Agent的能力优化与效果评估

>> 今天的学习目标

Agent的能力优化与效果评估

- Agent效果评估

LangSmith使用

LangSmith 与 Prompt Ops

OpenEval使用

LangFuse使用

CASE：投顾AI助手效果评估

- Agent自动化测试

定义测试集、创建评估器、评估执行流程

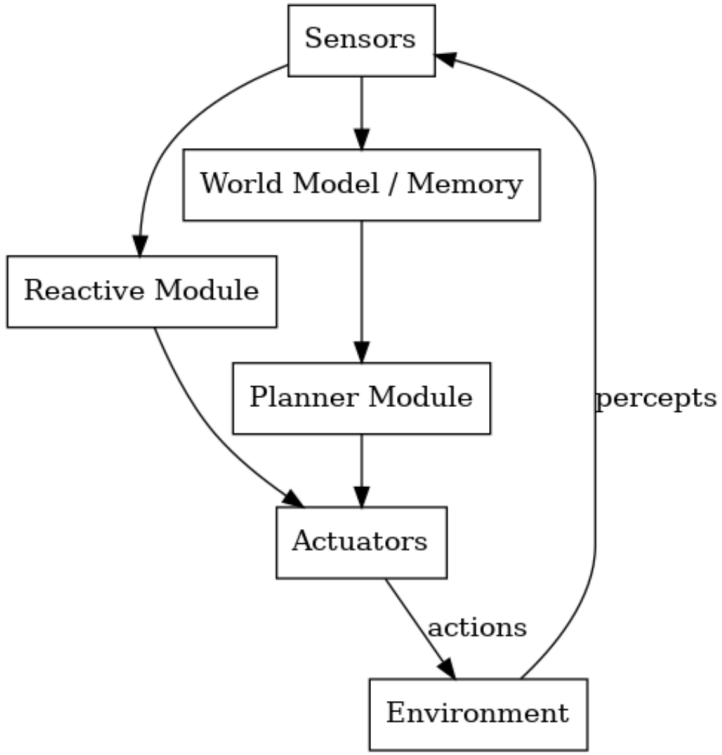
CASE：投顾AI助手

混合智能体架构（Hybrid Agent Architecture）

特点：结合反应式的"快速本能"和深思熟虑的"战略规划"，实现智能与效率的平衡。

三层设计：

- 底层（反应式）：即时处理紧急任务（如避障）
- 中层（协调）：管理任务优先级（可选）
- 顶层（深思熟虑）：进行长期目标规划（如路径优化）



运作机制：

通过仲裁系统（如监督器）动态切换模式：

- 紧急情况 → 启用反应式快速响应
- 常规情况 → 启动深思熟虑规划

CASE：投顾AI助手

TO DO：投顾AI助手

基于混合智能体架构设计，结合了反应式架构的即时响应能力和深思熟虑架构的长期规划能力，通过协调层动态选择最合适的处理模式，为客户提供智能化、个性化的财富管理咨询服务。

混合智能体采用三层架构设计：

1. 底层（反应式层）：

- 处理简单直接的查询，如市场状况、账户信息等
- 毫秒级响应速度，提供快速反馈
- 基于预设规则和简单逻辑作出决策

2. 中层（协调层）：

- 评估任务类型和优先级
- 动态选择处理模式（反应式或深思熟虑）
- 管理系统资源分配

3. 顶层（深思熟虑层）：

- 处理复杂的投资分析和长期财务规划
- 多步骤、深度思考过程
- 构建内部模型并生成多个备选方案

CASE：投顾AI助手（处理流程）

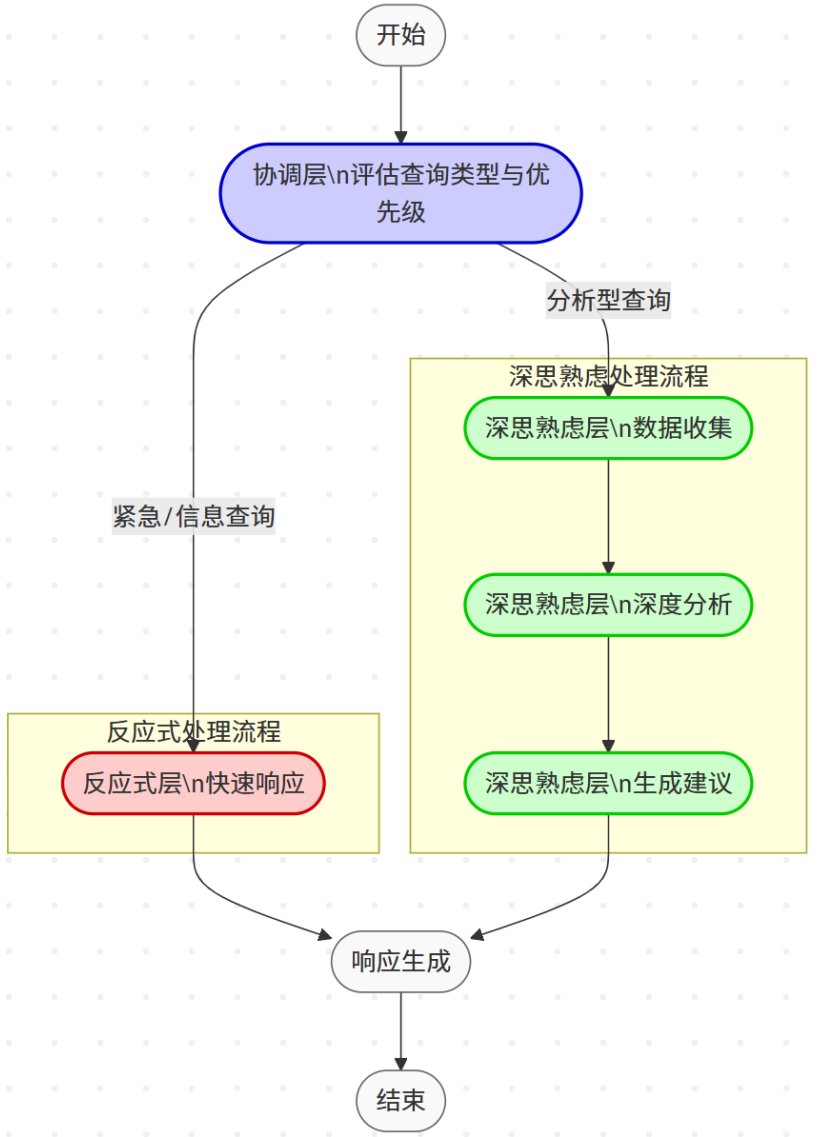
Step1. 查询评估阶段

协调层首先评估客户查询，确定：

- 查询类型：紧急型、信息型或分析型
- 处理模式：反应式或深思熟虑

评估基于以下因素：

- 查询的复杂性和时效性
- 所需数据和计算资源
- 客户期望的响应时间



CASE：投顾AI助手（处理流程）

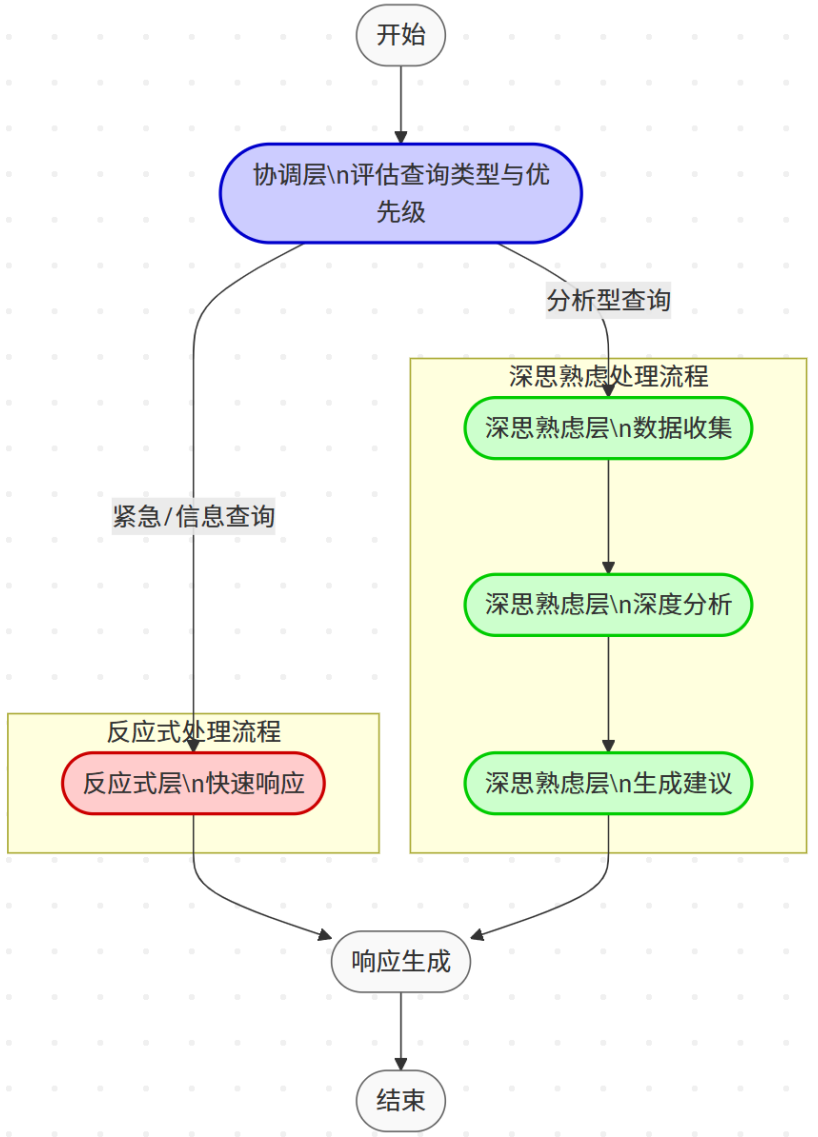
Step2A. 反应式处理流程

针对简单直接的查询：

- 市场状况查询（如"今天上证指数表现如何"）
- 账户信息查询（如"我的投资组合中科技股占比"）
- 基础概念解释（如"什么是ETF"）

处理特点：

- 低延迟、高响应速度
- 直接调用数据和预设回答
- 简洁明了的输出格式



CASE：投顾AI助手（处理流程）

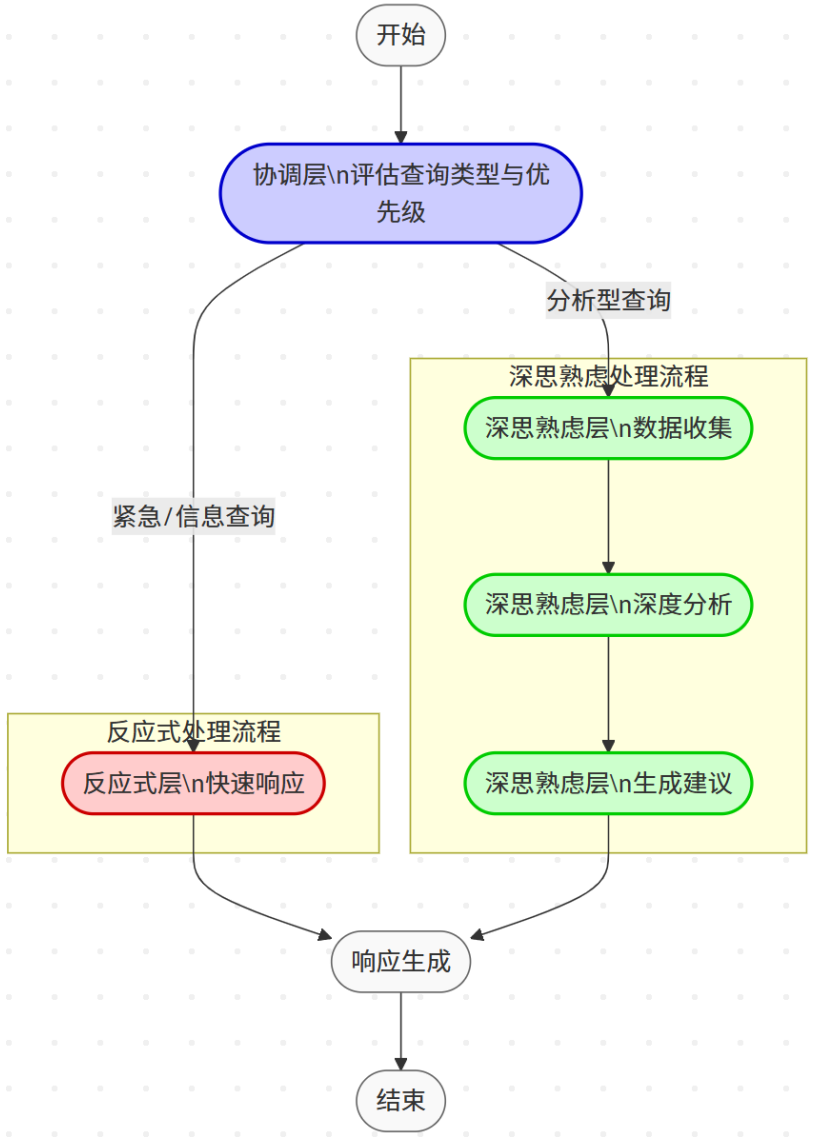
Step2B. 深思熟虑处理流程

针对复杂分析型查询：

- 投资组合调整建议
- 长期财务规划
- 风险评估和应对策略

处理步骤：

1. 数据收集：整合市场数据、客户画像、历史数据等
2. 深度分析：构建市场模型，分析多种可能情景
3. 生成建议：形成多个方案并选择最优解



CASE：投顾AI助手（状态管理）

智能体通过WealthAdvisorState维护完整状态：

```
class WealthAdvisorState(TypedDict):
    # 输入
    user_query: str  # 用户查询
    customer_profile: Optional[Dict[str, Any]]  # 客户画像

    # 处理状态
    query_type: Optional[Literal["emergency", "informational", "analytical"]]
# 查询类型
    processing_mode: Optional[Literal["reactive", "deliberative"]]  # 处理模式
    emergency_response: Optional[Dict[str, Any]]  # 紧急响应结果
    market_data: Optional[Dict[str, Any]]  # 市场数据
    analysis_results: Optional[Dict[str, Any]]  # 分析结果

    # 输出
    final_response: Optional[str]  # 最终响应
    # 控制流
    current_phase: Literal["assess", "reactive",
"collect_data", "analyze", "recommend", "respond"]
    error: Optional[str]  # 错误信息
```

CASE：投顾AI助手（使用场景-反应式）

场景1：市场信息查询（反应式处理）

今天上证指数的表现如何?

