大型语言模型的局限性与链式调用的崛起

一个独立的语言模型，在**访问工具、外部上下文和多步骤工作流程**等方面存在固有的局限性。这些限制促使我们探索更复杂的系统设计。

探索LLM应用的控制流

工具访问

LLM需要外部工具来执行特定任务，例如数据查询或代码执行。

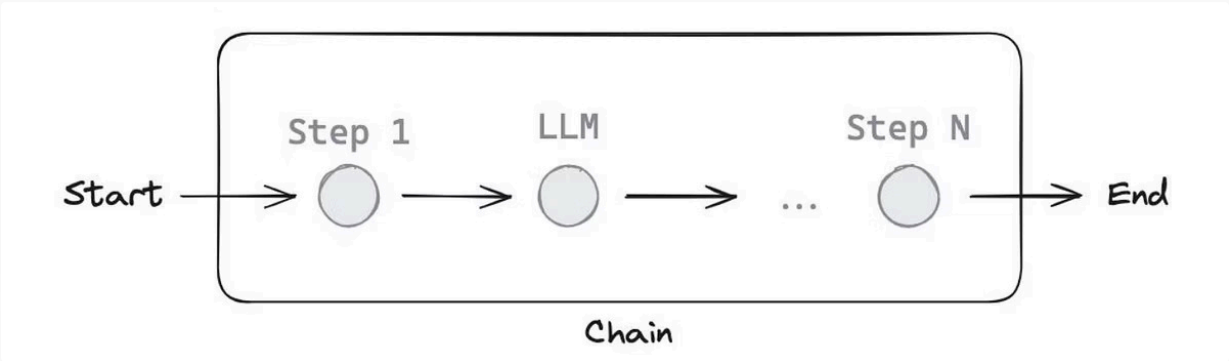
外部上下文

为了生成更准确和相关的响应，LLM需要访问超出其训练数据的最新信息。

多步骤工作流程

复杂的任务通常需要将多个LLM调用与其他操作（如数据预处理或后处理）结合起来。

为了克服这些局限性，**许多LLM应用程序采用了一种控制流模式**，在LLM调用前后执行预处理、工具调用和检索等步骤。



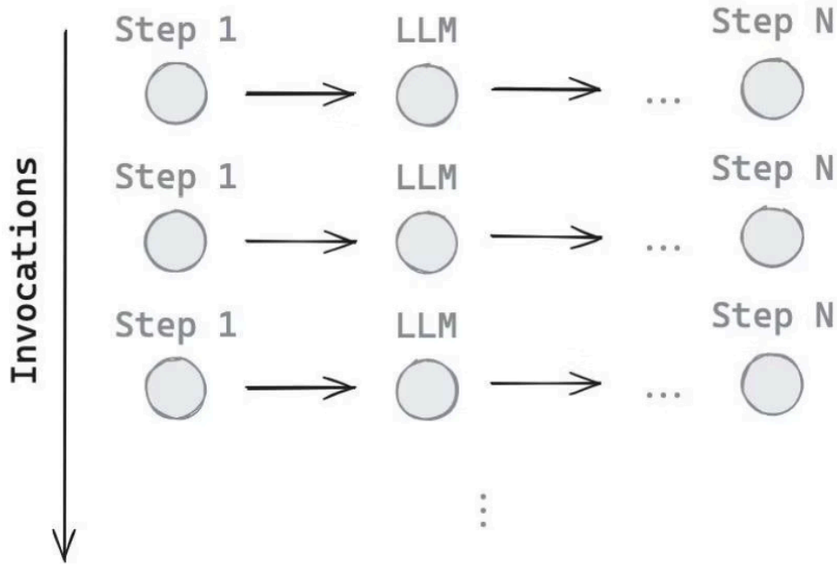
链式调用的核心概念

