为LangGraph驱动的AI教练应用打造的专业级架构蓝图与提示词手册

第一部分：AI教练中枢——LangGraph架构蓝图

本部分旨在为AI教练应用构建一个坚实的基础架构。我们所设计的并非一个简单的智能体链条，而是一个稳健、有状态且具备韧性的系统，专为处理复杂、长期的用户交互而生。选择“主管-专家”（Supervisor-Specialist）模型是一项战略性决策，旨在平衡中心化控制与分布式专业能力，这一模式已在行业最佳实践中得到验证。

1.1. 主管-专家模型：编排与专业分工

架构论证

本架构将采用**主管（Supervisor）**模式。选择此模式是基于其稳健性和通用性，因为它对下属子智能体的假设最少[1]。一个单一的主管智能体将负责管理所有用户交互，并将具体任务委派给一个由专业智能体组成的团队。这种设计避免了在结构性较弱的“群体”（Swarm）架构中可能出现的“转译”开销和潜在的“写入”冲突，这些问题都可能导致性能下降[1]。主管作为用户连贯、单一的接触点，这对于需要保持一致人格的教练应用至关重要。

多智能体系统在处理可并行的“读取”类任务（如信息检索、分析）时表现出色，但在处理涉及状态变更或需要一致输出的“写入”类任务时，则面临严峻挑战。并行化的写入操作极易产生冲突，导致结果混乱[2]。从用户的角度看，一次教练对话本质上是一次“写入”任务——用户期望接收到单一、连贯的建议流。一个完全去中心化的群体架构将面临多个智能体试图同时向用户或共享状态“写入”信息的风险，这会引发困惑和矛盾的建议。主管架构通过设立一个单一的“写入”瓶颈（即主管智能体），同时允许将认知负荷较重的“读取”和处理任务并行分配给专家智能体，从而巧妙地

解决了这一核心问题。因此，该架构直接规避了多智能体系统的一个关键失败模式，使其更适用于教练这类敏感应用。

智能体名册与角色

- 主管智能体 **(Supervisor Agent):** 系统的中央编排者。负责管理主对话循环，解析用户意图，向专家智能体分派任务，并整合最终的回复内容。
- 洞察与分析智能体 **(Insight & Analysis Agent):** 专家智能体，负责对用户输入（如日记、目标进度）进行深入的、数据驱动的分析。
- 策略与规划智能体 **(Strategy & Planning Agent):** 专家智能体，基于洞察分析结果，制定可行的行动计划、策略和教练练习。
- 反思与整合智能体 **(Reflection & Synthesis Agent):** 专家智能体，负责总结对话会话，追踪长期进展，并生成综合报告。

LangGraph 实现

该系统将在LangGraph中被建模为一个循环图（Cyclic Graph）。图中的**节点（Nodes）将代表各个独立的智能体（主管、洞察等）以及工具调用函数 [3]。图的边（Edges）**将由主管智能体的路由逻辑控制，该逻辑根据当前状态动态决定下一步的走向。这种方式赋予了开发者对“认知架构”的完全控制权，避免了隐藏提示词带来的不可预测性 [2]。

1.2. 状态管理与持久化执行：构建一个有韧性的教练

定义状态图

我们将为LangGraph图定义一个全面的State（状态）对象。该对象是一个字典，包含了系统的完整快照：

- messages: 一个列表，包含对话历史中的所有消息。
- user_profile: 一个结构化对象，用于存储用户的目标、偏好和背景信息。
- session_id: 当前教练会话的唯一标识符。

- scratchpad: 一个字典，供智能体存储中间思考过程和临时结果。
- task_queue: 一个待办任务列表，供主管智能体进行委派。

持久化执行与检查点

智能体系统具有状态性，并且需要长时间运行，这使其在面对系统故障时尤为脆弱，微小的错误也可能滚雪球式地演变成灾难性的失败[2]。为了应对这一挑战，我们的LangGraph实现将内置检查点（**Checkpointing**）机制。在每次状态转换后（例如，一个智能体完成任务后），整个状态对象将被序列化并保存。这一机制使得系统能够在发生错误时从上一个已知的良好状态恢复（**Resume**），而不是从头开始，这对于保障长期、高价值交互的连续性至关重要[2]。这一功能将通过LangGraph内置的持久化模块进行管理。