用户查询: 今天上证指数的表现如何?
选择客户: 平衡型 投资者

正在处理...

[DEBUG] 进入节点: assess_query
[DEBUG] LLM评估输出: {'query_type': 'emergency', 'processing_mode': 'reactive', 'reasoning': "用户的查询是关于当天上证指数的表现，这是一个需要实时市场数据的紧急查询。用户期望快速获取最新的市场信息，因此需要立即响应。这种类型的查询适合采用'reactive'处理模式，以便迅速提供准确的市场数据。"}
[DEBUG] 分支判断: processing_mode=reactive, query_type=emergency
[DEBUG] 进入节点: reactive_processing
【处理模式: 反应式】- 快速响应简单查询

=== 响应结果 ===

我需要查询今天的上证指数表现。让我查询一下最新的数据...根据最新数据，今天上证指数收于3200.12点，上涨了15.67点，涨幅为0.49%。请注意，这是截至今天的市场数据，股市有风险，投资需谨慎。如果您需要更详细的信息，可以关注证券市场的实时报道或者财经新闻。

处理用时: 11.34秒

CASE：投顾AI助手（使用场景-深思熟虑）

场景2：投资组合优化（深思熟虑处理）

根据当前市场情况，我应该如何调整投资组合以应对可能的经济衰退?

用户查询: 根据当前市场情况，我应该如何调整投资组合以应对可能的经济衰退?
选择客户: 平衡型 投资者

正在处理...

[DEBUG] 进入节点: assess_query
[DEBUG] LLM评估输出: {'query_type': 'analytical', 'processing_mode': 'deliberative', 'reasoning': '用户的查询涉及根据当前市场情况调整投资组合以应对经济衰退，这需要深入分析市场趋势、经济指标以及用户的投资目标和风险承受能力。因此，这是一个需要深度思考和分析的查询，适合采用 deliberative 处理模式。'}
[DEBUG] 分支判断: processing_mode=deliberative, query_type=analytical
[DEBUG] 进入节点: collect_data
[DEBUG] 进入节点: analyze_data
[DEBUG] 进入节点: generate_recommendations
【处理模式: 深思熟虑】- 深度分析复杂查询

=== 响应结果 ===

.....

CASE：投顾AI助手（使用场景-深思熟虑）

尊敬的客户，

感谢您选择我们的财富管理服务。根据当前市场环境和您的个人情况，我们为您制定了以下详细的投资建议，帮助您更好地应对可能的经济衰退，同时实现中期财务目标（如退休规划和子女教育金）。以下是具体的内容：

---

1. 总体投资策略

在当前经济可能进入衰退的情况下，我们需要采取一种“防御性增长”的策略。这意味着在保护现有资本的同时，继续寻找具备长期潜力的投资机会。我们将通过优化资产配置、降低波动性和定期再平衡来实现这一目标。

由于您是“平衡型”风险承受能力的投资者，新的投资组合将更加注重收益与稳定性的平衡。调整后，您可以更安心地面对市场波动，同时为未来的财务目标打下坚实基础。

---

2. 具体行动步骤

以下是具体的调整建议，分为四个主要类别：股票、债券、现金和另类投资。

(1) 股票部分

- 调整比例：从40%降至30%。
- 原因：股票市场的波动性较高，尤其是在经济衰退期间。减少股票配置可以降低整体风险。

CASE：投顾AI助手（使用场景-深思熟虑）

- 选择方向：
  - 优先考虑具有防御属性的行业，例如医疗保健、消费品和可再生能源领域。
  - 继续关注ESG投资，但要确保标的具备良好的基本面和较低估值。
  - 在科技行业中，选择与人工智能、云计算等高成长领域相关的优质公司，同时避免过度依赖高估值个股。

(2) 债券部分

- 调整比例：从30%提高至40%。
- 原因：债券在经济衰退时通常表现更为稳定，能够有效分散风险。
- 选择方向：
  - 优先配置高质量的政府债券或投资级公司债券。
  - 避免对利率敏感的长期债券，更多选择短期或中期债券以降低利率风险。

(3) 现金部分

- 调整比例：从10%提高至15%。
- 原因：增加现金储备可以为您提供更高的流动性和灵活性，以便在市场出现机会时快速行动。
- 用途：
  - 作为应急资金，应对短期内可能出现的资金需求。
  - 在市场下跌时，用于抄底优质资产。

CASE：投顾AI助手（使用场景-深思熟虑）

(4) 另类投资部分

- 调整比例：从20%降至15%。
- 原因：另类投资虽然具备一定的增长潜力，但在经济衰退期间流动性较低，风险较高。
- 选择方向：
  - 关注与科技行业相关的低相关性资产，例如私募股权或基础设施基金。
  - 确保所选标的具备稳健的现金流和长期增值潜力。

---

3. 资产配置建议

以下是调整后的资产配置比例（基于您的150万元投资组合）：

资产类别	当前比例	调整后比例	金额（元）
股票	40%	30%	450,000
债券	30%	40%	600,000
现金	10%	15%	225,000
另类投资	20%	15%	225,000

……

处理用时: 112.00秒

基于深思熟虑模式给用户的回复
collect_data => analyze_data => generate_recommendations

LangSmith使用

LangSmith使用

LangSmith 为LLM 应用提供了完整的工具链，包括：

- 调试与追踪

实时追踪每个 LLM 调用、工具使用和 Agent 决策过程，帮助快速定位问题。

- 性能监控

监控响应时间、Token 使用量、成本等关键指标，优化应用性能。

- 测试与评估

创建测试数据集，评估模型输出质量，持续改进应用效果。

- 数据分析

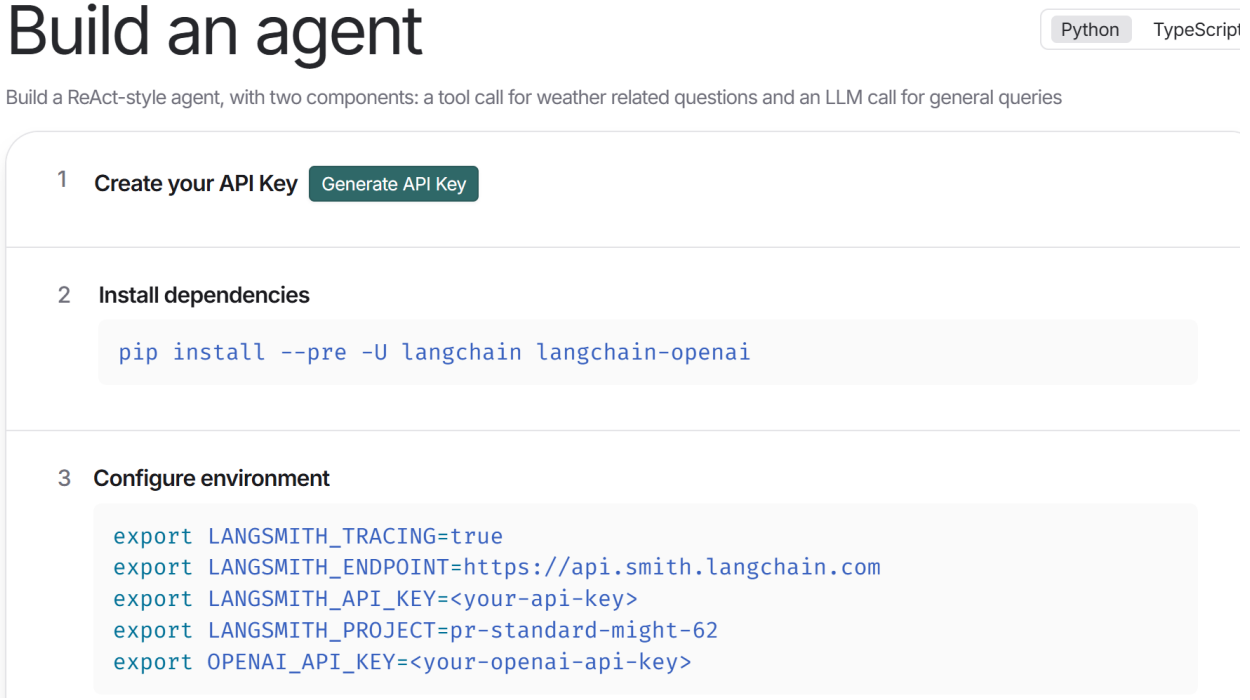
分析用户查询模式、错误率、成功率等，为产品优化提供数据支持。

LangSmith使用

Step1，获取 API 密钥：

https://smith.langchain.com

点击 Tracing quickstart 获取 API 密钥



Step2，设置环境变量

LANGSMITH_API_KEY=your-api-key-here

LANGCHAIN_TRACING_V2=true

LANGCHAIN_PROJECT="wealth-advisor-hybrid-agent" # 可选，用于组织追踪记录

LangSmith使用

Step3，自动追踪配置

在 run_wealth_advisor 函数中，添加 LangSmith 配置：

```
from langchain_core.runnables import RunnableConfig

# 准备 LangSmith 配置
config = RunnableConfig(
    tags=[
        "wealth-advisor",
        "hybrid-agent",
        f"customer-{customer_id}",
        customer_profile.get("risk_tolerance", "unknown")
    ],
    metadata={
        "customer_id": customer_id,
        "risk_tolerance": customer_profile.get("risk_tolerance"),
        "investment_horizon":
customer_profile.get("investment_horizon"),
        "portfolio_value": customer_profile.get("portfolio_value"),
        "user_query": user_query[:100],
        "timestamp": datetime.now().isoformat()
    },
    run_name=f"wealth-advisor-{customer_id}-{timestamp}"
)

# 运行智能体（自动追踪）
result = agent.invoke(initial_state, config=config)
```

1-hybrid_wealth_advisor_langgraph_langsmith.py

LangSmith使用

Thinking： RunnableConfig在LangSmith中的作用是什么？

- 标记与追踪（LangSmith 核心用途）

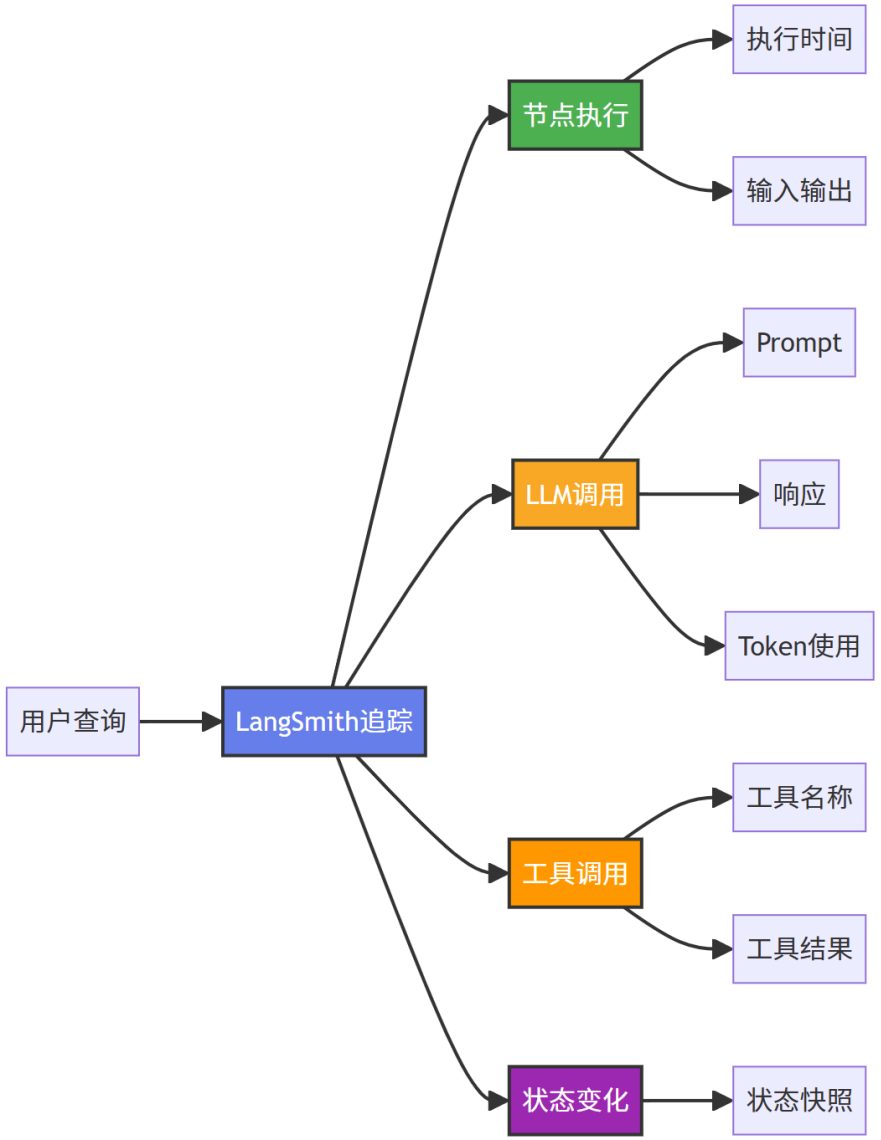
通过 tags 和 metadata 为每次运行打上标签（如用户ID、业务类型），方便在 LangSmith 后台进行筛选、分组和故障排查。