这种控制流形成了一个“链”，其中每个步骤按预定义顺序执行。**链式调用的核心优势在于其可靠性：**每次都能遵循相同的控制流。

> 通过结构化链式调用，我们确保每次LLM交互都伴随着必要的支持操作，从而提高输出的质量和一致性。

从固定流到LLM驱动的智能代理

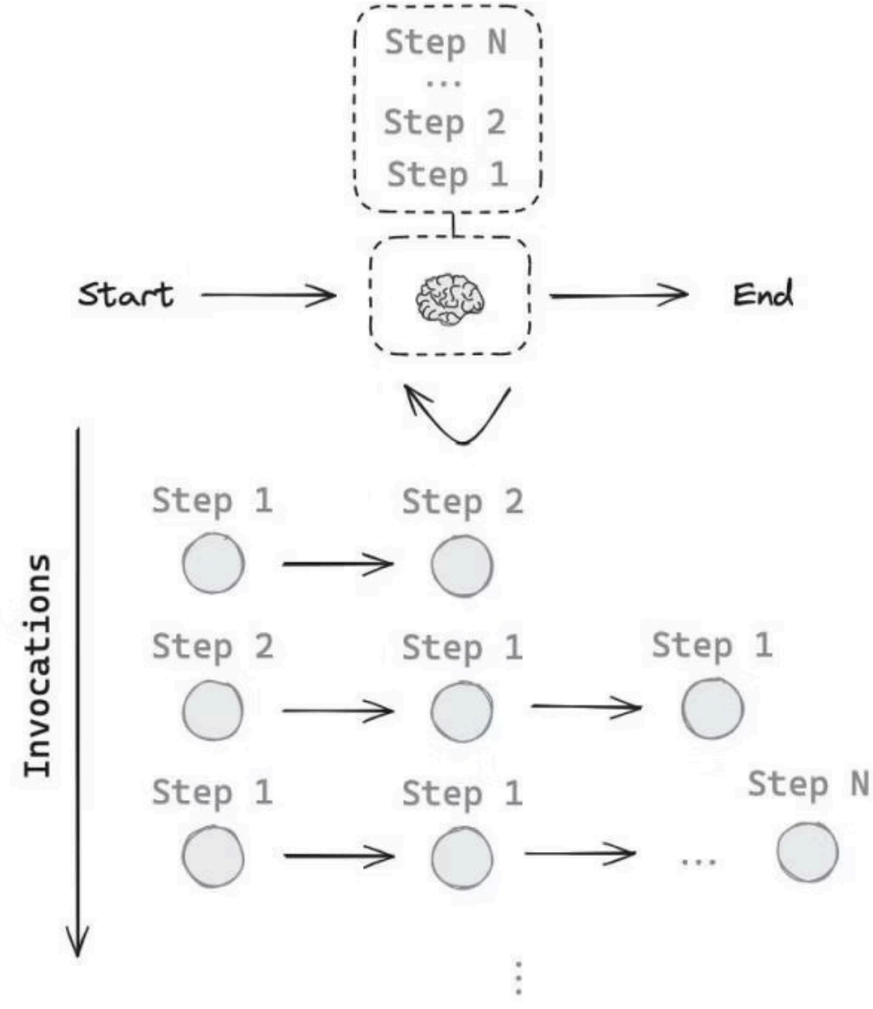
传统的固定控制流



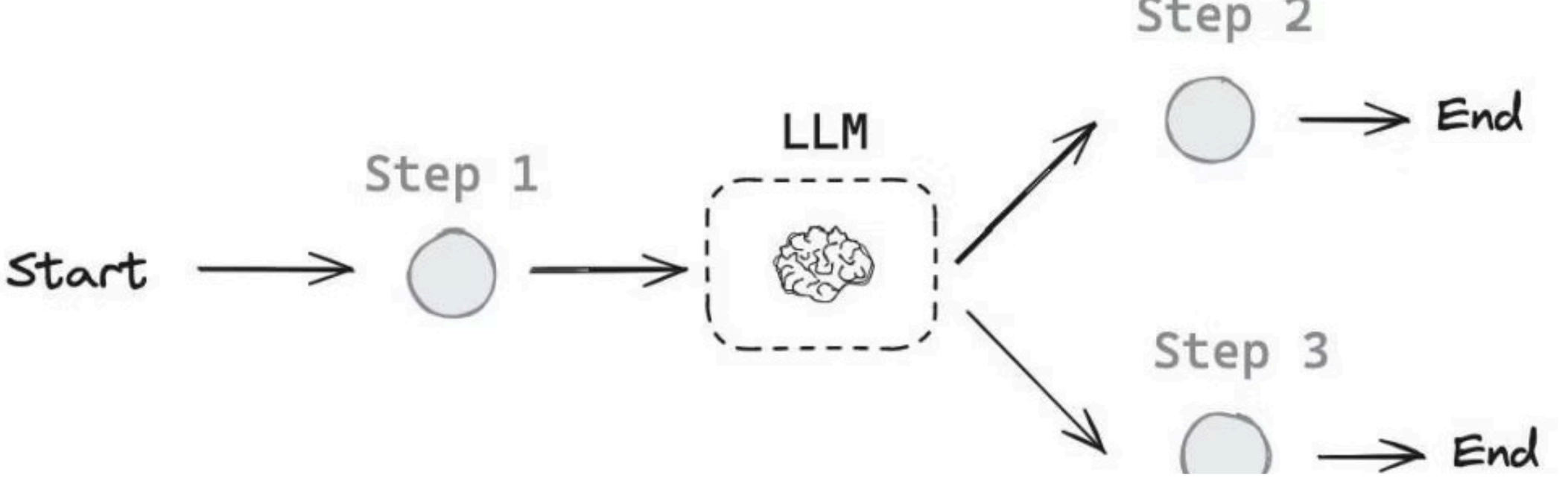
在固定控制流中，每一步都是预先确定的。

- **虽然可靠，但缺乏适应性和灵活性**
- 难以应对复杂多变的场景

我们期望的LLM驱动代理



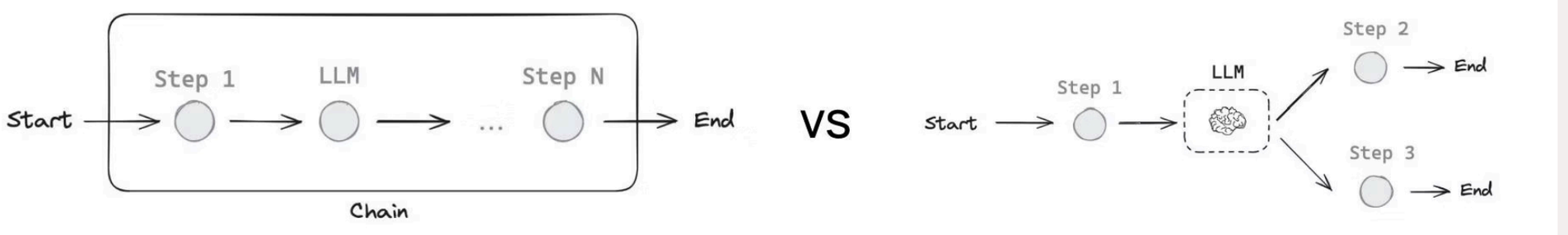
我们真正想要的是**能够自主选择控制流的LLM系统**，这使得系统更智能、更具响应性，能够根据情况动态调整策略。