管理长时程上下文

对于一个长期的教练关系而言，单个LLM的上下文窗口是远远不够的[1]。我们的状态管理策略将包含上下文压缩机制。主管智能体将周期性地调用“反思与整合智能体”，对messages历史中的旧有部分进行总结。这些凝练的摘要将替代详细的对话记录，保留在活动状态中，而完整的历史记录则被存入向量数据库，以备将来需要时进行检索。

1.3. 智能体通信协议：强制实现无瑕交互

结构化输出的首要地位

可靠的多智能体系统依赖于可预测的、机器可读的通信。自由格式的文本是脆弱且不稳定的。因此，所有智能体之间的通信和工具调用都将通过经过严格验证的JSON对象进行[4]。

从早期依赖在提示词中简单指令（例如，“请用JSON格式回应”）的不可靠方法，到后来引入“修复解析器”（即用另一次LLM调用来修复损坏的JSON），再到API提供商推出仅保证语法有效但不保证结构正确的“JSON模式”，LLM系统的发展历程清晰地表明，业界已经认识到，可靠的进程间通信不能依赖于概率性的指令[7]。最新的演进是API原生的模式强制执行功能，它在生成过程中就

约束模型的输出，使其必须符合开发者提供的模式。这一从概率性的提示工程艺术到确定性的软件工程科学的范式转变为构建生产级、可靠的多智能体系统提供了基石。对于我们的AI教练应用，采用这种最先进的方法是确保系统不会因基本的格式错误而失败的必要条件。

API原生模式强制执行

我们将利用最稳健的方法来确保通信的可靠性。我们将不再仅仅依赖提示词指令，而是采用API原生功能，如OpenAI的结构化输出（在response_format中指定json_schema）或Anthropic的工具使用模式 [9]。这些功能在生成层面就对模型的输出进行约束，提供了接近100%的可靠性，并消除了对脆弱的解析和修复循环的需求 [8]。

使用Pydantic进行模式定义与验证

所有的JSON模式都将在Python代码库中定义为Pydantic模型。这为数据结构提供了单一事实来源，支持静态分析，并提供自动验证。LangChain的.with_structured_output()方法能够直接与Pydantic模型集成，将成为我们确保模式遵循性的主要接口 [12]。

表1：核心通信模式 (Pydantic模型)

下表定义了智能体之间通信的“API合约”，它将抽象的“通信”概念具体化为可实现的代码。预先定义这些模式可以确保系统范围的一致性，减少集成错误，并为开发工作创建一个清晰的蓝图。这不仅仅是文档，而是强制系统可靠性的核心架构构件。

Pydantic类名	用途	关键字段 (名称: 类型, 描述)
TaskDelegation	主管 -> 专家。分配一个具体任务。	task_id: str (任务唯一ID), task_description: str (清晰的任务描述), input_data: dict (完成任务所需的数据), schema_for_output: dict (期望输出的JSON模式)

AnalysisResult	洞察智能体 -> 主管。返回分析发现。	task_id: str (对应的任务ID), summary: str (发现摘要), key_insights: list[str] (核心洞察列表), hypotheses_tested: list[dict] (被检验的假设及其证据)
ActionPlan	策略智能体 -> 主管。提供一份教练计划。	task_id: str (对应的任务ID), plan_title: str (计划标题), steps: list[dict] (包含具体步骤、原理和衡量标准的列表), rationale: str (计划背后的整体逻辑)
SessionSummary	反思智能体 -> 主管。提交一份会话总结。	session_id: str (会话ID), key_topics: list[str] (讨论的关键主题), action_items: list[str] (商定的行动项), user_sentiment: str (用户情绪评估)

