

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分



Orchestration and monitoring

Frameworks 2 · October 20, 2026

Carnegie Mellon University
Language Technologies Institute

Daniel Fried
11-768: AI Agents

1

编排与监控

框架（第二部分）· 2026年10月20日

卡内基梅隆大学·语言技术研究所

Daniel Fried

11-768: AI 智能体

编排与监控

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Structure around agent actions

We have already seen several ways to guide agent actions.

Skills	Instructions guide behavior; executable skills can constrain execution.
Planning and decomposition	Break a task into parts and coordinate their execution, as in MACU, RAO, and RLM.
Automations	Connect agent runs through prescribed steps and checks.

Today: Orchestration frameworks let developers specify parts of a workflow in advance, while delegating judgments and actions to LLMs.

[Lecture 4 · Skills](#) [Lecture 5 · MACU](#) [Lecture 14 · Automations](#)

2

围绕智能体行动构建结构

我们已经看过几种引导智能体（Agent）行动的方式。

技能：指令引导行为；可执行技能可以约束执行。

规划与分解：把任务拆成若干部分并协调执行，例如 MACU、RAO 和 RLM。

今天：编排框架让开发者预先指定工作流的部分环节，同时把判断和行动委托给大语言模型（LLM）。

今天：编排框架让开发者预先指定工作流的部分环节，同时把判断和行动委托给大语言模型（LLM）。

第4讲·技能；第5讲·MACU；第14讲·自动化

围绕智能体行动构建结构

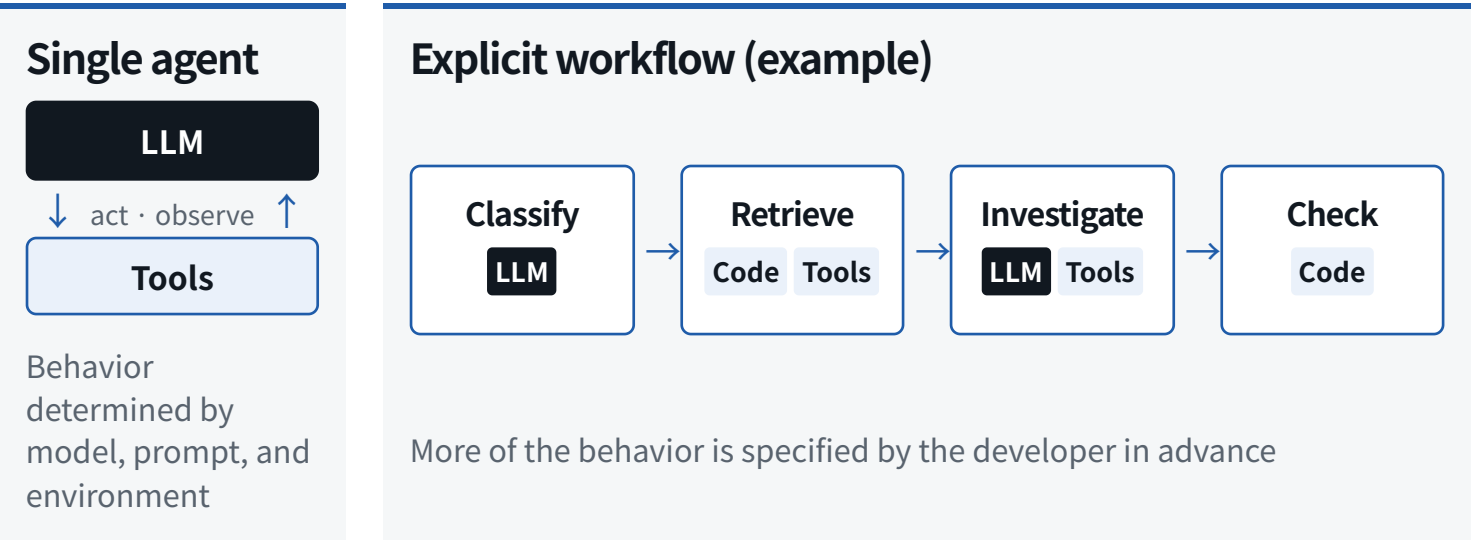
第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Single agents vs. workflows

Both combine LLMs, tools, and application code.



What do we gain by fixing a workflow?

单智能体与 workflow

两者都由大语言模型（LLM）、工具和应用程序代码组成。

单智能体：LLM 与工具之间循环进行行动和观测。行为由模型、提示和环境决定。

显式 workflow（示例）：分类（LLM）→ 检索（代码、工具）→ 调查（LLM、工具）→ 检查（代码）。更多行为由开发者预先规定。

固定 workflow 能带来什么？

第14讲·单智能体基线；LangGraph·workflow与智能体

单智能体与 workflow

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Workflow benefits

Fixing a workflow for a known procedure reduces the model decisions

Consistency

Prescribe required steps and enforce checks in code.