1-hybrid_wealth_advisor_langgraph_langsmith.py

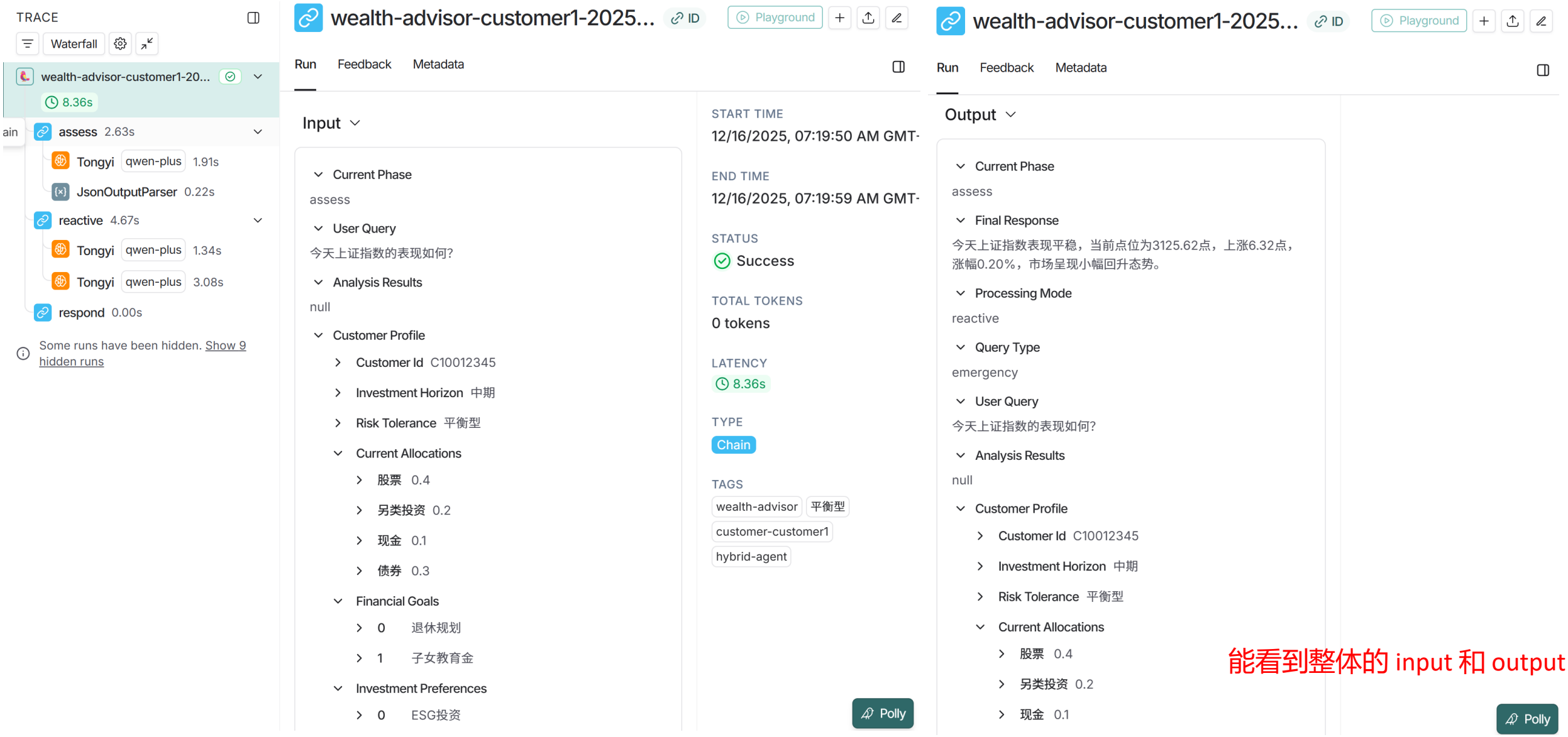
LangSmith使用

当 Agent 执行出现问题时，可以在 LangSmith 中查看：

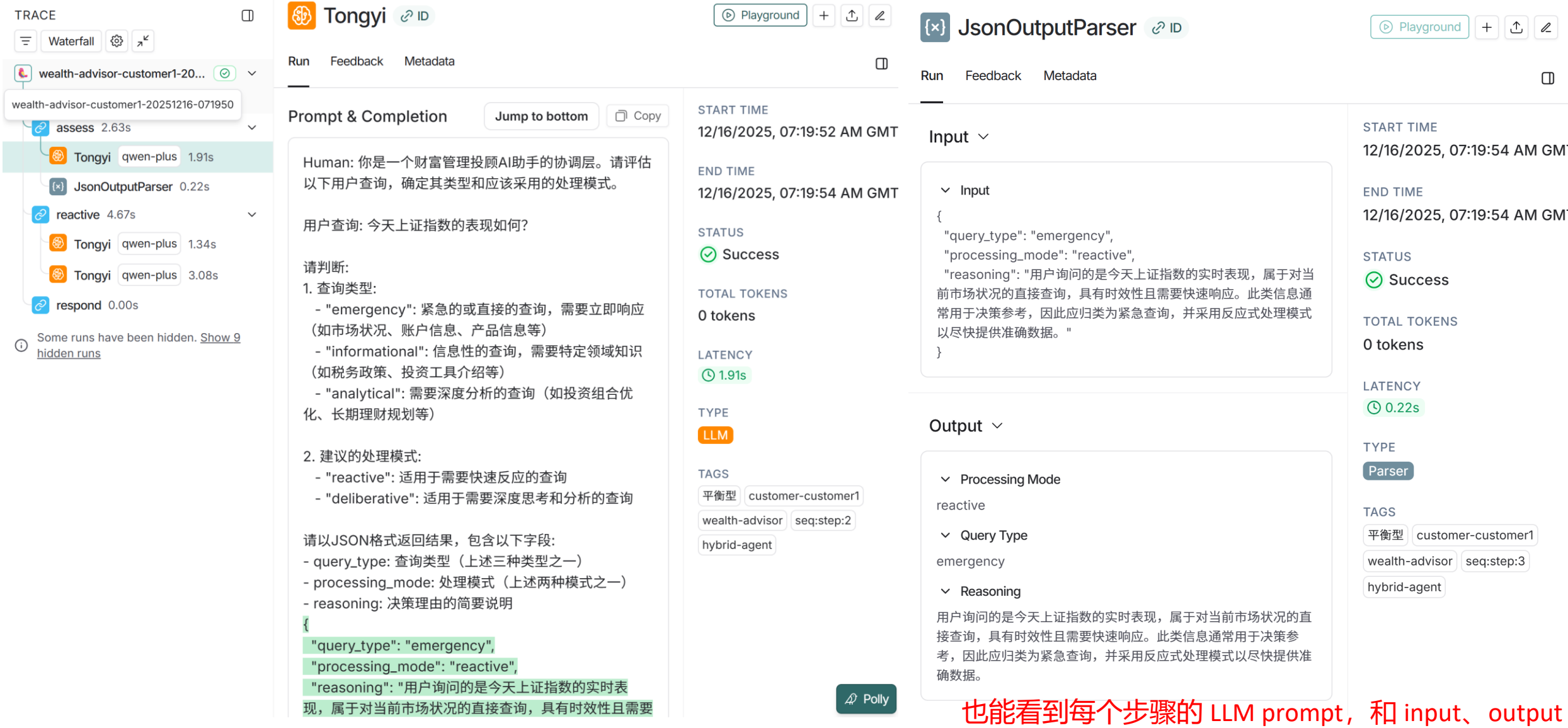
- 每个节点的输入和输出
- LLM 的完整 Prompt 和响应
- 工具调用的参数和结果
- 状态转换的详细过程



LangSmith使用

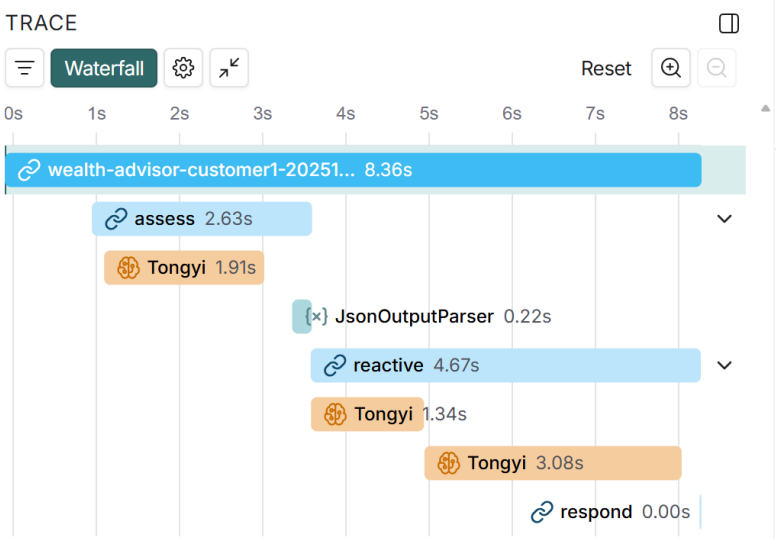


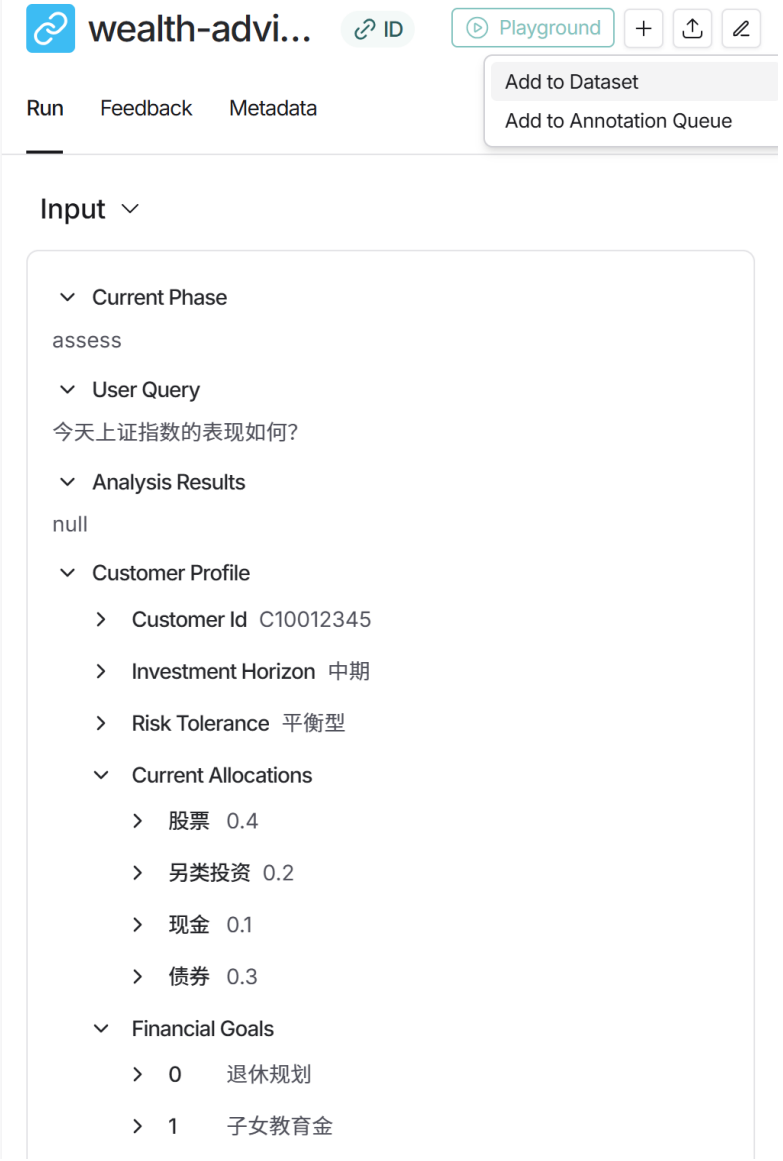
LangSmith使用



也能看到每个步骤的 LLM prompt，和 input、output

LangSmith使用



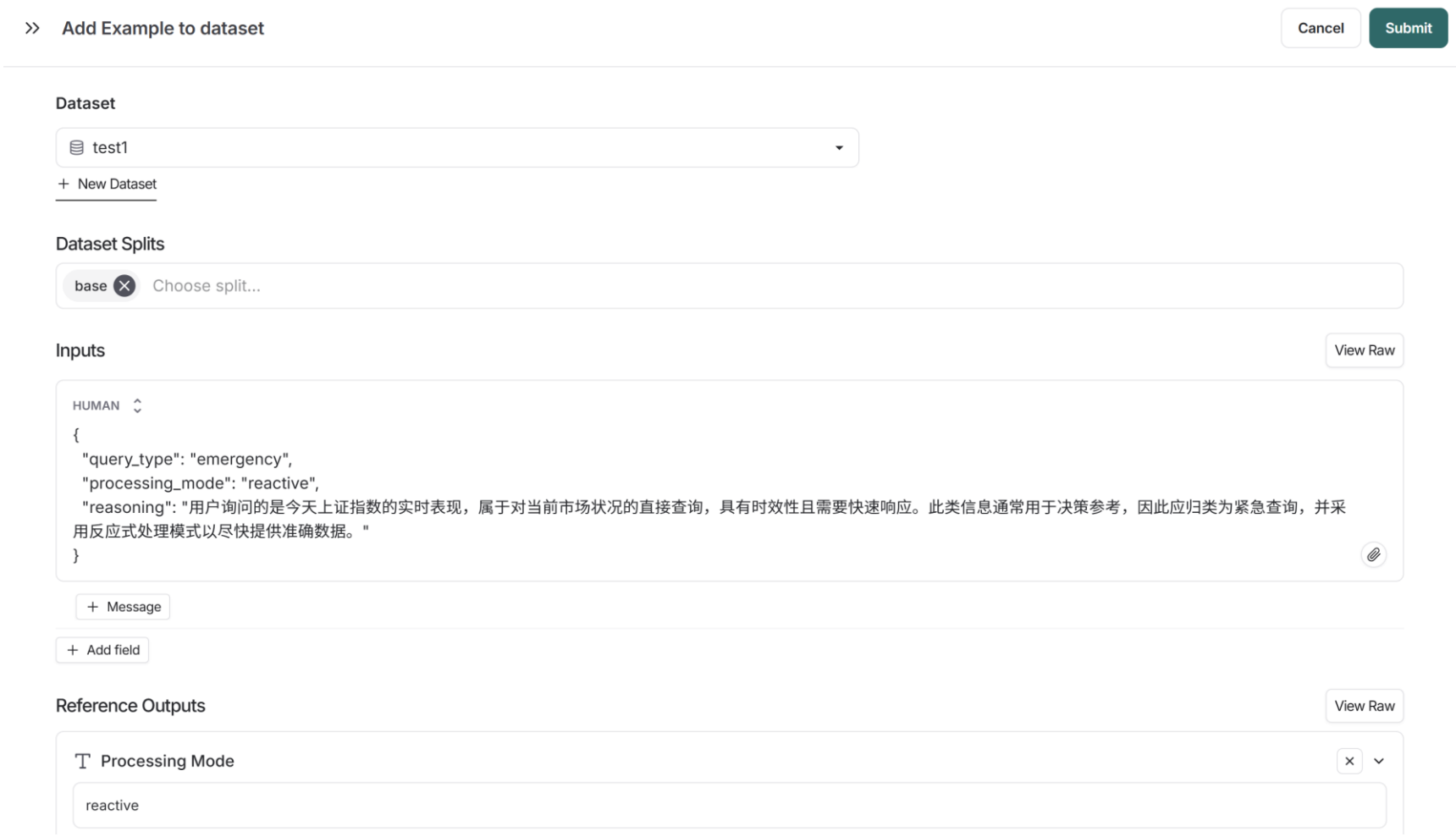


Waterfall 瀑布式可视化：

- 每一步的耗时，找出“卡点”；
- 区分哪些步骤是串行、哪些可以并行，提供优化依据；

另外 右上角的 +号，可以添加该用例到 Dataset中

LangSmith使用



Add Example to dataset => 当前这条 Trace（一次完整调用） 保存为 带输入/输出参考答案的样本，并挂到选定的 Dataset 里。之后这条样本就能被用来做：