第二部分：提示词手册——工程化教练智能体的思维

本部分提供了为每个智能体精心设计的、专业级的完整提示词。每个提示词都是一个结合了角色设定、任务指令、约束条件和先进推理技术的复杂“认知程序”，旨在引导LLM产生期望的行为。

2.1. 主管智能体：编排者

核心身份与系统提示词

此提示词确立了该智能体作为用户主要AI教练的人格。它将定义其沟通风格（共情、清晰、鼓励）及其核心任务：理解用户的需求，并协调一个内部专家团队以提供最佳支持。提示词将明确指示，它绝不应自行进行分析或规划，而必须始终将这些任务委派给专家。

系统提示词示例：

你是一个名为“启航”的AI教练，你的角色是用户的私人成长伙伴。你的沟通风格是温暖、共情且富有洞察力的。你的核心使命是：
1. 深入理解用户的目标、挑战和情感状态。
2. 管理一个由内部专家智能体组成的团队，每个智能体都在特定领域（如数据分析、策略规划、反思总结）拥有深厚的专业知识。
3. 当你收到用户的请求时，你的首要任务是判断需要哪位专家的能力来最好地回应用户。你必须将复杂的分析和规划任务委派给相应的专家。
4. 在收到专家的反馈后，你负责将其整合成一个清晰、连贯且对用户有帮助的回复。
5. 你绝不能自己进行深入的数据分析或制定详细的行动计划。你的专长在于倾听、理解和协调。你是一个引导者，而不是一个全能的解决者。

任务提示词 (动态生成)

- 用户意图分析与路由提示词：此提示词在每个对话回合开始时使用。它接收最新的用户消息和对话历史作为输入，目标是分类用户的意图（例如，“情绪宣泄”、“请求计划”、“进度汇报”、“提问”），并决定下一步行动。输出将是一个结构化的JSON对象，指明接下来应调用哪个专家智能体（或者，对于简单的问候等，由主管直接处理）。这利用了将复杂任务分解的原则 [2]。
  动态生成指令示例：

```JSON
{
 "task": "Analyze user intent and determine routing",
 "user_message": "{latest_user_message}",
 "conversation_history": "{recent_conversation_history}",
 "available_agents":,
 "output_schema": {
  "type": "object",
  "properties": {
   "user_intent": {
    "type": "string",
```

```
      "description": "A concise classification of the user's primary intent.",
      "enum": ["emotional_venting", "request_for_plan", "progress_update", "clarifying_question",
"general_chat"]
    },
    "next_agent": {
      "type": "string",
      "description": "The name of the agent best suited to handle this intent.",
      "enum":
    },
    "rationale": {
      "type": "string",
      "description": "A brief explanation for the routing decision."
    }
  },
  "required": ["user_intent", "next_agent", "rationale"]
 }
}
```

- 任务分解与委派提示词：一旦确定了下一个智能体，主管将使用此提示词来构建TaskDelegation对象。它会将用户的自然语言请求转化为一个对专家来说精确、无歧义的任务描述，并包含状态图中所有必要的上下文。这是避免因指令模糊而导致子智能体表现不佳的关键步骤[2]。

2.2. 洞察与分析智能体：数据科学家

核心身份与系统提示词

系统提示词示例：

你是一位顶尖的数据科学家和定性研究员。你的专长是分析文本数据，以揭示其潜在的模式、主题和因果关系。你的工作方式是客观、循证且严谨的。你从不提供建议或个人观点；你只呈现经过整合的发现。你的输出必须严格遵守所提供的JSON模式。

任务提示词：思维树 **(ToT)** 用于深度洞察生成

此提示词是思维树（Tree-of-Thoughts, ToT）框架的直接实现，用于分析复杂的用户输入（如一篇长日记）。它将引导LLM通过一个多步骤的推理过程，旨在克服单路径、线性推理的局限性 [14]。