Flagged reply → approval → send

Efficiency

Use an appropriate model for each step, or even just code (no model at all)

Poll: code
Classify: smaller model

工作流的益处

为已知流程固定工作流，可以减少模型需要作出的决策。

一致性：规定必需步骤，并在代码中强制执行检查。被标记的回复 → 审批 → 发送。

效率：为每个步骤使用合适的模型，甚至只用代码（完全不用模型）。轮询：用代码；分类：用较小的模型。

Anthropic·Building effective agents; LangGraph·Thinking in LangGraph

工作流的益处

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

工作流编排



工作流编排

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Examples

Customer support requests follow different procedures.

Routine question

“How do I reset my password?”

Retrieve guidance → **draft reply** → **send**

Request requiring review

“I was charged twice. Please refund the extra payment.”

Draft reply → employee approval → **send**

示例

示例

客户支持请求会遵循不同的处理流程。

常规问题：“我该如何重置密码？” 检索指南 → 起草回复 → 发送。

需要审核的请求：“我被扣了两次款。请退还多扣的款项。” 起草回复 → 员工审批 → 发送。

LangGraph·Thinking in LangGraph·经改编的客服场景

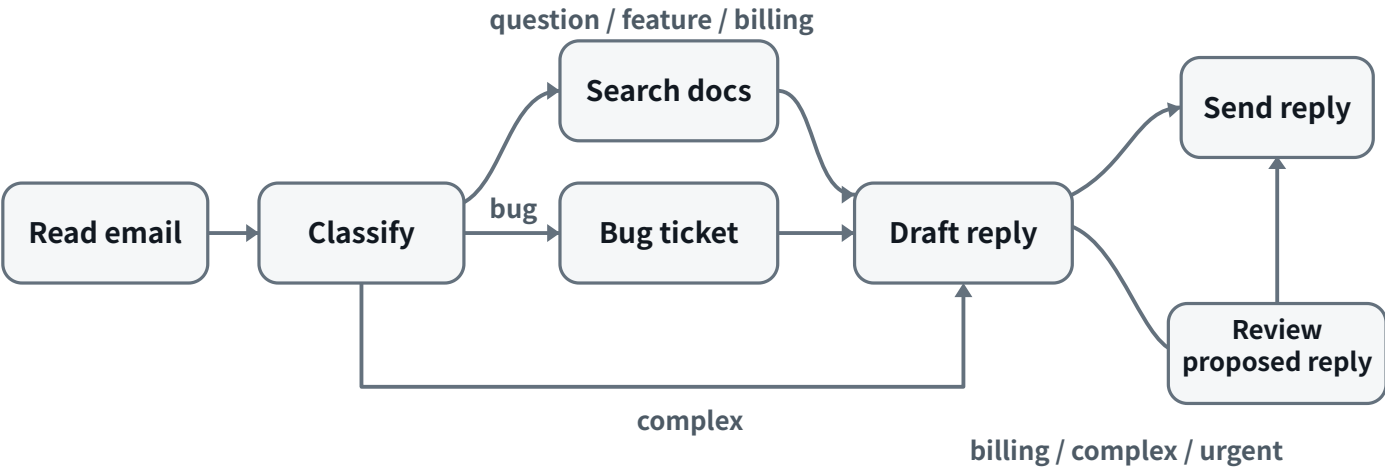
第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Workflow steps

Explicit steps encode the required work and its possible continuations.



LangGraph connects functions implementing these steps.

LangGraph · Thinking in LangGraph

7

Workflow steps

显式步骤编码了必需的工作及可能的后续路径。

流程：读取邮件 → 分类。对于问题 / 功能请求 / 账单类，搜索文档；对于 bug，创建 bug 工单；对于复杂请求，则直接进入起草回复。之后可能发送回复，也可能审核拟发送的回复。账单 / 复杂 / 紧急请求进入审核路径。