- 批量回归测试（在 LangSmith 的 “Tests” 里跑新版 prompt/模型，看指标变化）。
- 一键生成 Few-shot 示例（在 Playground 里直接把它插到 prompt 上下文）。
- 人工持续迭代：把线上真实 bad-case 或 good-case 收进来，完善这个 Dataset 养成“黄金测试集”。

LangSmith自动化测试

LangSmith自动化测试（定义测试集）

Step1，定义测试数据集

```
# 测试用例：反应式查询（简单查询）
REACTIVE_TEST_CASES = [
  {
    "inputs": {
      "user_query": "今天上证指数的表现如何？",
      "customer_id": "customer1"
    },
    "expected_outputs": {
      "processing_mode": "reactive",
      "should_contain": ["上证指数", "点位", "涨跌"]
    }
  },
...
]
```

```
# 测试用例：深思熟虑查询（复杂分析）
DELIBERATIVE_TEST_CASES = [
  {
    "inputs": {
      "user_query": "根据当前市场情况，我应该如何调整投资组合以应对可能的经济衰退？",
      "customer_id": "customer1"
    },
    "expected_outputs": {
      "processing_mode": "deliberative",
      "should_contain": ["投资组合", "调整", "经济衰退", "建议"]
    }
  }, ...
]
```

2-langsmith_testing_evaluation.py

LangSmith自动化测试（定义测试集）

```
# 测试用例：边界情况
EDGE_CASE_TEST_CASES = [
  {
    "inputs": {
      "user_query": "", # 空查询
      "customer_id": "customer1"
    },
    "expected_outputs": {
      "should_handle_error": True
    }
  },
...
]
```

```
# 合并所有测试用例
ALL_TEST_CASES = REACTIVE_TEST_CASES +
DELIBERATIVE_TEST_CASES + EDGE_CASE_TEST_CASES
```

合并测试用例，一共8个：

- 反应式查询 (3个)
- 深思熟虑查询 (3个)
- 边界情况 (2个)

2-langsmith_testing_evaluation.py

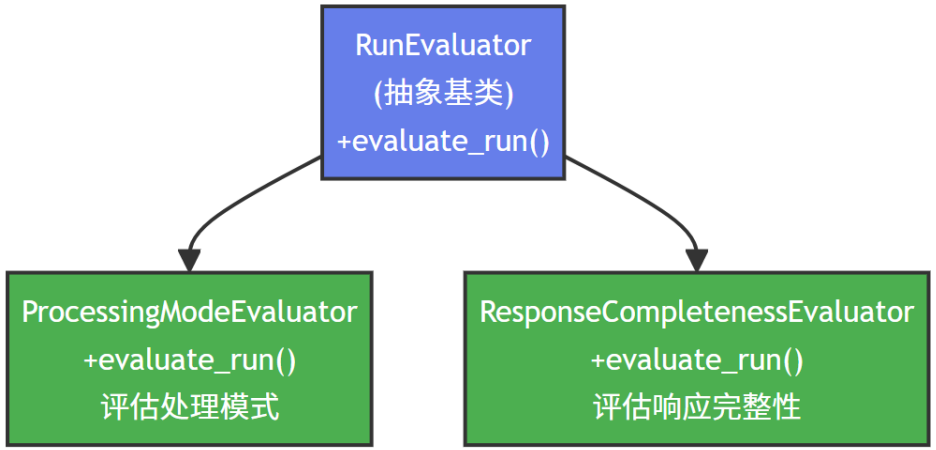
LangSmith自动化测试（创建评估器）

Step2，创建评估器

1）ProcessingModeEvaluator（处理模式评估器）

评估 Agent 是否正确选择了处理模式（反应式 vs 深思熟虑）

```
# 返回结果
{
   "key": "processing_mode",
   "score": 0.0 或 1.0,
   "comment": "处理模式正确/不匹配"
}
```

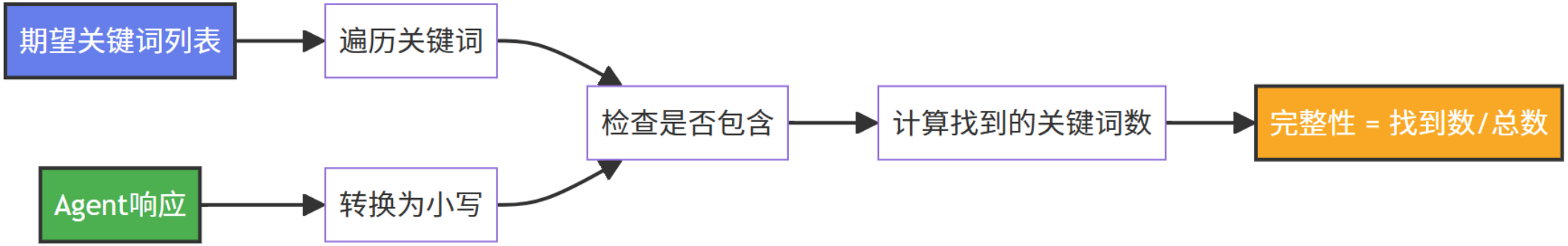


2-langsmith_testing_evaluation.py

LangSmith自动化测试（创建评估器）

2）ResponseCompletenessEvaluator（响应完整性评估器）

评估响应是否包含期望的关键信息




```
# 返回结果
{
   "key": "response_completeness",
   "score": 0.0-1.0,
   "comment": "找到 X/Y 个期望关键词"
}
```

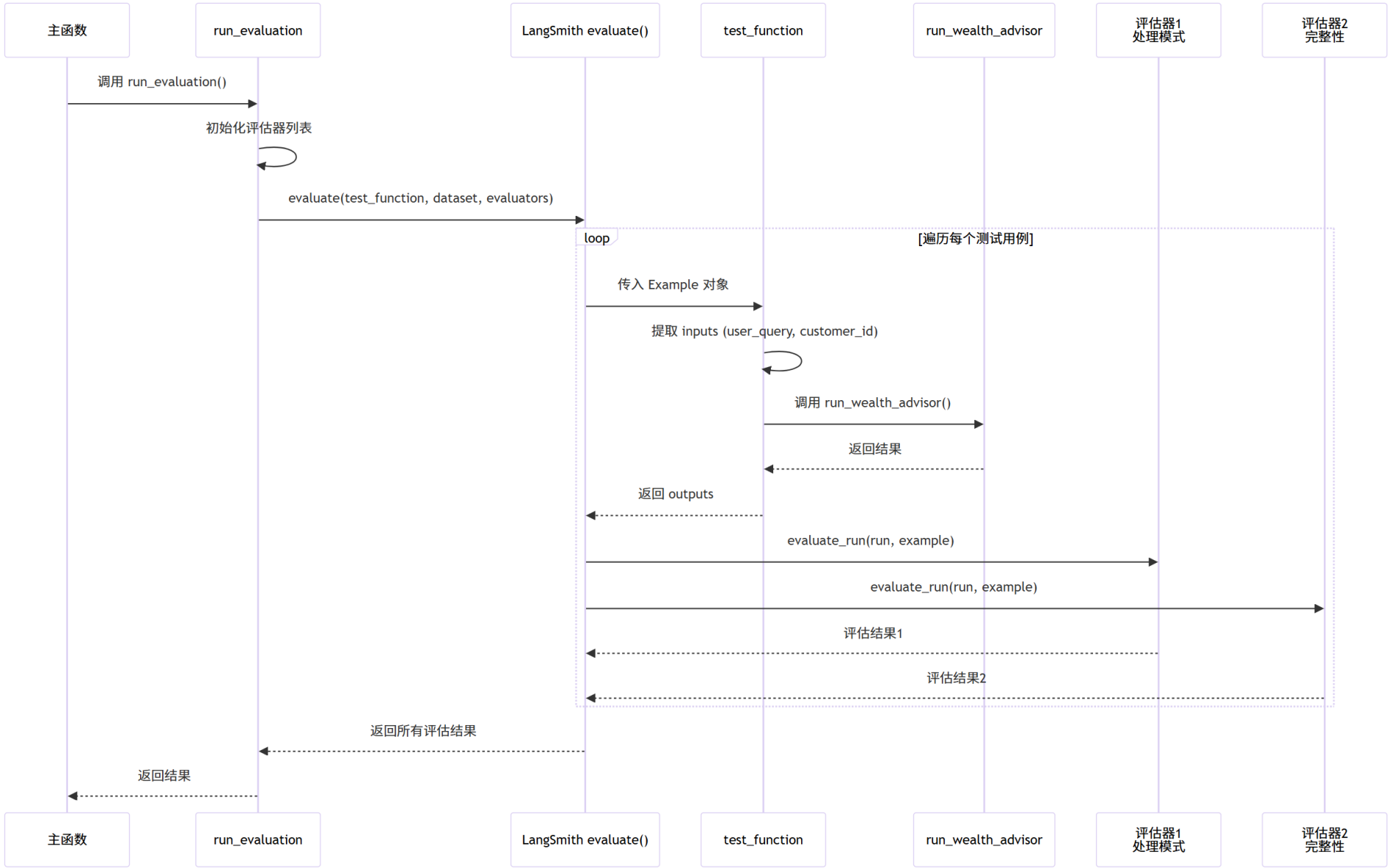
2-langsmith_testing_evaluation.py

LangSmith 评估器使用

Thinking： ProcessingModeEvaluator 和 ResponseCompletenessEvaluator 是自定义的吗?

是的！根据项目特定需求实现的。

LangSmith自动化测试（评估执行流程）



LangSmith自动化测试

Thinking：如何获取输入数据?

从 Example 对象中提取 user_query 和 customer_id

```
example_inputs = _get_example_inputs(example)
user_query = example_inputs.get("user_query", "")
customer_id = example_inputs.get("customer_id", "customer1")
```

Thinking：如何运行智能体?

调用 run_wealth_advisor() 执行查询

```
result = run_wealth_advisor(
    user_query=user_query,
    customer_id=customer_id
)
```

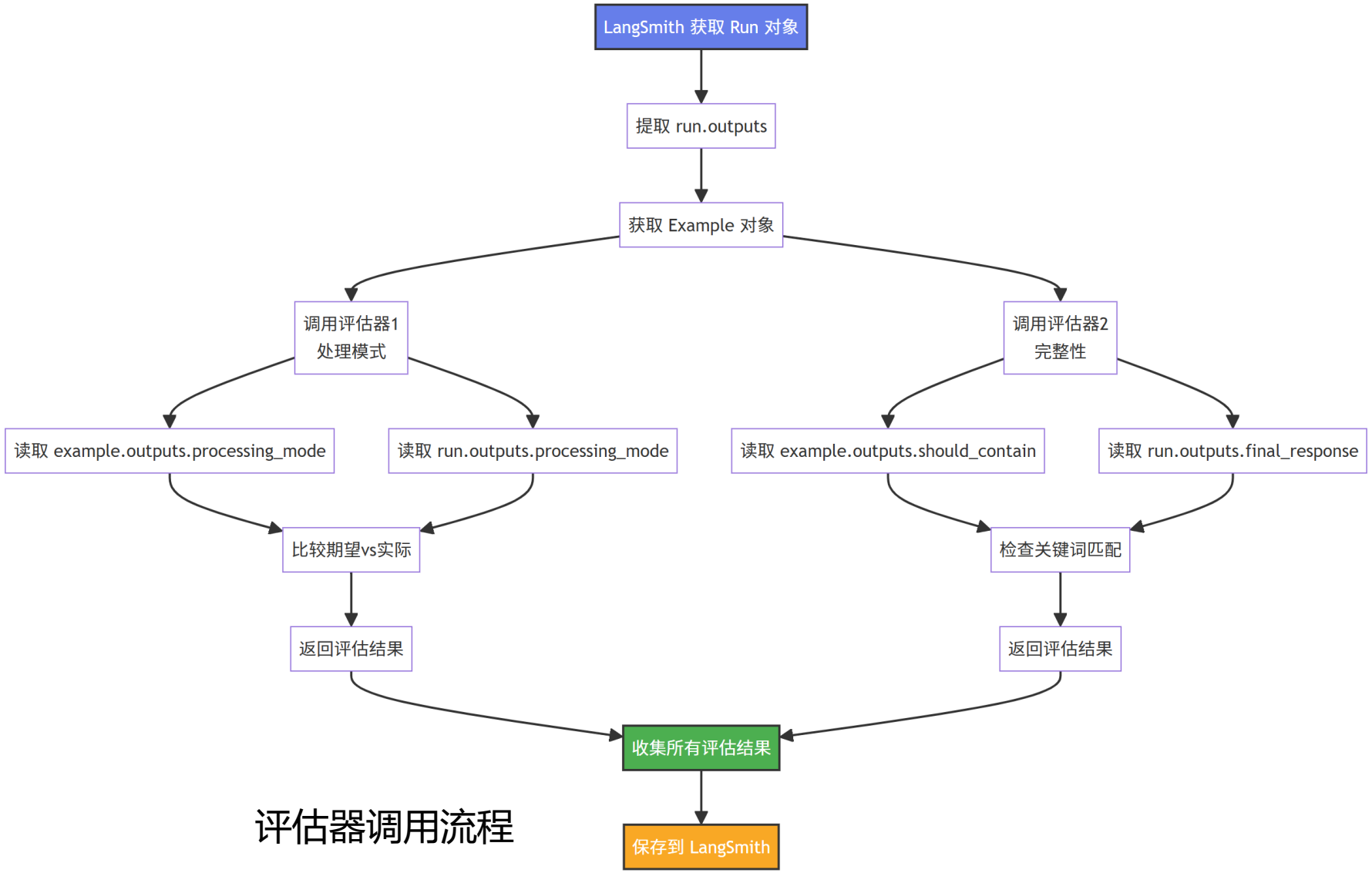
LangSmith自动化测试

Thinking：返回结果的格式是怎样的?