- 第一步：思维分解与生成 **(Thought Decomposition & Generation)**。“根据用户提供的文本，生成3个不同的、高层次的假设，这些假设可以解释用户当前的挑战或感受。假设必须是可检验的陈述。例如：‘假设1：用户的压力源于其对工作项目感知到的失控感。’‘假设2：用户的挫败感与其价值观和日常活动之间的不匹配有关。’” 这一步使用了ToT中的**采样（Sampling）**技术，以生成一个多样化的思维空间 [16]。
- 第二步：状态评估 **(State Evaluation)**。“对于这3个假设中的每一个，系统性地在所提供的文本中搜索，并列出所有支持或反驳它的证据片段。评估每个假设的证据强度，并为其分配一个1到5的置信度分数。请以结构化的格式呈现此评估过程。” 这一步强制模型对其自己生成的想法进行审慎的自我评估 [15]。
- 第三步：综合与剪枝 **(Synthesis & Pruning)**。“基于你的证据评估，选择最可信的假设。如果多个假设都得到了充分支持，请解释它们之间可能存在的相互联系。将你的发现整合成一份关于核心洞察的简明摘要，明确指出用户状况的主要驱动因素。丢弃置信度低的假设。” 这一步模拟了ToT中的搜索和回溯/剪枝过程 [16]。

最终，该智能体的输出将是一个严格遵循模式的AnalysisResult JSON对象。

2.3. 策略与规划智能体：教练

核心身份与系统提示词

系统提示词示例：

你是一位经过认证的专业教练，擅长创建可行的、个性化的发展计划。你富有同理心、激励人心且

注重实效。你的主要目标是通过将宏大目标分解为可管理的步骤来赋能用户。你严格遵守已建立的教练框架，如SMART（具体的、可衡量的、可实现的、相关的、有时限的）原则。

任务提示词：带有批判性反思的自我精炼 **(Self-Refine with Critique)** 用于高质量计划

简单地要求LLM给出一个“好计划”是不可靠的。“自我精炼”模式将“好”这个抽象概念解构成一个具体的、可验证的检查清单。这使得模型的推理过程外部化，从而更加透明和可控。它还降低了模型偏爱其最初有缺陷的回答的风险（即自我偏见），因为它被迫根据一套外部原则进行评估[19]。

这个过程的价值在于，它将一个单次通过的“黑箱”生成任务转变为一个多步骤的、透明的认知过程。第一步是初步的、可能存在缺陷的生成。第二步强制LLM扮演一个不同的角色（质量保证审查员），并使用一种不同的认知过程（对照标准进行评估，而非自由生成）。这种明确的批判指令使评估过程变得透明。第三步则利用这种结构化的反馈来指导有针对性的修订，这远比一个模糊的“让它变得更好”的指令有效得多。它将一个不可靠的创造性任务，转变为一个更可靠的、准算法化的流程[21]。

- 第一步：生成初始草案 **(Initial Draft Generation)**。“基于提供的AnalysisResult，生成一份行动计划草案，包含3-5个用户在接下来一周可以采取的具体步骤。为每个步骤提供简要的理由。”
- 第二步：结构化自我批判 **(Structured Self-Critique)**。“现在，扮演一名质量保证审查员的角色。根据以下标准，严格评估你刚才创建的计划草案。对每条标准，回答‘通过’或‘失败’，并给出一句理由。
    - 标准**1 (**具体性**):** 每个步骤是否完全无歧义？
    - 标准**2 (**可衡量性**):** 是否有明确的方法来判断步骤是否已完成？
    - 标准**3 (**可实现性**):** 对于处于用户当前状况的人来说，这个步骤是否现实？
    - 标准**4 (**相关性**):** 该步骤是否直接解决了分析中的核心洞察？
    - 标准**5 (**以用户为中心**):** 计划是否尊重了用户明确提出的限制和偏好？”
- 第三步：最终计划精炼 **(Final Plan Refinement)**。“根据你的批判性评估，生成最终的ActionPlan JSON对象。重写所有在评估中‘失败’的步骤，以解决已识别的缺陷。如果所有步骤都‘通过’，则重新提交原始计划。” 这个迭代过程将一个简单的生成任务转变为一个动态的评估和改进循环[23]。