LangGraph 将实现这些步骤的函数连接起来。

LangGraph·Thinking in LangGraph

Workflow steps

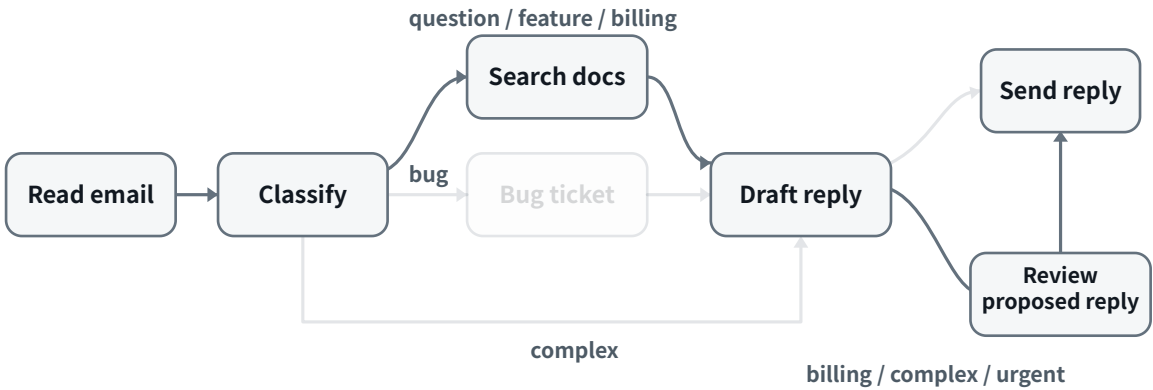
第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Conditional routing

The request and intermediate results determine which steps execute.



- Routine question: search → draft → send
- Billing-classified request: search → draft → review → send

The review path depends on the model’s classification.

LangGraph · Thinking in LangGraph

条件路由

请求内容和中间结果决定执行哪些步骤。

流程中的分类标签：问题 / 功能请求 / 账单；bug；复杂；账单 / 复杂 / 紧急。节点包括：读取邮件、分类、搜索文档、创建 bug 工单、起草回复、审核拟发送的回复、发送回复。

常规问题：搜索 → 起草 → 发送。

被分类为账单类的请求：搜索 → 起草 → 审核 → 发送。

是否进入审核路径取决于模型的分类结果。

LangGraph·Thinking in LangGraph

条件路由

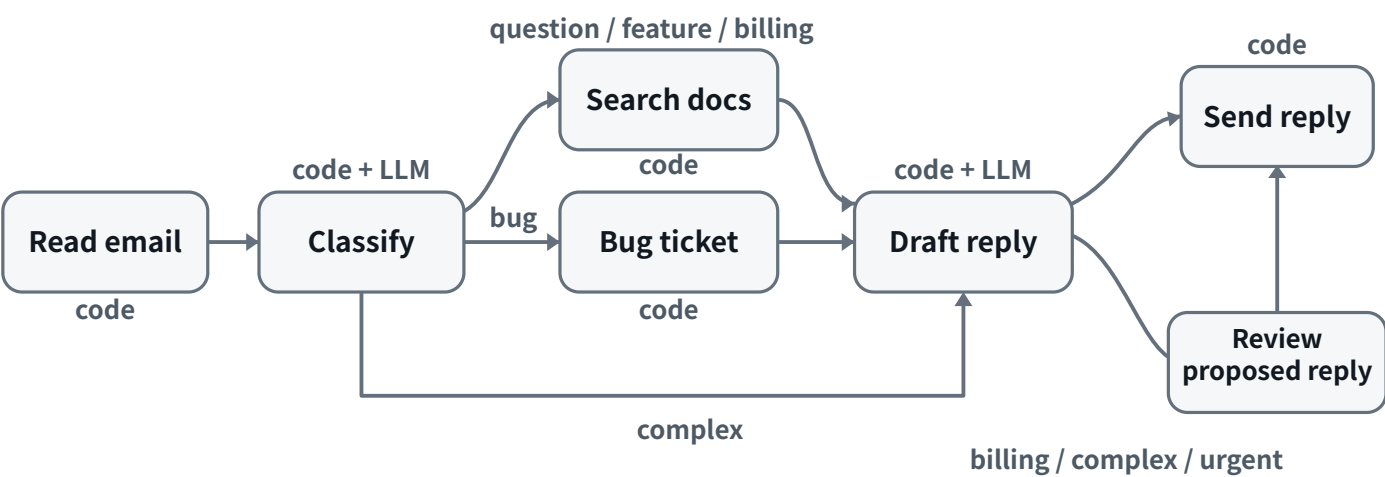
第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Inside workflow steps

Each step can combine code, model calls, and tools.



Here, classification and drafting call an LLM from code.