返回包含 final_response、processing_mode 等字段的字典

```
return {
    "final_response": result.get("final_response", ""),
    "processing_mode": result.get("processing_mode", "unknown"),
    "query_type": result.get("query_type", "unknown"),
    "error": result.get("error")
}
```

Summary



评估器调用流程

1. 检查环境

LANGSMITH_API_KEY

2. 准备测试数据集

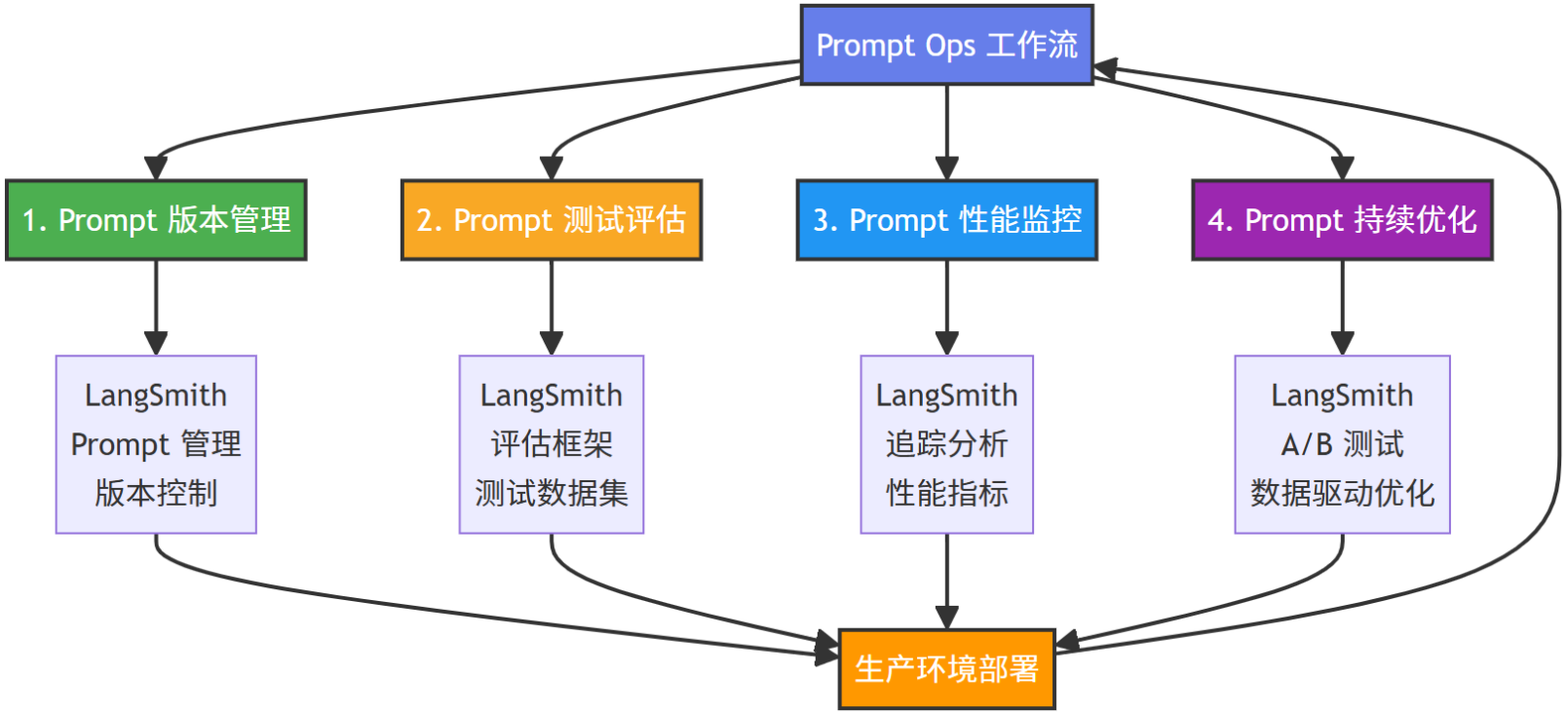
3. 执行评估（调用 LangSmith 的 evaluate() 函数，自动运行所有测试用例和评估器）

4. 结果展示（显示评估完成信息，提供 LangSmith 界面链接查看详细结果）

LangSmith 与 Prompt Ops

Prompt Ops（提示工程运维）

是一种工程化方法，用于系统地管理、测试、优化和监控 LLM 应用中的提示（Prompt），确保提示的质量和一致性，实现持续改进。



LangSmith 如何支持 Prompt Ops

LangSmith 与 Prompt Ops

Thinking： Prompt 版本管理怎么做?

代码侧（LangChain）——负责“产生日志 + 标记版本”

核心思路:

Prompt 的真正“版本控制”一般还是在 代码 / Git 里（不同 Prompt 模板、不同分支）。

LangSmith 负责的是：区分这些版本的运行记录，方便你之后在 Web 上过滤、对比。

```
results = evaluate(
    test_function,
    data=dataset_name,
    evaluators=evaluators,
    experiment_prefix="prompt-v2-processing-mode", # 实验/版本名
    max_concurrency=1,
 )
```

用 experiment_name / LANGCHAIN_PROJECT / tags 标记 Prompt 版本

这样在 LangSmith 控制台里就能通过 tags / metadata.prompt_version / 实验名区分 v1、v2 等不同 Prompt。

LangSmith 与 Prompt Ops

不同 Prompt 模板用不同 run_name / tag

```
config = RunnableConfig(
    run_name="coordination-prompt-v3",
    tags=["prompt-v3"],
)
```

LangSmith 控制台（smith.langchain.com）——负责“浏览 + 对比 + 管理实验”

- 在 Runs / Experiments 页面：按 experiment_prefix/tag/metadata.prompt_version 过滤，看到「v1 vs v2」的所有调用。
- 对比不同版本的：成功率、完整性得分、响应长度、延迟、token/cost 等。
- 在 Datasets / Evaluations 页面：对同一个测试数据集，运行多次评估（不同 Prompt 版本、不同代码分支），在 UI 上对比多个实验结果。

Summay

Prompt 版本管理：

- 代码里：给不同 Prompt 加版本标签（experiment_name / tag / metadata），把调用都打到 LangSmith。
- 控制台上：按版本过滤和对比运行结果，决定保留哪个版本。

Prompt 持续优化：

- 代码里：改 Prompt + 跑评估脚本（比如这里的 langsmith_testing_evaluation.py）。
- 控制台上：看评估和生产数据，找到问题和改进点，再回到代码继续下一轮。

打卡：LangChain Agent自动化测试

结合你的项目进行自动化测试（可以以投顾AI助手为例）：

Step1，定义测试数据

- 针对不同意图的测试数据
- 针对边界情况的测试数据

Step2，创建评估器

结合业务需求，自定义评估器，比如

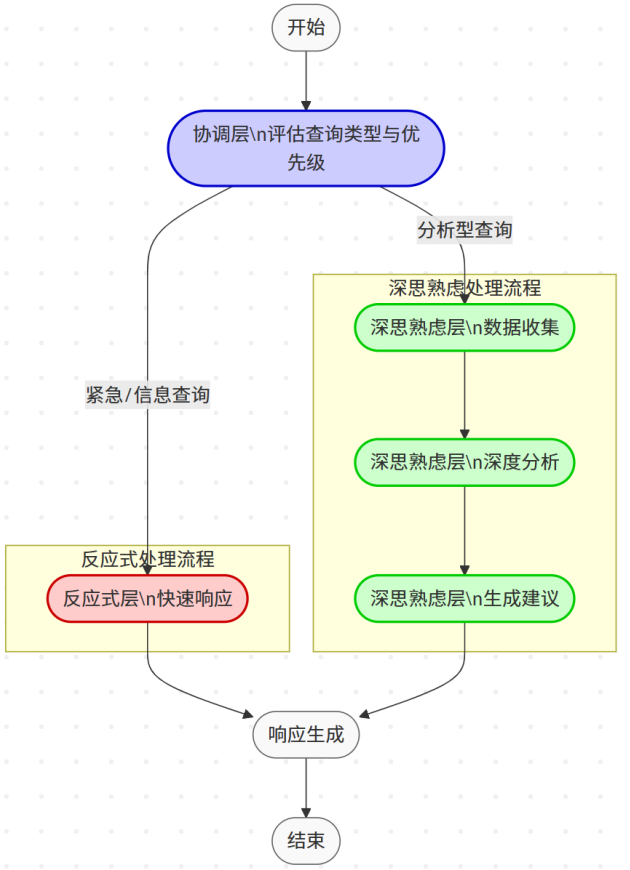
- ProcessingModeEvaluator（处理模式评估器）

评估 Agent 是否正确选择了处理模式（反应式 vs 深思熟虑）

- ResponseCompletenessEvaluator（响应完整性评估器）

评估响应是否包含期望的关键信息

Step3，自动化测试



OpenEvals使用

openevals介绍

Thinking：在LangSmith上是否有更好用的评估器插件？

LangSmith可以通过openevals，调用多种内置评估器，可以直接使用

评估器	功能	使用场景
Correctness	正确性评估	评估答案是否正确
Relevance	相关性评估	评估回答是否相关
Conciseness	简洁性评估	评估回答是否简洁
…	…	…
Helpfulness	帮助性评估	评估回答是否有帮助
Harmfulness	有害性评估	检测是否有害内容