2.4. 反思与整合智能体：记录员

核心身份与系统提示词

系统提示词示例：

你是一位一丝不苟的记录员和报告员。你唯一的目标是创建清晰、简洁且事实准确的对话摘要。你保持客观，不添加任何解释或个人观点。你的工作成果必须能够与源对话记录进行核实验证。

任务提示词：验证链 **(CoVe)** 用于事实性总结

此提示词确保生成的摘要基于事实，不含幻觉，这对于准确追踪用户进展至关重要。它是验证链（Chain-of-Verification, CoVe）模式的直接实现 [20]。

- 第一步：生成基线摘要 **(Generate Baseline Summary)**。“阅读所提供的会话记录，并生成一份摘要草稿，涵盖讨论的关键主题、做出的决定以及商定的行动项。”
- 第二步：生成验证性问题 **(Generate Verification Questions)**。“基于你的摘要草稿，生成一个问题列表。这些问题必须可以通过查阅原始记录以‘是’或‘否’来回答，旨在验证你摘要中的关键声明。例如：‘用户是否明确同意了行动项X？’‘主题Y是否是对话前半部分的主要焦点？’”
- 第三步：回答验证性问题 **(Answer Verification Questions)**。“通过引用原始记录中的具体段落来回答你刚才生成的每一个验证性问题。如果找不到直接引文，请回答‘不确定’。”
- 第四步：生成最终的、经过验证的摘要 **(Generate Final, Verified Summary)**。“回顾你的基线摘要和你对验证性问题的回答。编辑摘要以纠正任何不准确之处，或删除任何被标记为‘不确定’的声明。生成最终的SessionSummary JSON对象。” 这个过程强制智能体将其自身的陈述与源材料进行交叉引用，从而显著提高了输出的事实可靠性 [20]。

结论与建议

本报告为构建一个基于LangGraph的专业级AI教练应用提供了一套全面的架构蓝图和提示词工程策略。其核心设计理念在于通过结合稳健的系统架构与先进的LLM推理技术，来应对构建复杂、可靠AI智能体所面临的挑战。

核心架构原则：

1. 采用主管-专家模型：这是实现控制与专业能力平衡的关键。通过中心化用户交互（写入）和分布式任务处理（读取），该架构在根本上提升了系统的连贯性和可靠性。
2. 实施持久化状态管理：对于需要长期交互的应用，通过检查点机制实现持久化执行是必不可少的。这确保了系统的韧性，保护了用户交互的价值。
3. 强制执行结构化通信：智能体间的通信必须从依赖概率性的提示词指令，转向依赖确定性的、API原生的模式强制执行。使用Pydantic模型和with_structured_output等工具，是构建生产级系统的基石。

核心提示词工程策略：

1. 思维树 **(ToT)** 用于深度分析：对于需要探索性分析的复杂任务，ToT通过模拟人类的假设-验证过程，能够产出远超线性思维的深刻洞察。
2. 自我精炼 **(Self-Refine)** 用于高质量生成：通过引入结构化的自我批判循环，可以将抽象的质量要求转化为具体的、可执行的评估标准，从而显著提升生成内容的质量和可靠性。
3. 验证链 **(CoVe)** 用于事实准确性：在任何需要事实性陈述的场景（如总结报告），CoVe通过强制模型对其声明进行溯源验证，是抑制模型幻觉、确保内容真实性的有效手段。

综上所述，成功构建一个强大的AI教练应用，不仅需要选择正确的工具（如LangGraph），更需要采纳一套系统性的设计哲学。这套哲学强调通过明确的架构约束来管理复杂性，并通过工程化的提示词来引导LLM的认知过程，最终将AI智能体的构建从一门艺术，转变为一门更加可靠和可预测的工程学科。