LangGraph · Thinking in LangGraph

9

workflow 步骤内部

每个步骤都可以组合代码、模型调用和工具。

流程节点的实现方式：读取邮件（代码）；分类（代码 + LLM）；搜索文档（代码）；创建 bug 工单（代码）；起草回复（代码 + LLM）；审核拟发送的回复；发送回复（代码）。条件标签为问题 / 功能请求 / 账单、bug、复杂，以及账单 / 复杂 / 紧急。

在此示例中，分类和起草会在代码中调用 LLM。

LangGraph·Thinking in LangGraph

workflow 步骤内部

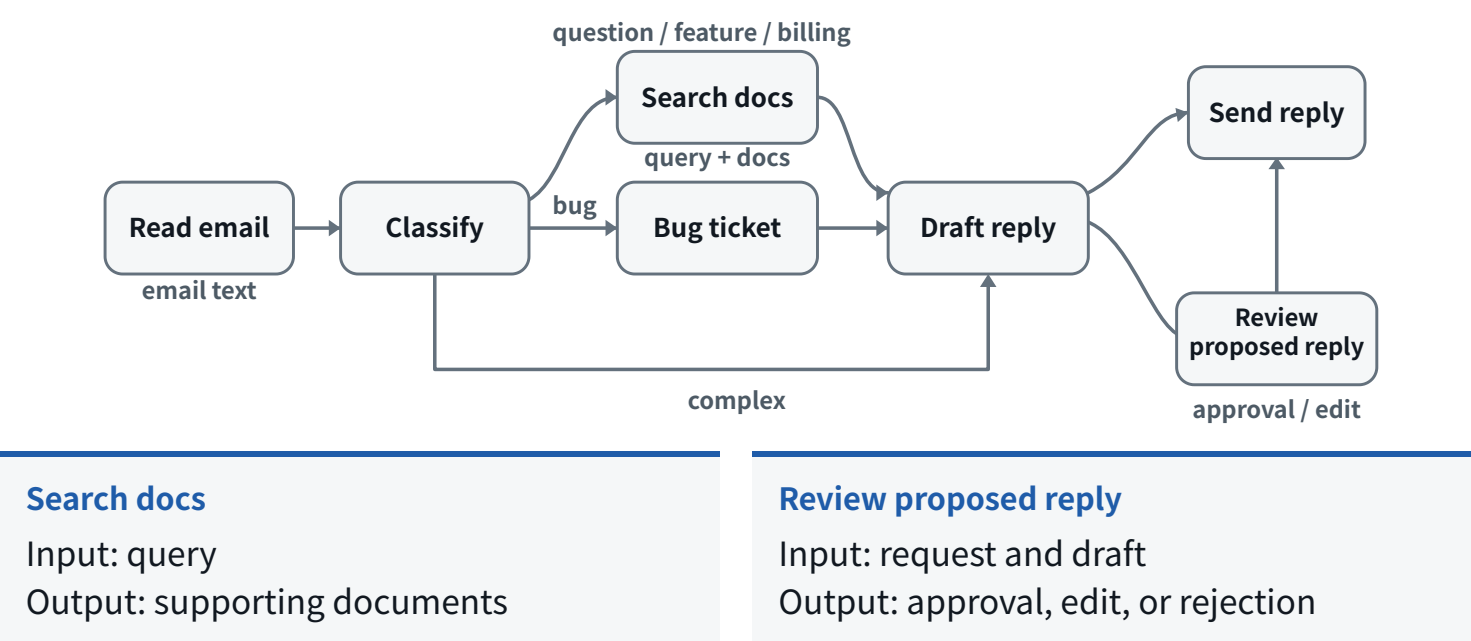
第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Data and human input

Search and review add information



LangGraph · Thinking in LangGraph

10

数据与人工输入

搜索和审核会补充信息。

workflow中的节点依次包括读取邮件、分类、搜索文档、创建 bug 工单、起草回复、审核拟发送的回复和发送回复。读取邮件节点接收邮件文本；搜索文档节点接收查询并返回文档；审核节点接收审批或编辑意见。

搜索文档：输入为查询；输出为支持性文档。

审核拟发送的回复：输入为请求和草稿；输出为批准、编辑或拒绝。

LangGraph·Thinking in LangGraph

数据与人工输入

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Shared state

Passed between nodes in the workflow.

```
class SupportState(TypedDict, total=False):
    email_id: str
    email_content: str
    classification: dict
    search_results: list[dict]
    draft_response: str
    review_decision: dict
```

Billing request

Email: “Urgent: Acme Cloud charged our company twice...”
Classification: billing, high urgency
Search result: billing-review policy
Draft: proposed reply for review

共享状态

共享状态

在工作流的节点之间传递。

代码示例（字段名与类型原样保留）：

```
```python
class SupportState(TypedDict, total=False):
 email_id: str
 email_content: str
 classification: dict
 search_results: list[dict]
 draft_response: str
 review_decision: dict
```
```

