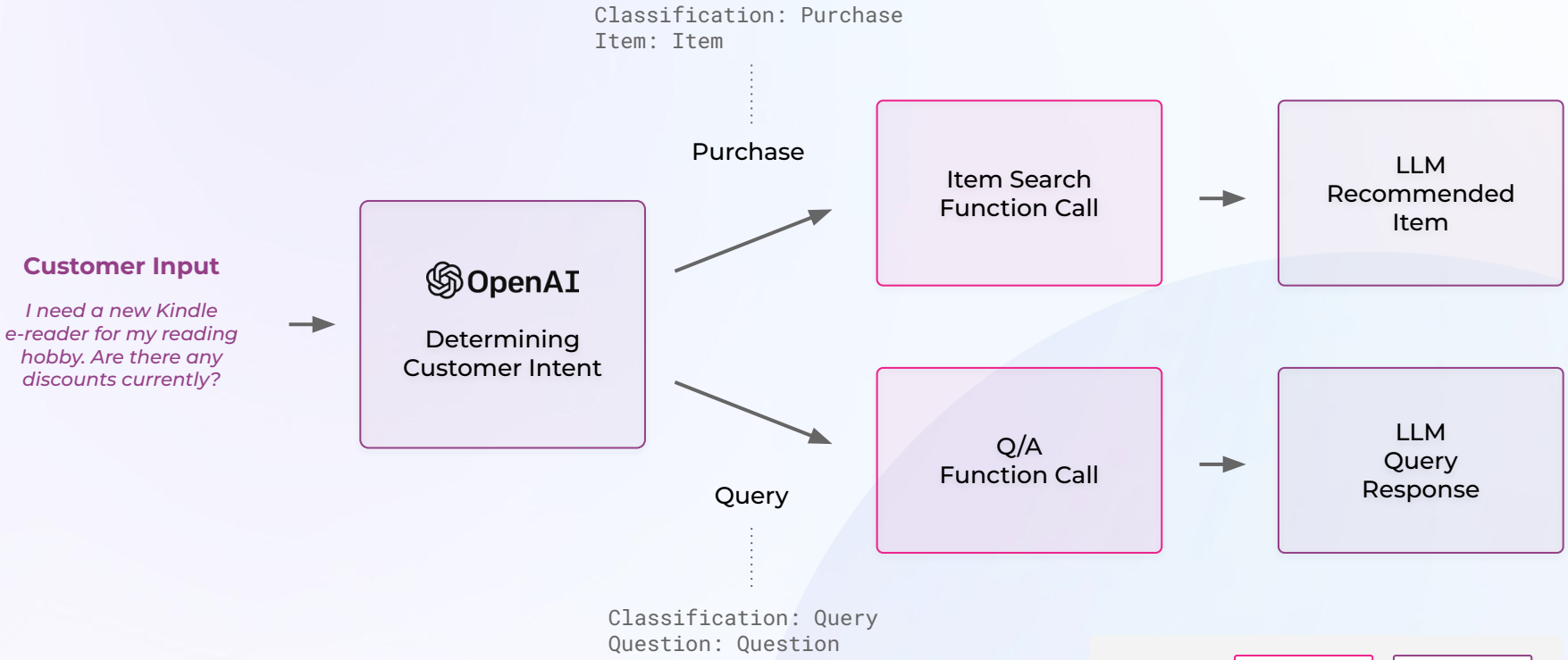
LLM Router Purpose:

An LLM router is main driver for what path or action should be taken within an Agent. There might be more than one LLM router within an agent.

The LLM router is normally handled using function calling in LLMs.