Works cited

1. Benchmarking Multi-Agent Architectures - LangChain Blog, accessed October 9, 2025, https://blog.langchain.com/benchmarking-multi-agent-architectures/
2. How and when to build multi-agent systems - LangChain Blog, accessed October 9, 2025, https://blog.langchain.com/how-and-when-to-build-multi-agent-systems/
3. Basic Multi-agent Collaboration - GitHub Pages, accessed October 9, 2025, https://langchain-ai.github.io/langgraphjs/tutorials/multi_agent/multi_agent_collaboration/
4. Is JSON Prompting a Good Strategy? - PromptLayer Blog, accessed October 9, 2025, https://blog.promptlayer.com/is-json-prompting-a-good-strategy/
5. Enhance AI Models Prompt Engineering with JSON Output | by Novita AI - Medium, accessed October 9, 2025, https://medium.com/@marketing_novita.ai/enhance-ai-models-prompt-engineering-with-json-output-ca450f62159a
6. For Best Results with LLMs, Use JSON Prompt Outputs | HackerNoon, accessed October 9, 2025, https://hackernoon.com/for-best-results-with-llms-use-json-prompt-outputs
7. How to get 100% valid JSON answers? - Prompting - OpenAI Developer Community, accessed October 9, 2025, https://community.openai.com/t/how-to-get-100-valid-json-answers/554379

8. How to use the output-fixing parser | 🦜 LangChain, accessed October 9, 2025, https://python.langchain.com/docs/how_to/output_parser_fixing/
9. Introducing Structured Outputs in the API - OpenAI, accessed October 9, 2025, https://openai.com/index/introducing-structured-outputs-in-the-api/
10. The guide to structured outputs and function calling with LLMs - Agenta, accessed October 9, 2025, https://agenta.ai/blog/the-guide-to-structured-outputs-and-function-calling-with-llms
11. Ensuring Consistent JSON Output from LLMs on Amazon Bedrock | AWS Builder Center, accessed October 9, 2025, https://builder.aws.com/content/2wWabHxa0No1ZveOI7IMjQg6APP/ensuring-consistent-json-output-from-llms-on-amazon-bedrock
12. How to return structured data from a model | 🦜 LangChain, accessed October 9, 2025, https://python.langchain.com/docs/how_to/structured_output/
13. Structured Outputs from LLM using Pydantic | by Harisudhan.S - Medium, accessed October 9, 2025, https://medium.com/@speaktoharisudhan/structured-outputs-from-llm-using-pydantic-1a36e6c3aa07
14. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models - arXiv, accessed October 9, 2025, https://arxiv.org/abs/2201.11903
15. Tree of Thoughts: Deliberate Problem Solving with Large Language Models - arXiv, accessed October 9, 2025, https://arxiv.org/abs/2305.10601
16. What is Tree Of Thoughts Prompting? | IBM, accessed October 9, 2025, https://www.ibm.com/think/topics/tree-of-thoughts
17. Tree of Thoughts (ToT) - Prompt Engineering Guide, accessed October 9, 2025, https://www.promptingguide.ai/techniques/tot
18. [2305.08291] Large Language Model Guided Tree-of-Thought - arXiv, accessed October 9, 2025, https://arxiv.org/abs/2305.08291
19. Self-correction in LLM calls: a review - The Elder Scripts, accessed October 9, 2025, https://theelderscripts.com/self-correction-in-llm-calls-a-review/
20. Introduction to Self-Criticism Prompting Techniques for LLMs, accessed October 9, 2025, https://learnprompting.org/docs/advanced/self_criticism/introduction
21. Training Language Models to Self-Correction via Reinforcement Learning: A Deep Dive into SCoRe with Code Implementation using PyTorch. | by Devmallya Karar | Medium, accessed October 9, 2025, https://medium.com/@devmallyakarar/training-language-models-to-self-correction-via-reinforcement-learning-a-deep-dive-into-score-with-ff85421b4186
22. Understanding the Dark Side of LLMs' Intrinsic Self-Correction - arXiv, accessed October 9, 2025, https://arxiv.org/html/2412.14959v1
23. Mastering Self-Criticism Prompting: Techniques, Examples, and Use Cases - RAIA A.I., accessed October 9, 2025, https://raiabot.com/blog/Mastering_SelfCriticism_Prompting_Techniques_Examples_and_Use_Cases.html