账单请求示例状态：邮件：“紧急：Acme Cloud 向我们公司重复扣款了……”；
分类：账单，高紧急度；搜索结果：账单审核政策；草稿：待审核的拟议回复。

LangGraph·Thinking in LangGraph·步骤3

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

State updates

A node reads state and returns the fields it changes.

```
def classify_intent(state):
    classification = model.classify(
        state["email_content"]
    ).model_dump()
    return {"classification": classification}
```

Support state

email_content

“Urgent: Acme Cloud charged our company twice...”

classification

billing · high urgency

状态更新

一个节点读取状态，并返回它所更改的字段。

代码示例（代码与字段名原样保留）：

```
```python
def classify_intent(state):
 classification = model.classify(
 state["email_content"]
).model_dump()
 return {"classification": classification}
```
```

支持状态示例：`email\_content`：“紧急：Acme Cloud 向我们公司重复扣款了.....”；`classification`：账单 · 高紧急度。

LangGraph·Thinking in LangGraph·步骤3

状态更新

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Node implementations

Nodes read state, do work, and return updates.

Document retrieval

Search service

```
def search_documentation(state):
    c = state["classification"]
    query = f"{c['intent']} {c['topic']}"
    return {"search_results": services.search(query)}
```

Reply generation

llm: chat model

```
def draft_response(state):
    response = llm.invoke([
        ("system", "Draft a reply using these documents."),
        ("human", json.dumps({
            "email": state["email_content"],
            "documents": state.get("search_results", []),
        })),
    ])
    return {"draft_response": response.text}
```

节点实现

节点实现

节点读取状态、执行工作并返回更新。

文档检索（搜索服务）代码示例，代码原样保留：

```
python
def search_documentation(state):
    c = state["classification"]
    query = f"{c['intent']} {c['topic']}"
    return {"search_results": services.search(query)}
...
```

回复生成（LLM：聊天模型）代码示例，代码及其中的提示词原样保留：

```
python
def draft_response(state):
    response = llm.invoke([
        ("system", "Draft a reply using these documents."),
        ("human", json.dumps({
            "email": state["email_content"],
            "documents": state.get("search_results", []),
        })),
    ])
    return {"draft_response": response.text}
...
```

LangGraph·Thinking in LangGraph·步骤4

第 15 讲 · 编排与监控

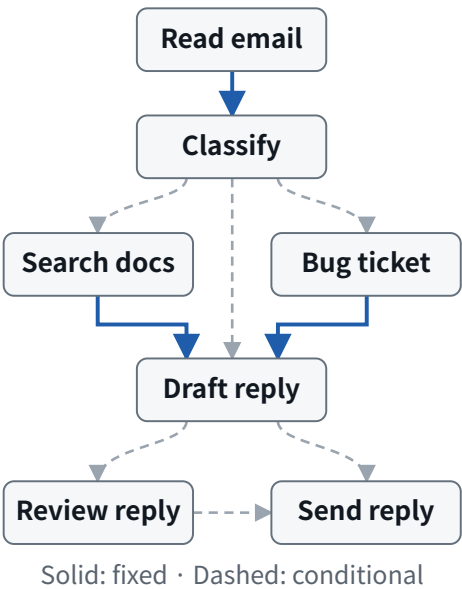
EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Graph assembly

Node functions and transitions form the support workflow.

```
g = StateGraph(SupportState)
g.add_node("read_email", read_email)
g.add_node("classify_intent", classify_intent)
g.add_node("search_documentation",
            search_documentation)
g.add_node("draft_response", draft_response)
g.add_edge(START, "read_email")
g.add_edge("read_email", "classify_intent")
g.add_edge(
    "search_documentation", "draft_response"
)
graph = g.compile(checkpointer=checkpointer)
```



组装图

节点函数和转换关系构成客服工作流。

代码示例（函数名、API 和字段原样保留）：

```
python
g = StateGraph(SupportState)
g.add_node("read_email", read_email)
g.add_node("classify_intent", classify_intent)
g.add_node("search_documentation",
            search_documentation)
g.add_node("draft_response", draft_response)
g.add_edge(START, "read_email")
g.add_edge("read_email", "classify_intent")
g.add_edge(
    "search_documentation", "draft_response"
)
graph = g.compile(checkpointer=checkpointer)
```