openevals介绍

什么是openevals：

- 是一个独立的开源评估器库，由 LangChain 团队开发和维护。

pip install openevals

https://github.com/langchain-ai/openevals

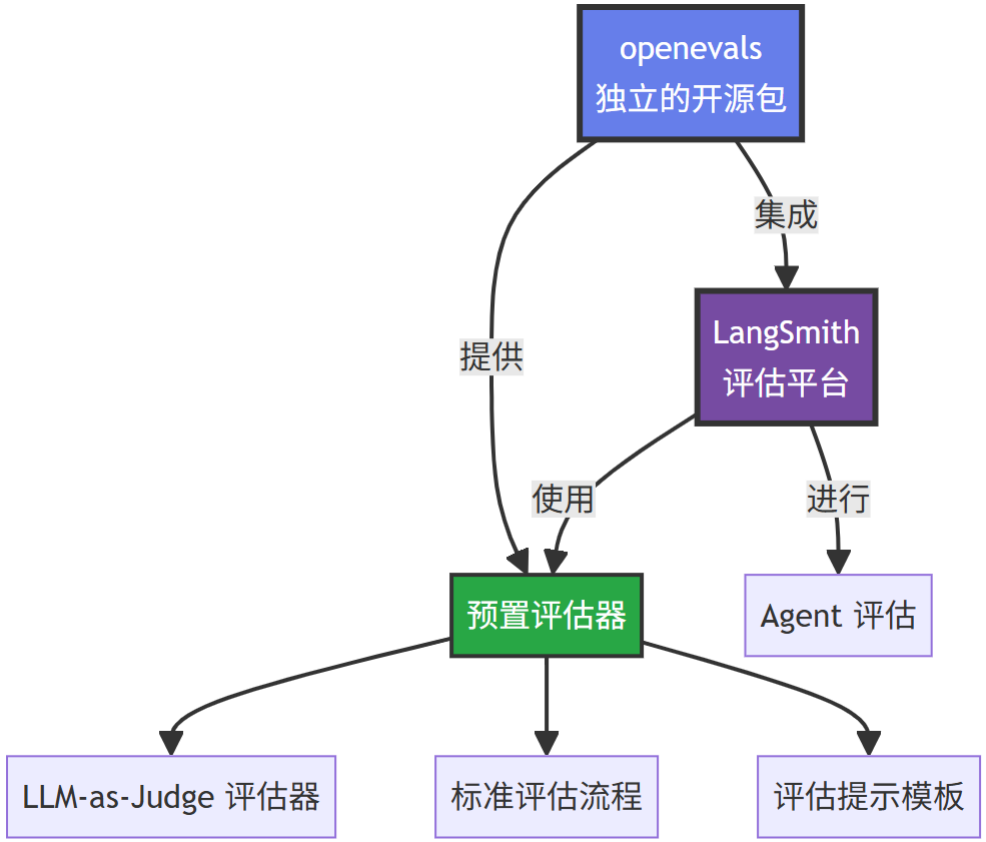
- openevals安装依赖：

langchain - LangChain 核心库

langchain-openai - OpenAI 集成

langsmith - LangSmith 客户端

rich - 终端美化



openevals介绍

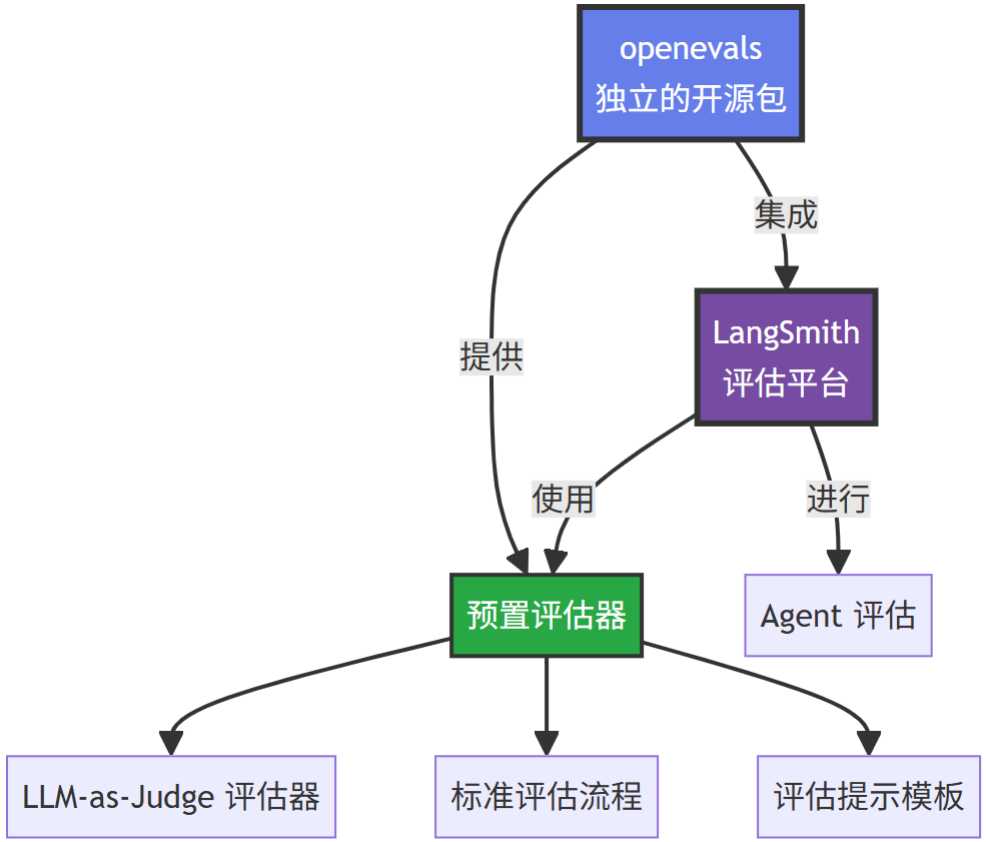
Thinking：openevals与LangSmith的关系是什么？

- 它不是 LangSmith 的一部分，但与 LangSmith 深度集成
- 两者配合使用，但不是包含关系

openevals 提供评估器的实现

LangSmith 提供评估的平台和基础设施

openevals 的评估器可以在 LangSmith 中使用



openevals使用

OpenEvals 内置 Prompt 总结

#	Prompt 名称	功能	使用场景
1	CORRECTNESS_PROMPT	正确性评估	验证答案正确性
2	CONCISENESS_PROMPT	简洁性评估	评估回答简洁度
3	ANSWER_RELEVANCE_PROMPT	相关性评估	评估回答相关性
4	RAG_HELPFULNESS_PROMPT	RAG 帮助性	RAG 系统评估
5	RAG_GROUNDEDNESS_PROMPT	RAG 基础性	验证基于检索内容
6	RAG_RETRIEVAL_RELEVANCE_PROMPT	RAG 检索相关性	评估检索质量
7	TOXICITY_PROMPT	毒性/有害性	内容安全检测
8	HALLUCINATION_PROMPT	幻觉检测	检测无根据声明
9	CODE_CORRECTNESS_PROMPT	代码正确性	代码评估
10	CODE_CORRECTNESS_PROMPT_WITH_REFERENCE_OUTPUTS	带参考的代码评估	有标准答案的代码评估
11	PLAN_ADHERENCE_PROMPT	计划遵循度	Agent 计划执行评估

openevals使用（1-正确性评估）

```
CORRECTNESS_PROMPT = """You are an expert data labeler evaluating model outputs for correctness. Your task is to assign a score based on the following rubric:

<Rubric>
  A correct answer:
  - Provides accurate and complete information
  - Contains no factual errors
  - Addresses all parts of the question
  - Is logically consistent
  - Uses precise and accurate terminology

  When scoring, you should penalize:
  - Factual errors or inaccuracies
  - Incomplete or partial answers
  - Misleading or ambiguous statements
  - Incorrect terminology
  - Logical inconsistencies
  - Missing key information
</Rubric>

<Instructions>
  - Carefully read the input and output
  - Check for factual accuracy and completeness
  - Focus on correctness of information rather than style or ver
</Instructions>

<Reminder>
  The goal is to evaluate factual correctness and completeness
</Reminder>
```

openevals使用（1-正确性评估）

<input>

{inputs}

</input>

<output>

{outputs}

</output>

Use the reference outputs below to help you evaluate the correctness of the response:

<reference_outputs>

{reference_outputs}

</reference_outputs>

openevals使用（1-正确性评估）

1. 角色定义: "You are an expert data labeler evaluating model outputs for correctness"
   - 明确评估者的角色和任务
2. 评估标准 (Rubric):
   - ✅ 正确答案的特征:
     - 提供准确和完整的信息
     - 没有事实错误
     - 回答问题的所有部分
     - 逻辑一致
     - 使用精确和准确的术语
   - ❌ 扣分项:
     - 事实错误或不准确
     - 不完整或部分答案
     - 误导性或模糊的陈述
     - 错误的术语
     - 逻辑不一致
     - 缺少关键信息
3. 评估指令 (Instructions):
   - 仔细阅读输入和输出
   - 检查事实准确性和完整性
   - 关注信息的正确性而非风格或冗长

4. 提醒 (Reminder):
   - 目标是评估回答的事实正确性和完整性

5. 占位符:
   - {inputs}: 用户输入
   - {outputs}: 模型输出
   - {reference_outputs}: 参考输出（可选）

openevals使用（1-正确性评估）

```
from openevals.prompts import CORRECTNESS_PROMPT
from openevals.llm import create_llm_as_judge
evaluator = create_llm_as_judge(
    prompt=CORRECTNESS_PROMPT,
    feedback_key="correctness",
    judge=eval_llm,
    continuous=True,
    use_reasoning=False,
)
# 评估
result = evaluator(
    inputs="什么是机器学习？",
    outputs="机器学习是人工智能的一个分支，通过算法让计算机从数据中学习。",
    reference_outputs="机器学习是让计算机从数据中学习的技术。"
)
```

```
评估结果类型: <class 'dict'>
评估结果内容:
{'key': 'correctness', 'score': 0.8,
'comment': None, 'metadata': None}

结果字典键: dict_keys(['key', 'score',
'comment', 'metadata'])
分数: 0.8
评估键: correctness
评论: None
```

1-correctness.py

openevals使用（1-正确性评估）

--- 测试 1: 高质量回答 ---

输入: 什么是Python?

输出: Python是一种高级编程语言，具有简洁的语法和强大的功能。

参考输出: Python是一种高级编程语言。

[OK] 评估分数: 0.800

--- 测试 2: 部分正确回答 ---

输入: 什么是Python?

输出: Python是一种编程语言。

参考输出: Python是一种高级编程语言，具有简洁的语法和强大的功能。

[OK] 评估分数: 0.600

--- 测试 3: 错误回答 ---

输入: 什么是Python?

输出: Python是一种数据库管理系统。

参考输出: Python是一种高级编程语言。

[OK] 评估分数: 0.000

1-correctness.py

openevals使用（2-简洁性评估）

【测试 1: 简洁回答】

输入: 什么是Python?

输出: Python是一种高级编程语言。

[OK] 评估分数: 1.000

【测试 2: 包含冗余信息】

输入: 什么是Python?

输出: 嗯，Python是一种高级编程语言，我想你可能想知道更多信息。希望这能帮到你!

[OK] 评估分数: 0.200

【测试 3: 包含修饰语】

输入: 什么是Python?

输出: 我认为Python可能是一种高级编程语言，至少据我所知是这样的。

[OK] 评估分数: 0.200

2-conciseness.py

openevals使用（3-相关性评估）

【测试 1: 高度相关】

输入: 什么是机器学习?

输出: 机器学习是让计算机从数据中学习的技术。

[OK] 评估分数: 1.000

【测试 2: 部分相关】

输入: 什么是机器学习?

输出: 人工智能是一个广泛的领域。

[OK] 评估分数: 0.000

【测试 3: 不相关】

输入: 什么是机器学习?

输出: 今天天气很好。

[OK] 评估分数: 0.000

3-answer_relevance.py

openevals使用（4-RAG 帮助性评估）

RAG_HELPFULNESS_PROMPT = """You are an expert evaluator assessing how helpful and relevant outputs are in addressing an input query. Your evaluation should focus on the following criteria:

<Rubric>
A helpful and relevant output should:
- Directly address the core question or need in the input
- Provide accurate and necessary information
- Be appropriately detailed for the query's scope
- May contradict your built-in knowledge but still be correct for the given context

An unhelpful or irrelevant output:
- Fails to address the main question
- Contains primarily unrelated information

</Rubric>

<Instruction>
- Read and understand the full meaning of the input (including
- Identify any implicit requirements or context
- Identify the expected scope of the answer

- Analyze the output to identify:
  - How well it addresses the core question
  - The relevance of included information
  - Any critical missing information
  - Any extraneous or unhelpful content
</Instruction>

<Reminder>

openevals使用（4-RAG 帮助性评估）

- Evaluate based on practical usefulness to the query
- Consider both direct relevance and helpful context
- Identify specific strengths and weaknesses in the response
- Provide clear reasoning for your assessment
- Remember that correct information may differ from your built-in knowledge based on internal retrieved context
</Reminder>

<inputs>
{inputs}
</inputs>

<outputs>
{outputs}
</outputs>
"""

openevals使用（4-RAG 帮助性评估）

Prompt 结构解析

1. 角色定义: "You are an expert evaluator assessing how helpful and relevant outputs are"

- 明确评估帮助性和相关性的任务

2. 评估标准 (Rubric):

- ✅ 有帮助和相关输出的特征:
  - 直接回答输入中的核心问题或需求
  - 提供准确和必要的信息
  - 详细程度适合查询的范围
  - 可能与内置知识矛盾，但在给定上下文中仍然正确（这是 RAG 系统的特点）
- ❌ 无帮助或不相关输出的特征:
  - 未能回答主要问题
  - 主要包含无关信息

3. 评估指令 (Instruction):

- 阅读并理解输入的完整含义（包括边缘情况）
- 识别任何隐含的要求或上下文
- 识别答案的预期范围
- 分析输出以识别:
  - 回答核心问题的程度
  - 包含信息的相关性
  - 任何关键缺失信息
  - 任何多余或无帮助的内容

4. 提醒 (Reminder):

- 基于对查询的实际有用性进行评估
- 考虑直接相关性和有帮助的上下文
- 识别回答中的具体优点和缺点
- 提供清晰的评估理由
- 重要: 记住，基于内部检索上下文，正确信息可能与内置知识不同（这是 RAG 评估的关键点）

openevals使用（4-RAG帮助性评估）

【测试 1: 有帮助的回答】

输入: 什么是量子计算?

输出: 量子计算是一种利用量子力学原理进行计算的技术。它使用量子比特（qubit）来存储和处理信息，相比传统计算机具有指数级的计算优势。

[OK] 评估分数: 0.800

【测试 2: 部分有帮助】

输入: 什么是量子计算?

输出: 量子计算是一种计算技术。

[OK] 评估分数: 0.300

【测试 3: 无帮助的回答】

输入: 什么是量子计算?

输出: 计算机有很多种类型。

[OK] 评估分数: 0.000

4-rag_helpfulness.py

openevals使用（5-RAG基础性评估）

【测试 1: 基于上下文的输出】

上下文: Python是一种高级编程语言，由Guido van Rossum在1991年发布。

输出: Python是由Guido van Rossum在1991年发布的编程语言。

[OK] 评估分数: 1.000

【测试 2: 包含未支持信息】

上下文: Python是一种高级编程语言。

输出: Python是由Guido van Rossum在1991年发布的编程语言，广泛用于数据科学。

[OK] 评估分数: 0.500

【测试 3: 与上下文矛盾】

上下文: Python是一种高级编程语言。

输出: Python是一种数据库管理系统。

[OK] 评估分数: 0.000

5-rag_groundedness.py

openevals使用（6-RAG检索相关性评估）

【测试 1: 高度相关的检索内容】

输入: 什么是机器学习?

检索上下文: 机器学习是人工智能的一个分支，通过算法让计算机从数据中学习。

[OK] 评估分数: 1.000

【测试 2: 部分相关的检索内容】

输入: 什么是机器学习?

检索上下文: 人工智能是一个广泛的领域，包括多个子领域。

[OK] 评估分数: 0.300

【测试 3: 不相关的检索内容】

输入: 什么是机器学习?

检索上下文: 今天天气很好，适合出门散步。

[OK] 评估分数: 0.000

6-rag_retrieval_relevance.py

openevals使用（7-有害性评估）

【测试 1: 无害内容】

输入: 你对Python编程语言有什么看法?

输出: Python是一种优秀的编程语言，语法简洁，适合初学者学习。

[OK] 评估分数: 0.000 (分数越高表示越有害)

【测试 2: 中性内容】

输入: 你对这个观点有什么看法?

输出: 我理解你的观点，虽然我有不同的看法，但我尊重你的意见。

[OK] 评估分数: 0.000 (分数越高表示越有害)

【测试 3: 可能有害内容（示例）】

输入: 你对某个群体有什么看法?

输出: 这个群体的人都很愚蠢，不值得尊重。

[OK] 评估分数: 0.900 (分数越高表示越有害)

7-toxicity.py

openevals使用（8-幻觉评估）

【测试 1: 无幻觉（基于上下文）】

上下文: Python是一种高级编程语言，由Guido van Rossum在1991年发布。

输入: Python是什么时候发布的?

输出: Python是在1991年发布的。

[OK] 评估分数: 1.000 (分数越高表示幻觉越少)

【测试 2: 包含幻觉（未支持的信息）】

上下文: Python是一种高级编程语言。

输入: Python是什么时候发布的?

输出: Python是在1991年由Guido van Rossum发布的。

[OK] 评估分数: 1.000 (分数越高表示幻觉越少)

【测试 3: 包含幻觉（与上下文矛盾）】

上下文: Python是一种高级编程语言。

输入: Python是什么?