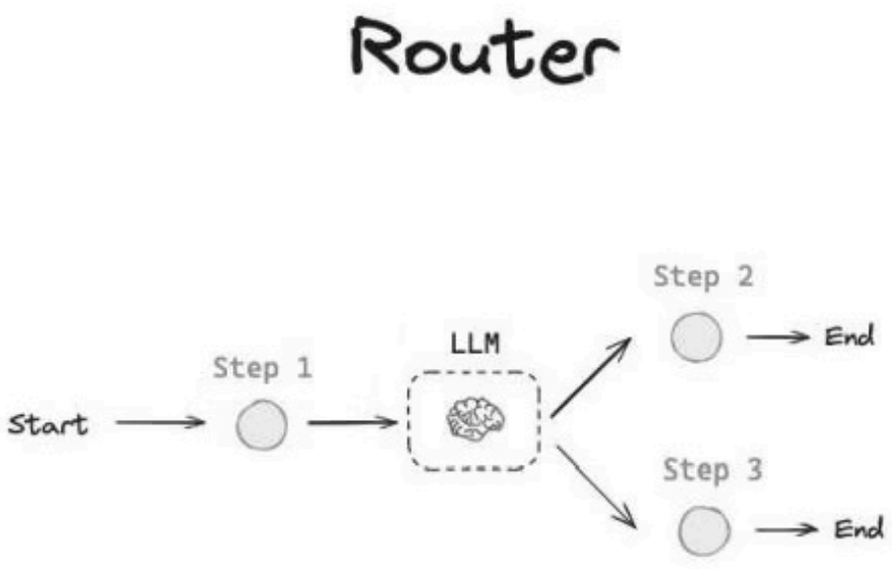
代理：LLM定义的控制流

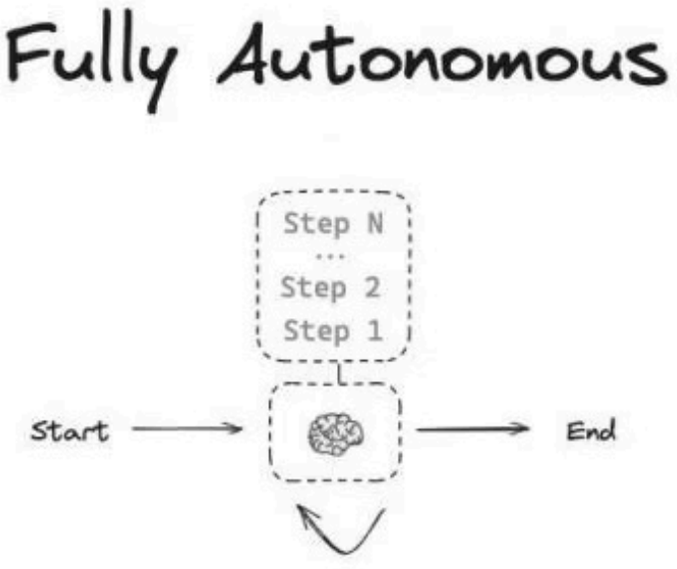
代理可以被理解为一种由LLM定义的控制流。它允许模型根据当前情境和目标，动态地决定下一步的行动，例如调用特定工具或检索信息。

固定 vs. LLM定义的控制流



固定流保证一致性，但缺乏适应性；**LLM定义的流则提供了灵活性和自主性，以应对新情况。**





代理的多样性与实际挑战

市场上存在多种类型的代理，它们被设计用于解决不同的问题。然而，在实际应用中，构建这些代理仍然面临诸多挑战。

- **Router（路由型）**：控制程度较强，LLM 仅在某些分支上做决策。→ 更受控
- **Fully Autonomous（完全自治型）**：LLM 几乎完全掌控控制流，自动选择步骤和工具。→ 更灵活