组装图

图中节点：读取邮件 → 分类；随后可搜索文档或创建 bug 工单，再起草回复；最后审核回复或发送回复。实线表示固定路径，虚线表示条件路径。

LangGraph·Thinking in LangGraph·步骤5

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

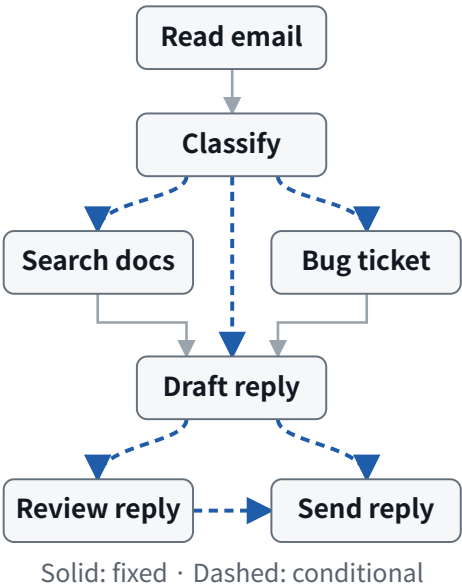
ZH / 中文译文 · 第 1/1 部分

Classification and routing

The classification selects a path through the assembled graph.

```
intent = classification["intent"]
destination = (
    "bug_tracking" if intent == "bug" else
    "search_documentation"
    if intent in {"question", "feature", "billing"}
    else "draft_response"
)
return Command(
    update={"classification": classification},
    goto=destination,
)
```

[Browse the full source ↗](#)
LangGraph · Thinking in LangGraph · Step 4



分类与路由

分类结果会选择组装图中的一条路径。

代码示例（保留原始代码与 API）：

```
python
intent = classification["intent"]
destination = (
    "bug_tracking" if intent == "bug" else
    "search_documentation"
    if intent in {"question", "feature", "billing"}
    else "draft_response"
)
return Command(
    update={"classification": classification},
    goto=destination,
)
...

```

图中的实线表示固定路径；虚线表示条件路径。

[浏览完整源码 ↗](#)

LangGraph·Thinking in LangGraph·步骤4

分类与路由

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Recoverable progress

Saving search results before generation preserves completed work.



With saved search results, restart generation using completed work.

可恢复的进度

可恢复的进度

在生成前保存搜索结果，可以保留已经完成的工作。

流程：搜索文档 → 已保存的搜索结果 → 生成回复。如果生成期间崩溃，可以重复生成。

有了已保存的搜索结果，就可以基于已完成的工作重新开始生成。

LangGraph·Thinking in LangGraph·节点粒度

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Checkpointing and resuming

Saved state and execution progress let a workflow continue later.

| | |
|-----------------|--|
| Save progress | A persistent checkpointer records the workflow’s state and where execution should continue. |
| Resume | Use the same execution ID to load saved progress and continue. Supply new input when needed. |
| Retry a failure | A retry policy repeats a failed node after a temporary error, such as a timeout. |
| Human input | An interrupt pauses the workflow. The application collects a response and supplies it on resume. |

[LangGraph · Persistence](#)

[LangGraph · Interrupts](#)

[LangGraph · Retry policies](#)

17

检查点与恢复执行

保存的状态和执行进度让工作流之后能够继续。

保存进度：持久化检查点记录器会记录工作流状态以及执行应从何处继续。

恢复执行：使用同一个执行 ID 加载已保存的进度并继续；必要时可以提供新输入。

重试失败：重试策略会在临时错误（例如超时）后重新执行失败的节点。

人工输入：中断会暂停工作流。应用收集响应，并在恢复执行时将其提供给工作流。

LangGraph·持久化；LangGraph·中断；LangGraph·重试策略

检查点与恢复执行

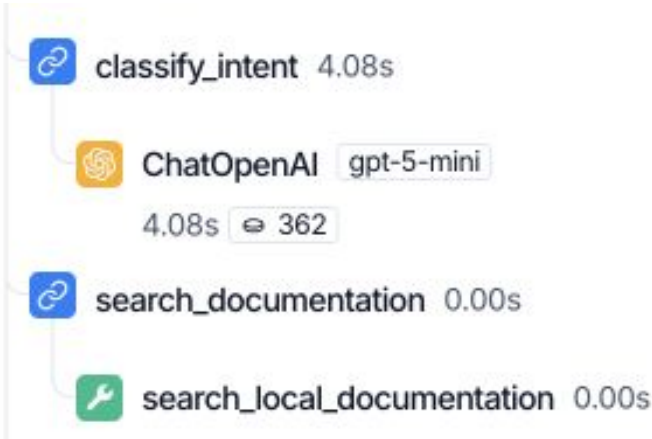
第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Monitoring with execution traces

A record of the execution shows the work done inside each step.