Simple Router Architecture: Chat to Purchase App



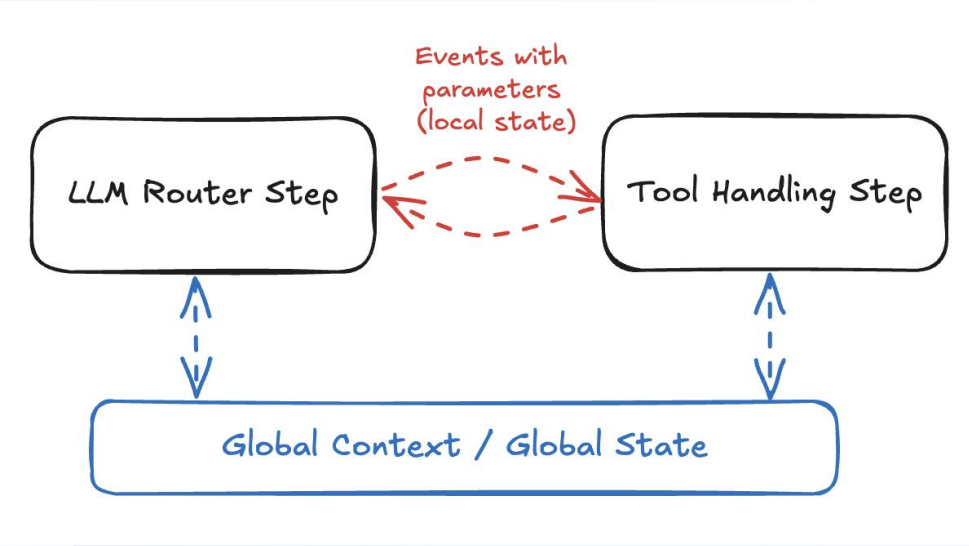
Global and Local State Within Agent

Examples of Where Saving State

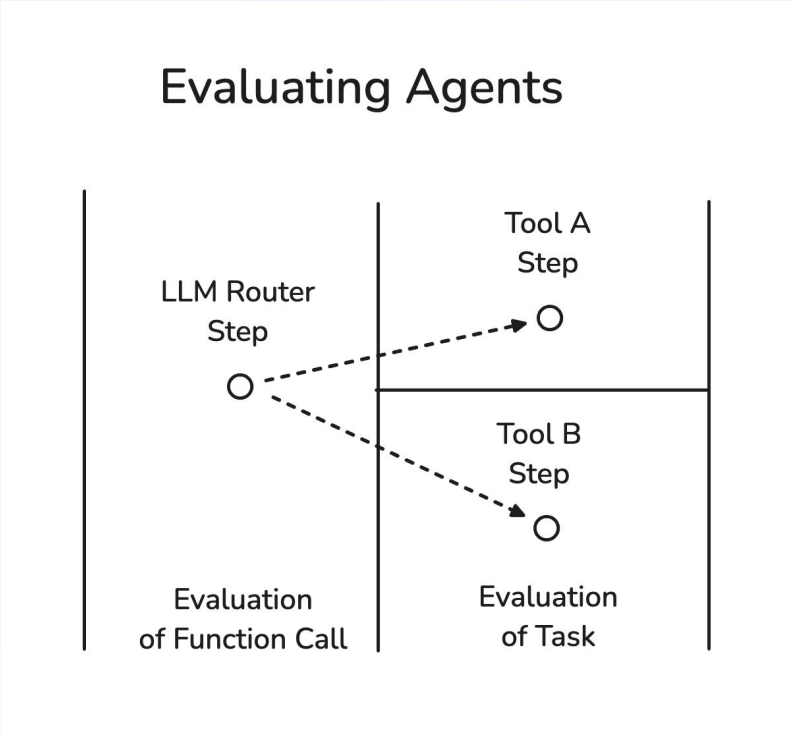
- Saving outputs of a task
  - Debugging results
  - Search results
- Saving large amounts of data you don't want in context window
- Saving interests from a user

Use of State

- Use state in one skill based on results of another
- Running and Iterating On Code
- Deeper analysis on a returned search result



Evaluation of Agent Workflows



- Break up Your Evaluation into Useful Components
- Evaluation of Function Call Routing and Parameter Extraction
- Evaluate Results of Actions or Tasks

Thank you.

Sign up for a free account at:

arize.com/join

Check out Phoenix, our OSS tool, at:

phoenix.arize.com