鲁棒性

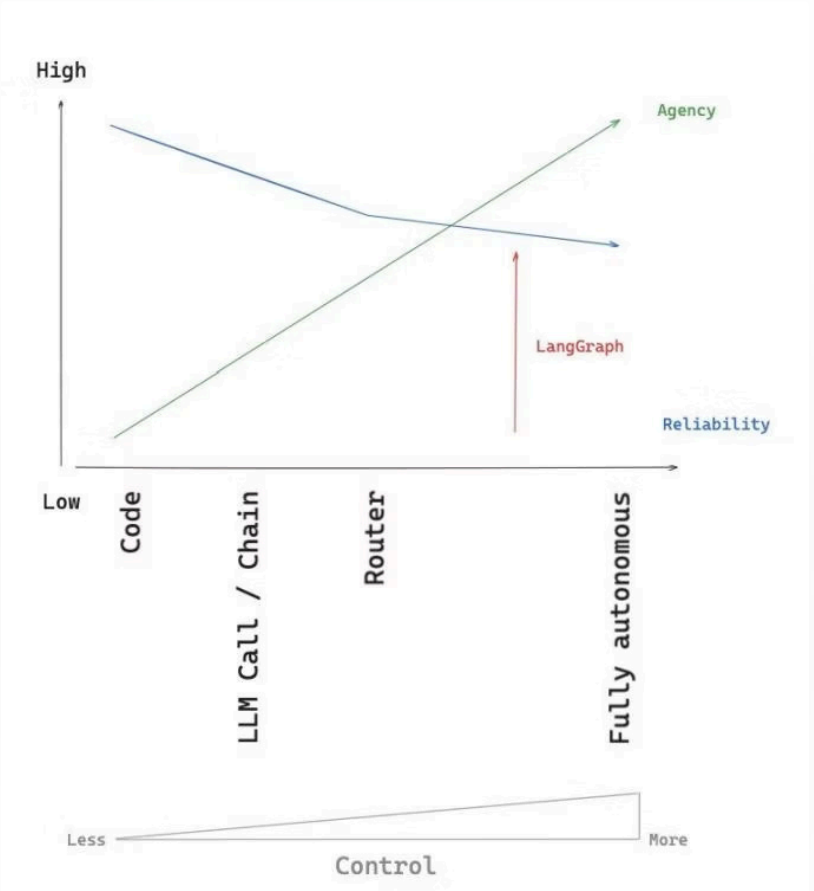
确保代理在各种输入和环境下都能**稳定可靠地运行**。

可控性

在赋予LLM自主决策能力的同时，保持对系统行为的有效监管。

效率

优化代理的执行速度和资源消耗，尤其是在复杂工作流程中。



在实践中，Agent 面临一个权衡问题：

- 灵活性（Agency/Flexibility）高时，可靠性（Reliability）往往低。
- 控制（Control）越少，系统不可预测性越强。

图表展示：随着从“Code → LLM Call/Chain → Router → Fully Autonomous”逐渐增加灵活性，可靠性逐渐下降。

LangGraph：平衡可靠性与控制

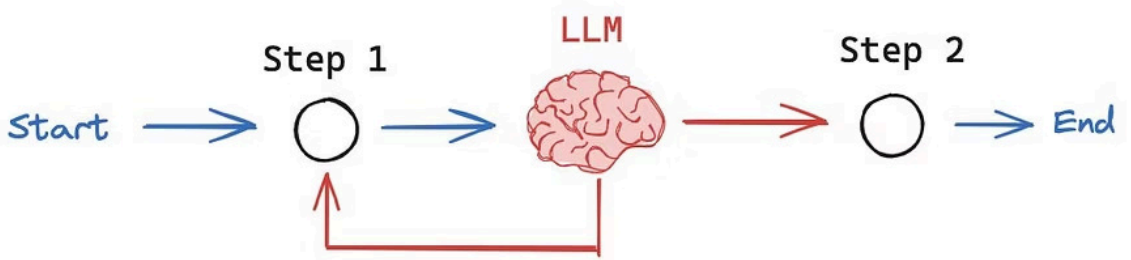
LangGraph框架旨在通过引入基于图的控制流，帮助开发者在LLM系统中平衡可靠性与控制，从而“弯曲可靠性曲线”。

> 通过LangGraph，开发者可以灵活地定义部分控制流，同时注入LLM的智能，实现既可靠又具备自主决策能力的代理。