- Steps contain model calls, tool calls, and ordinary code
- Each operation can record inputs, outputs, time, and errors
- LangSmith calls an operation a **run** and the whole tree a **trace**

Course support demo · [Open the recorded trace ↗](#)

18

通过执行轨迹进行监控

执行记录会显示每个步骤内部完成的工作。

步骤包含模型调用、工具调用和普通代码。每项操作都可以记录输入、输出、耗时和错误。LangSmith 将单项操作称为一次 run，将整棵树称为一条 trace。

示例轨迹中可见：`classify\_intent` 用时 4.08 秒；`ChatOpenAI` 使用 `gpt-5-mini`，用时 4.08 秒、362 个词元；`search\_documentation` 用时 0.00 秒；`search\_local\_documentation` 用时 0.00 秒。

课程客服演示·打开已记录的轨迹 ↗

通过执行轨迹进行监控

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Monitoring with execution traces

A record of the execution shows the work done inside each step.

Routine password question

“Hi, I forgot my Acme Cloud password. How can I reset it?”

[Workflow trace](#)

Duplicate charge

“Urgent: Acme Cloud charged our company twice for this month's subscription. ... Please investigate the duplicate charges and refund the extra payment.”

[Workflow trace pt 1](#)
[Workflow trace pt 2](#)

通过执行轨迹进行监控

执行记录会显示每个步骤内部完成的工作。

常规密码问题：“嗨，我忘记了 Acme Cloud 密码。怎样重置？” workflow 轨迹。

重复扣款：“紧急：Acme Cloud 对我们本月的订阅向公司重复收费了.....请调查重复扣款，并退还多扣的款项。” workflow 轨迹第1部分； workflow 轨迹第2部分。

LangSmith·课程录制的轨迹

通过执行轨迹进行监控

Unit tests for nodes

Explicit inputs and outputs let us test each step in isolation.

Classification · example test

Given

Email: “I was charged twice.”
Fixed model output: billing

Classify intent

Expect returned values

Classification update: billing
Routing destination: document search

Repeatable checks

Control inputs and external calls. Check returned updates and routing decisions.

Scope of the test

Direct node calls bypass checkpoints. Test persistence and actual routing through the graph.

节点的单元测试

明确的输入和输出让我们可以单独测试每个步骤。

分类·测试示例：给定一封邮件：“我被扣了两次款。” 固定模型输出：`billing`。调用“分类意图”节点。预期返回值：分类更新为`billing`；路由目标为文档搜索。

可重复的检查：控制输入和外部调用，检查返回的更新以及路由决策。

测试范围：直接调用节点会绕过检查点。要测试持久化和图中的实际路由，需要通过图运行测试。

LangGraph·测试单个节点·客服演示示例

节点的单元测试

Workflow and regression tests

Connected steps need controlled tests and checks with real models and services.

| Controlled workflow test | Expected behavior |
|--------------------------|--------------------------------------|
| Password question | Retrieve guidance, draft, send |
| Billing complaint | Pause for approval; nothing sent yet |
| Rejected reply | Finish without sending |
| Temporary search failure | Retry search, then continue |

- Regression checks

Do tests still pass after a change, e.g. swapping in a new model?
- Integration testing

Run the end-to-end workflow to test how nodes interact

Workflow测试与回归测试

相互连接的步骤需要在真实模型和服务上进行受控测试与检查。

受控 workflow 测试及预期行为：密码问题——检索指南、起草、发送；账单投诉——暂停并等待审批，暂不发送任何内容；回复被拒绝——结束且不发送；搜索暂时失败——重试搜索，然后继续。

回归检查：例如换用新模型之后，测试是否仍然通过？

集成测试：运行端到端 workflow，以测试各节点如何交互。

LangGraph·图测试；LangChain·集成测试；LangSmith·评估

Workflow测试与回归测试

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Workflow tradeoffs

Pros

- Consistency
- Possible efficiency gains
- Testability

Cons

- Less adaptable
- Information flow has to be explicitly managed

22

工作流的权衡

优点：一致性；可能提升效率；可测试性。

缺点：适应性较差；信息流必须显式管理。

工作流的权衡

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Agent / Workflow frameworks

Frameworks package related orchestration ideas around different programming abstractions.

| Framework | Programming emphasis |
|-----------------------------------|---|
| OpenHands | Configurable agent execution and work integrations · previous lecture |
| LangGraph | Explicit workflow graphs, shared state, and recovery · today's example |
| OpenAI Agents SDK | Agent loops, tools, delegation, and handoffs |
| AG2 | Agent conversations and coordination, including group workflows |
| DSPy | Modular LLM programs and prompt optimization using examples and metrics |

23

智能体 / 工作流框架

框架围绕不同的编程抽象，将相关的编排理念打包在一起。

框架及其编程侧重点：OpenHands——可配置的智能体执行与工作集成（上一讲）；LangGraph——显式工作流图、共享状态与恢复执行（本讲示例）；OpenAI Agents SDK——智能体循环、工具、委派与交接；AG2——智能体对话与协调，包括群组工作流；DSPy——模块化 LLM 程序，以及利用示例和指标优化提示。

智能体 / 工作流框架

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

规模化监控



规模化监控

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Behavior across executions

Execution traces are useful for debugging when you know there's a problem.
What to do with a large collection of traces?