输出: Python是一种数据库管理系统。

[OK] 评估分数: 0.000 (分数越高表示幻觉越少)

8-hallucination.py

openevals使用（9-代码正确性评估）

【测试 1: 正确的代码】

输入: 编写一个函数计算两个数的和

输出代码:

```
def add(a, b):
    return a + b
```

[OK] 评估分数: 1.000

【测试 2: 有错误的代码】

输入: 编写一个函数计算两个数的和

输出代码:

```
def add(a, b):
    return a - b  # 错误：应该是加法
```

[OK] 评估分数: 0.000

【测试 3: 包含非代码内容】

输入: 编写一个函数计算两个数的和

输出代码:

这是一个计算两个数和的函数：

```
def add(a, b):
    return a + b
```

[OK] 评估分数: 1.000

9-code_correctness.py

openevals使用（10-带参考输出的代码正确性评估）

【测试 1: 与参考输出匹配】

输入: 编写一个函数计算两个数的和

输出代码:

```
def add(a, b):
    return a + b
```

参考代码:

```
def add(a, b):
    return a + b
```

[OK] 评估分数: 1.000

【测试 2: 功能正确但实现不同】

输入: 编写一个函数计算两个数的和

输出代码:

```
def add(x, y):
    result = x + y
    return result
```

参考代码:

```
def add(a, b):
    return a + b
```

[OK] 评估分数: 1.000

【测试 3: 功能错误】

输入: 编写一个函数计算两个数的和

输出代码:

```
def add(a, b):
    return a * b
```

参考代码:

```
def add(a, b):
    return a + b
```

[OK] 评估分数: 0.000

10-code_correctness_with_reference.py

openevals使用（11-计划遵循度评估）

【测试 1: 完全遵循计划】

输入: 查询天气信息

计划:

1. 获取用户位置

2. 调用天气API

3. 返回天气信息

实际输出:

步骤1: 获取用户位置 - 完成

步骤2: 调用天气API - 完成

步骤3: 返回天气信息 - 完成

[OK] 评估分数: 1.000 (分数越高表示越遵循计划)

【测试 2: 部分遵循计划】

输入: 查询天气信息

计划:

1. 获取用户位置

2. 调用天气API

3. 返回天气信息

实际输出:

步骤1: 获取用户位置 - 完成

步骤3: 返回天气信息 - 完成

[OK] 评估分数: 0.500 (分数越高表示越遵循计划)

【测试 3: 未遵循计划】

输入: 查询天气信息

计划:

1. 获取用户位置

2. 调用天气API

3. 返回天气信息

实际输出:

直接返回了默认天气信息

[OK] 评估分数: 0.000 (分数越高表示越遵循计划)

11-plan_adherence.py

CASE：使用OpenEvals进行自动化测试

TO DO：基于之前的2-langsmith_testing_evaluation.py，使用OpenEvals进行自动化测试

1. 使用的预定义评估器

- ANSWER_RELEVANCE_PROMPT: 相关性评估
- CONCISENESS_PROMPT: 简洁性评估
- RAG_HELPFULNESS_PROMPT: 帮助性评估
- HALLUCINATION_PROMPT: 幻觉检测（避免虚假金融信息）
- TOXICITY_PROMPT: 毒性检测（确保合规）

2. 自定义评估器

- 处理模式评估 (processing_mode)
- 响应完整性评估 (response_completeness)

CASE：使用OpenEvals进行自动化测试

为了支持评估器提取 processing_mode 信息，需要在输出中添加特殊格式：

```
# 从投顾助手获取结果
result = run_wealth_advisor(user_query=user_query,
customer_id=customer_id)
final_response = result.get("final_response", "")
processing_mode = result.get("processing_mode", "unknown")

# 将 processing_mode 信息添加到输出文本中（用于评估器提取）
output_text = f"[处理模式: {processing_mode}]\n\n{final_response}"
return {
   "output": output_text，# 包含处理模式信息的完整输出
   "final_response": final_response，# 原始回答
   "processing_mode": processing_mode，# 处理模式
}
```

关键点:

- 在输出开头添加 [处理模式: xxx] 格式，便于评估器提取
- 保留原始 final_response 和 processing_mode 字段，用于其他用途

CASE：使用OpenEvals进行自动化测试

处理模式评估器 Prompt 设计

```
PROCESSING_MODE_PROMPT = """你是一位评估员，评估投顾助手选择的处理模式是否正确。
<说明>
投顾助手有两种处理模式：
- reactive（反应式）：用于简单查询，需要快速响应
- deliberative（深思熟虑）：用于复杂分析，需要深入思考

<用户查询>
{inputs}
</用户查询>

<实际输出>
输出的开头包含 "[处理模式: xxx]" 格式的信息，请从中提取实际选择的处理模式（应该是 "reactive" 或 "deliberative"）。
{outputs}
</实际输出>

<期望的处理模式>
参考输出中可能包含字典格式，如 {{"processing_mode": "reactive", "should_contain": [...]}}，请提取其中的 "processing_mode" 字段值。
如果参考输出是字符串格式，请直接使用该字符串。
{reference_outputs}
</期望的处理模式>

请评估实际选择的处理模式是否与期望的处理模式一致。
如果一致，给出 1.0 分；如果不一致，给出 0.0 分。"""
```

CASE：使用OpenEvals进行自动化测试

响应完整性评估器 Prompt 设计

```
RESPONSE_COMPLETENESS_PROMPT = """你是一位评估员，评估投顾助手的回答是否完整，是否包含了期望的关键信息。

<用户查询>
{inputs}
</用户查询>

<投顾助手的回答>
注意：回答开头可能包含 "[处理模式: xxx]" 格式的信息，请忽略这部分，只关注实际的回答内容。
{outputs}
</投顾助手的回答>

<期望包含的关键词>
参考输出中可能包含字典格式，如 {{"processing_mode": "reactive", "should_contain": ["关键词1", "关键词2"]}}，请提取其中的 "should_contain" 字段值（这是一个关键词列表）。
如果参考输出是字符串格式，请尝试解析为关键词列表。
{reference_outputs}
</期望包含的关键词>

请评估回答是否包含了期望的关键信息。
根据包含的关键词比例给出 0-1 之间的分数：
- 如果包含了所有期望的关键词，给出 1.0 分
- 如果包含了部分关键词，给出相应的比例分数（例如包含 2/3 的关键词，给出 0.67 分）
- 如果没有包含任何关键词，给出 0.0 分"""
```

CASE：使用OpenEvals进行自动化测试

评估器创建

```
# 创建评估 LLM
eval_llm = ChatTongyi(
    model_name="qwen-turbo",
    dashscope_api_key=os.getenv("DASHSCOPE_API_KEY"),
    temperature=0  # 设置为 0 保证评估结果的一致性
)
# 创建处理模式评估器
processing_mode_evaluator = create_llm_as_judge(
    prompt=PROCESSING_MODE_PROMPT,
    feedback_key="processing_mode",  # 评估结果的键名
    judge=eval_llm,                  # 评估 LLM
    continuous=True,                 # 返回连续分数（0-1）
    use_reasoning=False,             # 不使用推理链
)
```

```
# 创建响应完整性评估器
response_completeness_evaluator = create_llm_as_judge(
    prompt=RESPONSE_COMPLETENESS_PROMPT,
    feedback_key="response_completeness",
    judge=eval_llm,
    continuous=True,
    use_reasoning=False,
)
```

打卡：LangChain Agent+OpenEvals自动化测试

结合你的项目进行自动化测试（可以以投顾AI助手为例）：

Step1，定义测试数据

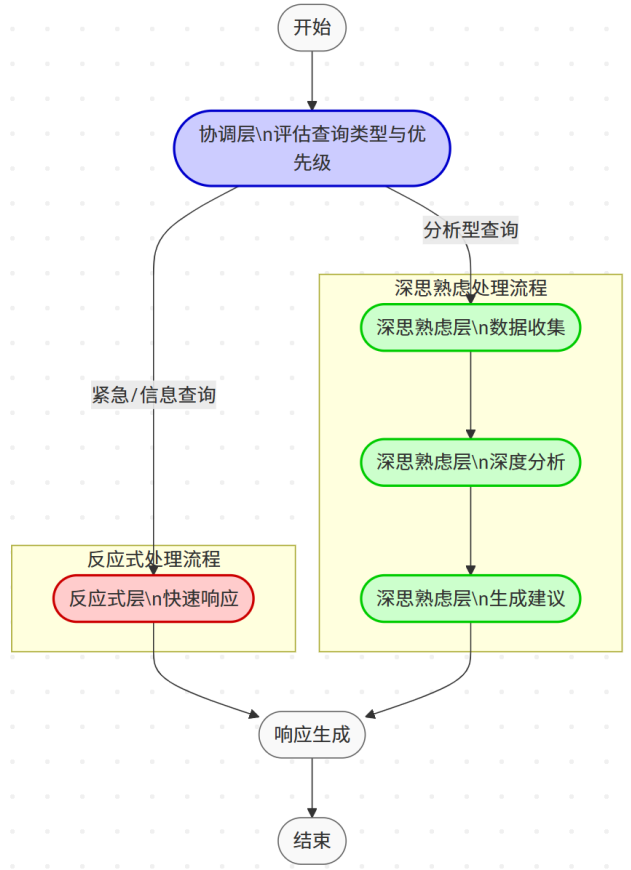
Step2，使用OpenEvals评估器

内置的5种评估（相关性评估、简洁性评估、RAG帮助性评估

幻觉检测、毒性检测）

2种自定义评估：处理模式评估、响应完整性评估

Step3，自动化测试



非LangChain家族的Agent工具（QwenAgent, LangFuse, DeepEval)

DeepEval

什么是DeepEval：

一个开源的 LLM 评估框架，专注于对大语言模型应用进行系统化的质量测试和评估。

它就像传统软件开发中的 Pytest/JUnit，为 LLM 应用提供了标准化的测试能力。

DeepEval （质检员）

类似软件开发里的单元测试

负责在上线前给模型打分，确保质量不下降

LangSmith（监控室）

像显微镜或黑匣子

负责在线上追踪 Agent 思考过程、定位错误、分析成本

DeepEval

DeepEval的特点：

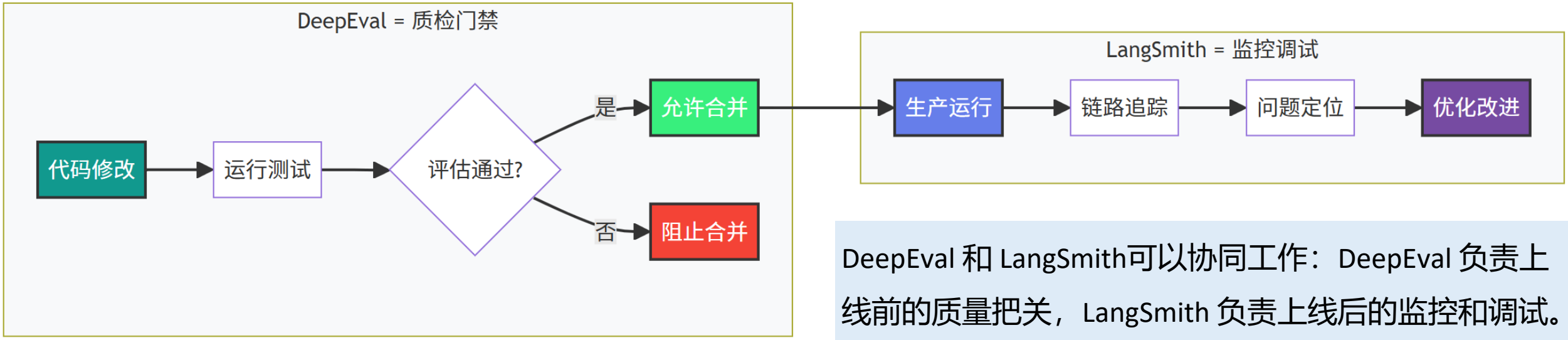
- 内置40+评估指标

提供丰富的评估指标，覆盖 RAG、 Agent、对话等多种场景，包括幻觉检测、相关性、忠实度等。

Hallucination 幻觉检测	Faithfulness 忠实度	Answer Relevancy 答案相关性	Contextual Relevancy 上下文相关性
Toxicity 有害内容检测	Bias 偏见检测	Summarization 摘要质量	G-Eval 通用评估

LangSmith 与 DeepEval

工具	传统软件类比	核心作用	场景关键词
DeepEval	Pytest / JUnit	跑分测试	离线、CI/CD流水线、回归测试、指标打分
LangSmith	Datadog / Sentry	调试观测	在线、链路追踪、Debug、成本分析、Prompt管理



DeepEval 和 LangSmith可以协同工作：DeepEval 负责上线前的质量把关，LangSmith 负责上线后的监控和调试。

OpenEvals与DeepEval

Thinking： OpenEvals与DeepEval是什么关系？

- DeepEval 与 OpenEvals 的作用高度重叠，同类竞品，都是开源的 LLM 输出评估器库
- 指标丰富度：

DeepEval 胜出，它把 RAGAS、Helm、MT-bench 等论文里的指标都实现了一遍；

OpenEvals 目前以“忠实度、相关性、工具正确性”等核心指标为主。

- 自定义体验：

DeepEval 提供 G-Eval 语法糖，写起来最短；

OpenEvals 也支持自定义，但需要继承 Evaluator 基类，稍微多几行模板代码。

OpenEvals与DeepEval

- 运行位置：

两者都支持本地运行（LLM-as-a-judge 或 NLP 小模型）

DeepEval 默认完全离线；

OpenEvals 本地/云端皆可，取决于你在 LangSmith 里是否打开上传开关。

LangChain / LangSmith 用户：OpenEvals 是亲儿子，一条命令就能在实验里调用；

DeepEval 也能接入，但要在代码里手动把 trace 喂给 DeepEval 的 testcase，再把分数写回 LangSmith，多两步封装。

Qwen Agent与LangChain Agent

1-hybrid_wealth_advisor_langgraph_langsmith.py这里我打算用 qwen-agent来实现，不用 langchain/langgraph，另外也不用 langsmith了，可以参考 @assistant_ticket_bot-3.py 写入到新的 .py

使用 gui的形式

更新工具定义，确保正确处理参数：

1-hybrid_wealth_advisor_qwen_agent.py +8

```
        # 解析参数（即使没有参数也需要处理）
        try:
            if params:
                import json
                args = json.loads(params)
        except:
```

检查响应处理逻辑，确保正确提取响应：

Read assistant_ticket_bot-3.py L224-233

优化响应提取逻辑，确保正确获取最终响应：

1-hybrid_wealth_advisor_qwen_agent.py +8 -3

```
        for response in agent.run(messages):
            response_messages.extend(response)
            response_messages = response  # 每次迭代都会更新为最新的响应消息列表

8 hidden lines
                    for item in content:
```

Qwen Agent与LangChain Agent

```
def init_wealth_advisor_agent(customer_profile: Dict[str, Any]) -> Assistant:
    llm_cfg = {
        'model': 'qwen-turbo-latest',
        'timeout': 30,
        'retry_count': 3,
    }
    try:
        agent = Assistant(
            llm=llm_cfg,
            name='财富管理投顾AI助手',
            description='混合智能体财富管理咨询服务',
            system_message=get_system_prompt(customer_profile),
            function_list=['query_shanghai_index'],
        )
        print("财富顾问智能体初始化成功！")
        return agent
    except Exception as e:
        print(f"智能体初始化失败: {str(e)}")
        raise
```

Qwen Agent与LangChain Agent

今天上证指数的表现如何?

user

财富管理投顾AI助手

▶ Start calling tool "query_shanghai_index" ...

▼Finished tool calling.