LangGraph 可靠性与控制

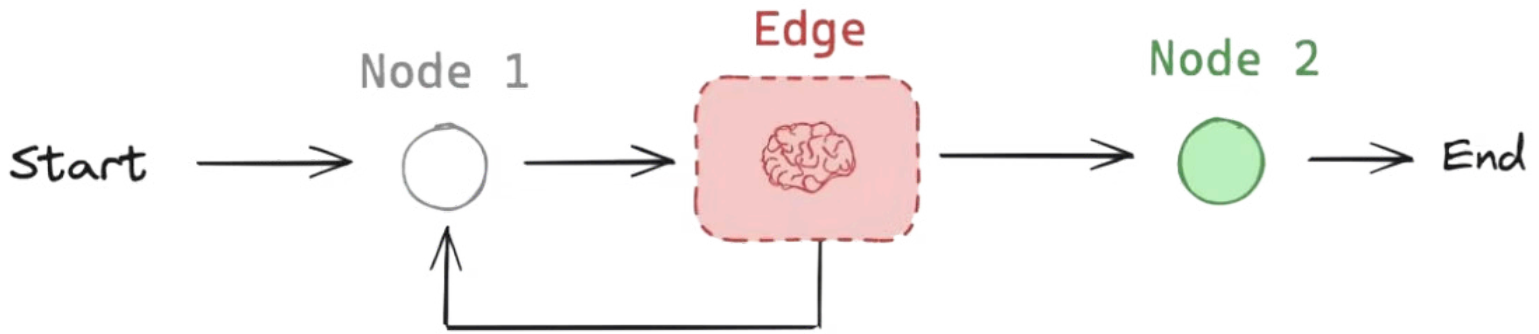
LangGraph 的直觉是：

- **由开发者定义部分控制流**
  （保证可靠性）
- **由 LLM 决定其他部分**（保证灵活性）



上图：开发者固定 Step 1 和 Step 2，流程可靠。

下图：在 Step 1 与 Step 2 之间，插入 LLM 作为 Agent 进行决策，使流程具备一定的控制能力。



LangGraph: 将自定义的控制流表示为图形

LangGraph的核心在于能够**将自定义的控制流表示为图形**。这意味着开发者可以直观地设计和管理LLM代理的复杂行为。

LangGraph 将这种思路用**图（Graph）**来表示：

- **节点（Node）：**表示一个步骤或操作→节点可以调用工具、修改状态
- **边（Edge）：**连接节点，定义控制流→根据 LLM 决策来路由流程
- LLM 可以作为图中的决策点；内存（Memory）在整个图中共享