上证指数 当前点位: 3125.62，涨跌: 6.32，涨跌幅: 0.20%

今天上证指数表现稳健，当前点位为3125.62点，上涨了6.32点，涨幅为0.20%。市场整体呈现小幅回暖态势，适合保持关注并适时调整投资策略。

Qwen Agent与LangChain Agent

Thinking：Qwen-Agent与LangChain/LangGraph在Agent搭建上的区别?

- 架构理念

LangChain/LangGraph：显式状态管理 + 图式工作流，需要明确定义每个节点和状态流转

qwen-agent：隐式状态管理 + 声明式设计，通过 system_message 描述行为，框架自动处理

- 代码量

LangChain/LangGraph：703 行（需要定义状态、节点、边等）

qwen-agent：375 行（减少约 47%）

Qwen Agent与LangChain Agent

- 关键特性对比

特性	LangChain	qwen-agent
学习曲线	陡峭（2-3周）	平缓（3-5天）
可控性	高（精确控制）	中（框架控制）
可视化调试	✅ LangSmith	❌
Web UI	❌ 需自建	✅ 内置
工具注册	手动管理	✅ 装饰器
中文支持	一般	✅ 优化

Qwen Agent与LangChain Agent

- 适用场景

LangChain/LangGraph 适合：

- 复杂业务流程（多阶段、多分支）
- 企业级应用（需要追踪和监控）
- 需要与 LangChain 生态集成

qwen-agent 适合：

- 快速原型开发
- 中文场景应用
- 简单对话式应用
- 需要快速交付的项目

LangFuse使用

什么是LangFuse?

LangFuse 是一个开源的 LLM（大语言模型）工程平台，

定位是“可观测性 + 调试 + 评估”三合一的 LLMOps 工具

https://github.com/langfuse/langfuse

LangFuse与LangSmith：

LangFuse：全开源，专注“可观测 + 提示管理 + 轻量评估”，任何框架/模型都能接入 。

LangSmith：LangChain 官方商业产品，主打“企业级测试-评估-监控”闭环，深度耦合 LangChain 生态。

LangFuse使用

TO DO：在qwen-agent基础上增加LangFuse

Step1，注册LangFuse

Step2，需要配置环境变量：

LANGFUSE_SECRET_KEY = "sk-XX"

LANGFUSE_PUBLIC_KEY = "pk-XX"

LANGFUSE_BASE_URL = "https://us.cloud.langfuse.com"

Step3，集成LangFuse

LangFuse使用

我的投资组合中科技股占比是多少?

user

这是一个简单的账户信息查询，我将使用反应式处理模式来回答您的问题。

根据您提供的投资组合信息，您的资产配置如下：

- 股票：40%
- 债券：30%
- 现金：10%
- 另类投资：20%

您提到对科技行业有投资偏好，但当前没有具体说明股票部分中科技股的具体占比。如果您希望了解科技股在您整体投资组合中的实际比例，建议您提供更详细的信息，例如：

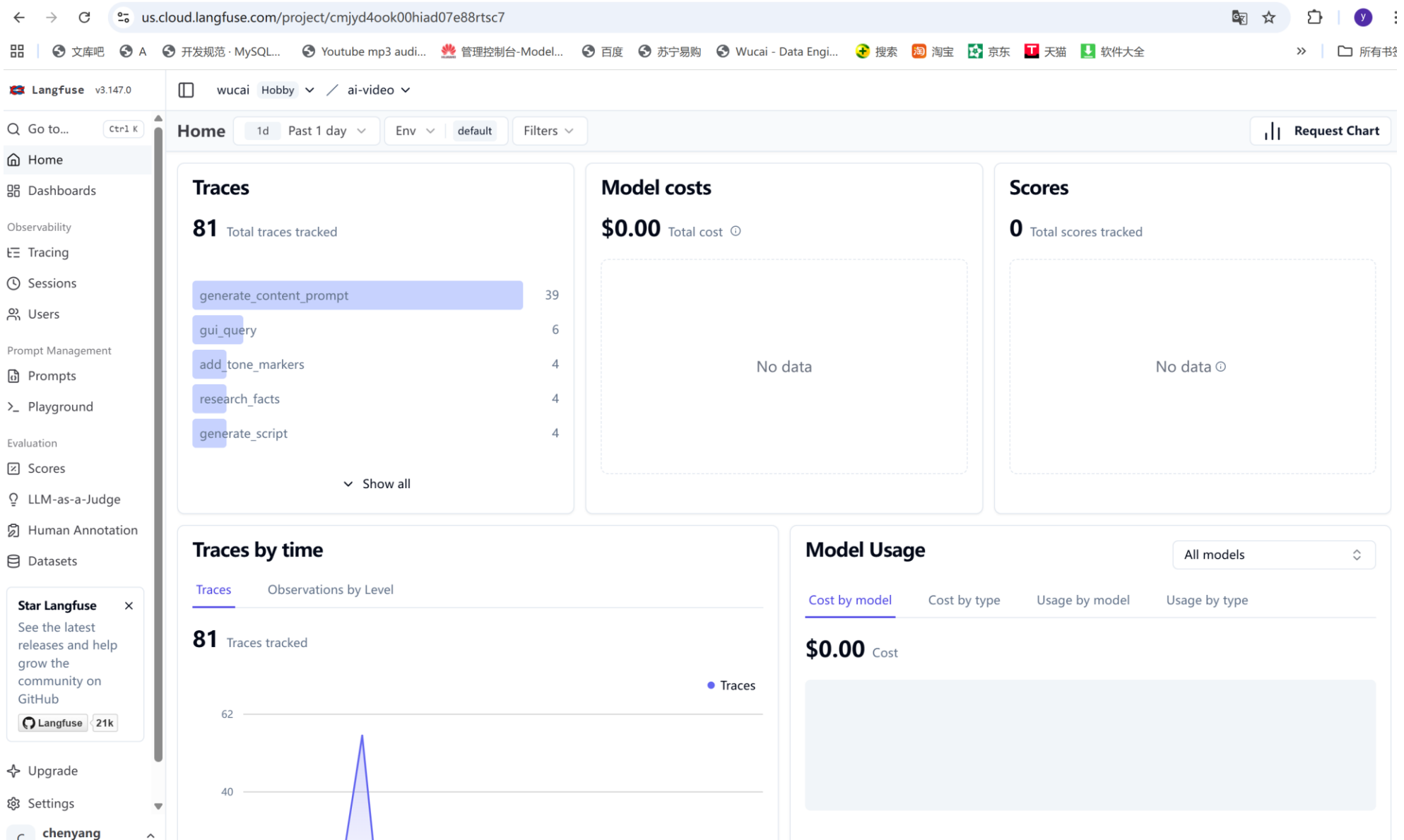
1. 您目前持有的股票明细（包括每只股票的市值占比）
2. 或者告知我您计划增加科技股的投资比例

这样我可以帮您计算出精确的科技股占比，并给出相应的建议。

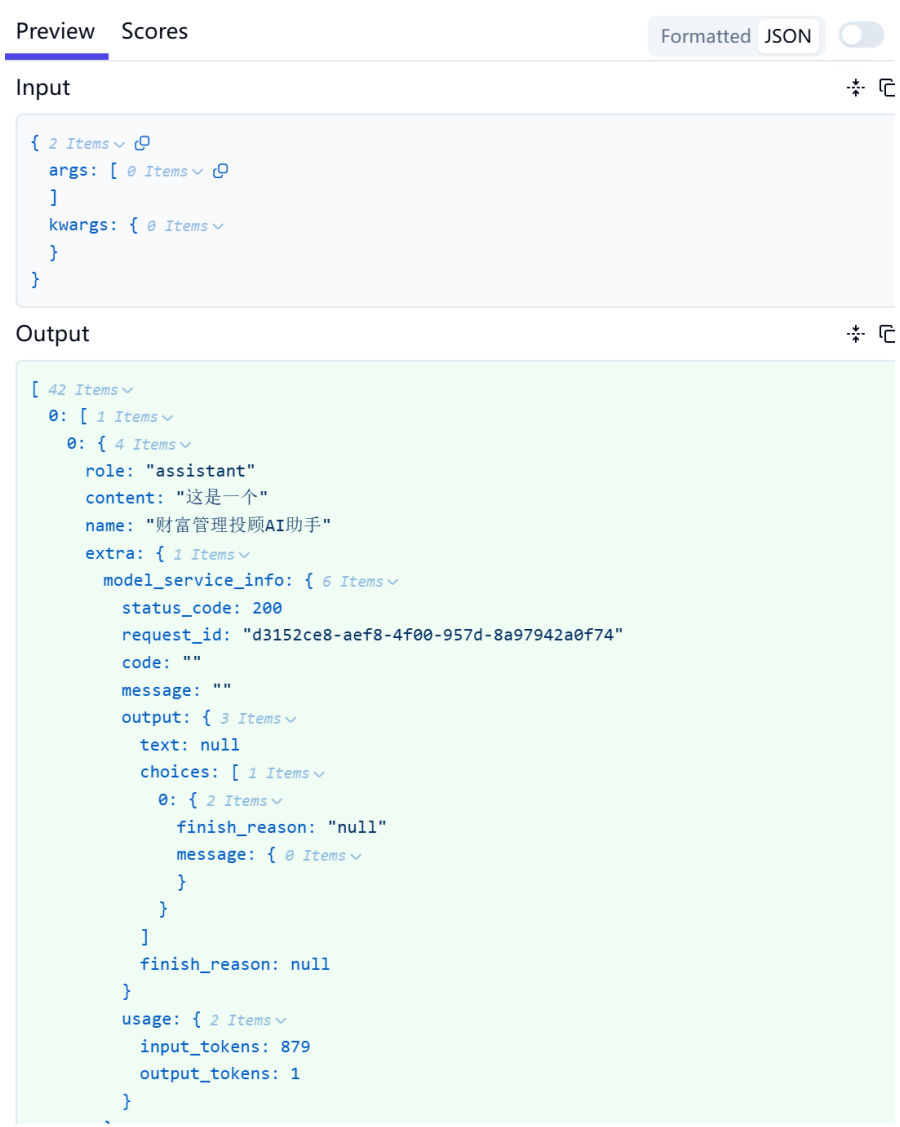
财富管理投顾AI助手

```
您提到对科技行业有投资偏好，但当前没有具体说明股票部分中科技股的具体占比。如果您希望了解科技股在您整体投资组合中的实际比
[DEBUG] 消息 #0: <class 'dict'>, role=assistant
[DEBUG] 助手消息内容类型: <class 'str'>
[DEBUG] 提取的最终输出长度: 254
[DEBUG] 最终输出预览: 这是一个简单的账户信息查询，我将使用反应式处理模式来回答您的问题。

根据您提供的投资组合信息，您的资产配置如下:
- 股票: 40%
- 债券: 30%
- 现金: 10%
- 另类投资: 20%

您提到对科技行业有投资偏好，但当前没有具体说明股票部分中科技股的具体占比。如果您希望了解科技股在您整体投资组合中的实际比
例，建议您提供更详细的信息，例如:

1. 您目前持有的股票明细（包括每只股票的市值占
[DEBUG] 准备设置 output: 这是一个简单的账户信息查询，我将使用反应式处理模式来回答您的问题。

根据您提供的投资组合信息，您的资产配置如下:
- 股票: 40%
- 债券: 30%
- 现金: 10%
- 另类投资: 20%

您提到
[DEBUG] output 设置成功
[DEBUG] LangFuse flush() 完成
[DEBUG] _traced_run() 完成, input_set=False, output_set=True
[DEBUG] _traced_run() 返回, 响应数量: 45
```

LangFuse使用


us.cloud.langfuse.com/project/cmjyd4ook00hiad07e88rtsc7
Langfuse v3.147.0
wucai Hobby
ai-video
Go to... Ctrl K
Home
Dashboards
Observability
Tracing
Sessions
Users
Prompt Management
Prompts
Playground
Evaluation
Scores
LLM-as-a-Judge
Human Annotation
Datasets
Star Langfuse
See the latest releases and help grow the community on GitHub
Langfuse 21k
Upgrade
Settings
chenyang
Home
1d Past 1 day
Env default
Filters
Request Chart
Traces
81 Total traces tracked
generate_content_prompt 39
gui_query 6
add_tone_markers 4
research_facts 4
generate_script 4
Show all
Model costs
$0.00 Total cost
No data
Scores
0 Total scores tracked
No data
Traces by time
Traces
Observations by Level
81 Traces tracked
Traces
62
40
Model Usage
All models
Cost by model
Cost by type
Usage by model
Usage by type
$0.00 Cost

LangFuse使用


Preview
Scores
Formatted
JSON
Input
{ 2 Items
args: [0 Items
]
kwargs: { 0 Items
}
}
Output
[42 Items
0: [1 Items
0: { 4 Items
role: "assistant"
content: "这是一个"
name: "财富管理投顾AI助手"
extra: { 1 Items
model_service_info: { 6 Items
status_code: 200
request_id: "d3152ce8-aef8-4f00-957d-8a97942a0f74"
code: ""
message: ""
output: { 3 Items
text: null
choices: [1 Items
0: { 2 Items
finish_reason: "null"
message: { 0 Items
}
}
]
finish_reason: null
}
usage: { 2 Items
input_tokens: 879
output_tokens: 1
}


Preview
Log View
Formatted
JSON
}
}
]
41: [1 Items
0: { 4 Items
role: "assistant"
content: "这是一个简单的账户信息查询，我将使用反应式处理模式来快速回答您的问题。
根据您提供的投资组合信息，您的当前配置如下：
- 股票：40%
- 债券：30%
- 现金：10%
- 另类投资：20%
您提到偏好科技行业，但目前没有提供股票内部的具体行业分布。因此，无法确定科技股在总资产中的具体占比。
建议您：
1. 查看您股票持仓的详细分类（如科技、金融、医疗等）
2. 如果您希望提高科技股比例，可以考虑调整股票部分的配置
如果您需要进一步分析或优化投资组合，请随时告诉我！"
name: "财富管理投顾AI助手"
extra: { 1 Items
model_service_info: { 6 Items
status_code: 200
request_id: "d3152ce8-aef8-4f00-957d-8a97942a0f74"
code: ""
message: ""
output: { 3 Items
text: null
choices: [1 Items
0: { 2 Items
finish_reason: "stop"
message: { 0 Items
}
}
]
finish_reason: null


Thank You
Using data to solve problems