直观的流程设计

以图形方式定义LLM调用的序列、并行执行和条件分支。

模块化构建

将复杂的代理分解为可重用、易于管理的独立节点。

强大的调试能力

图形化界面有助于跟踪和理解代理在运行时的决策路径。

LangGraph的模块化学习路径

01

基础知识

了解LangGraph的基础、链、路由器和通用自主代理。

02

记忆管理

构建能够记住过去交互的代理，增强上下文理解。

03

人机协作

设计有人类监督的代理，实现智能与人工干预的平衡。

04

定制化开发

根据具体需求定制和优化代理，打造实用解决方案。

05

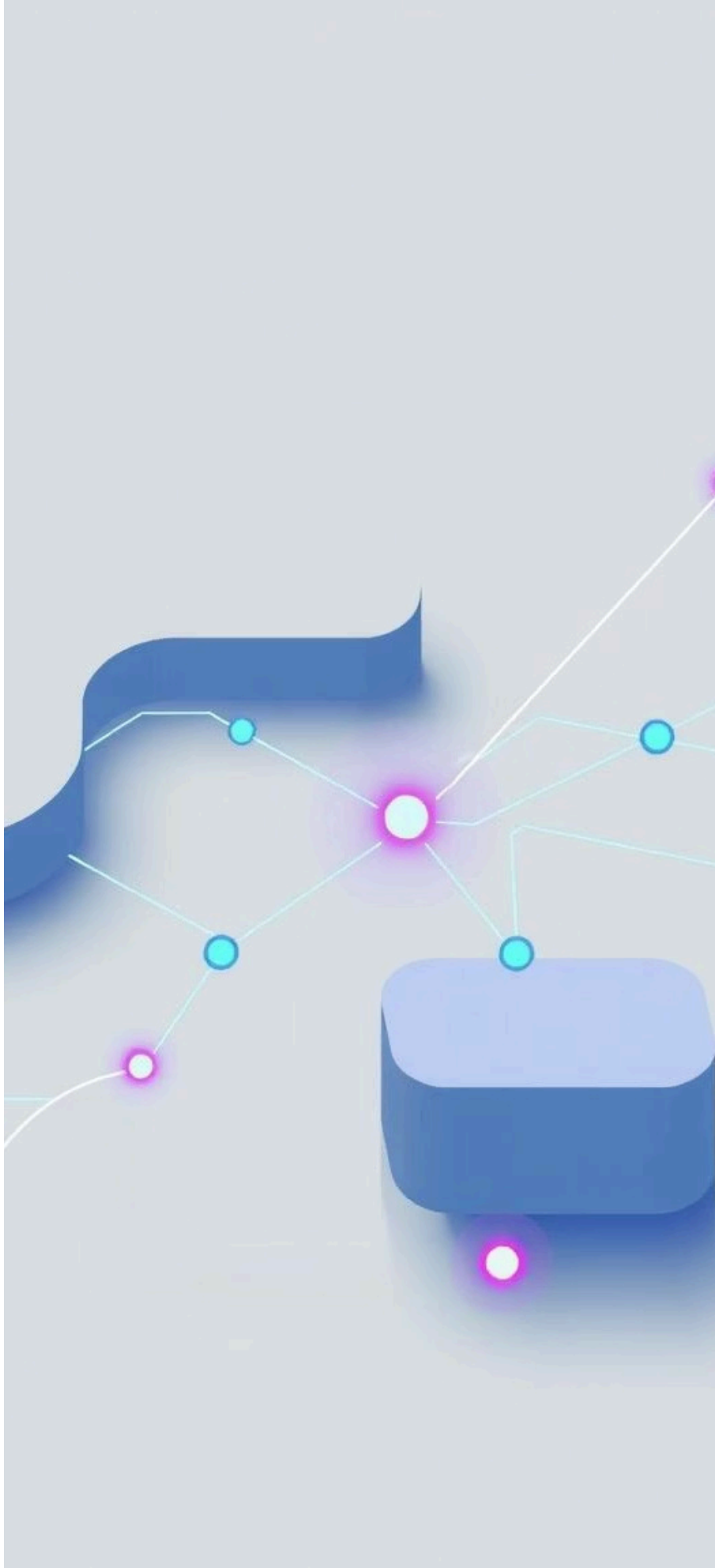
长期记忆

让智能体能够长期记住用户与上下文信息

06

部署

构建 Docker 镜像，部署智能体



LangGraph的未来展望

LangGraph通过其图结构为构建更可靠、更灵活的LLM代理提供了强大的框架。这不仅简化了复杂LLM应用的设计，也为未来的AI系统开发开辟了新的可能性。

随着LLM技术不断演进，LangGraph将继续赋能开发者，共同探索和实现更智能、更自主的AI解决方案。