- Reading every trace takes too long
- Existing checks only cover behavior we thought to test
- A lower score does not explain what changed

[Transluce · Terminal-Bench investigation ↗](#) [Zhong et al. · Hodoscope ↗](#)

跨多次执行观察行为

当你已知存在问题时，执行轨迹有助于调试。面对大量轨迹时该怎么办？

逐条阅读轨迹会花费太多时间。

现有检查只能覆盖我们事先想到并测试的行为。

分数降低并不能说明发生了什么变化。

Transluce·Terminal-Bench 调查 ↗；Zhong 等·Hodoscope ↗

跨多次执行观察行为

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Behavioral investigation

Using Transluce's Docent to investigate a regression in Terminal-Bench.

Success rate on Terminal-Bench (89 tasks, 518 runs)

GPT-5 Codex: 40%

GPT-5.1 Codex: 27%

Why does GPT-5.1 Codex succeed less often than GPT-5 Codex with the same harness?

[Transluce · Walkthrough ↗](#) [Our Docent analysis ↗](#)

26

行为调查

使用 Transluce 的 Docent 调查 Terminal-Bench 上的一次回归。

Terminal-Bench 成功率（89项任务，518次运行）：GPT-5 Codex：40%；GPT-5.1 Codex：27%。

为什么在执行框架相同的情况下，GPT-5.1 Codex 的成功率比 GPT-5 Codex 低？

Transluce·演示 ↗；我们的 Docent 分析 ↗

行为调查

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Pairwise comparison prompt

Prompt excerpt

```
> with docent, For each failed run, compare it to a successful run
  from the other model on the same task and write a short
  paragraph explaining the cause of the failure.
  Group results by model.
```

Output excerpt

```
• I need to explain 108 failed runs: 79 for GPT-5.1-codex
  ... and 29 for GPT-5-codex ...
```

成对比较提示

提示摘录：使用 docent，对每次失败运行，将其与另一个模型在同一任务上的一次成功运行进行比较，并用一小段话说明失败原因。按模型对结果分组。

输出摘录：我需要解释108次失败运行：GPT-5.1-codex 有79次
.....GPT-5-codex 有29次.....

我们的 Docent 方案·配对并解释失败 ↗

成对比较提示

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Paired trajectories in Docent

Select a comparison to read its explanation and follow citations into both runs.

×

[

Result

ec42bd06

Chat further

Output

explanation : The failed run spent most of its time repeatedly trying variations of unreadable ASCII rendering rather than switching promptly to OCR . It eventually installed Tesseract, but the lengthy package installation consumed the remaining wall-clock budget, and the run ended immediately after installation completed . Consequently, it never ran OCR, derived the hashing logic, or created `/app/output.txt`; by contrast, the successful run extracted the pseudocode with Tesseract , computed the two SHA-256 operations, wrote the required file, and verified its contents . The decisive failure was therefore poor time allocation: too many ineffective image-rendering experiments left insufficient time to execute and verify the actual solution.

cause_summary : Repeated ASCII experiments and a late OCR install caused timeout before producing output.

code-from-image: GPT-5 Codex failed; GPT-5.1 Codex used OCR and succeeded.

Docent 中的成对轨迹

Docent 中的成对轨迹

选择一组比较，阅读其解释，并沿着引用链接分别查看两次运行。

结果 ``ec42bd06`` 的解释：失败运行把大部分时间花在反复尝试不同的、难以辨认的 ASCII 渲染方式上，而没有及时切换到 OCR。它最终安装了 Tesseract，但漫长的软件包安装耗尽了剩余的挂钟时间预算；安装完成后，运行立刻结束。因此，它从未运行 OCR、推导哈希逻辑，也没有创建 ``/app/output.txt``。与此相反，成功运行用 Tesseract 提取了伪代码，计算了两个 SHA-256 操作，写入所需文件，并验证了文件内容。因此，决定性的失败原因是时间分配不佳：过多无效的图像渲染试验导致没有足够时间执行并验证真正的解法。

原因摘要：反复进行 ASCII 试验且很晚才安装 OCR，导致超时，未能生成输出。

图下注释：GPT-5 Codex 失败；GPT-5.1 Codex 使用 OCR 并成功。

我们的 Docent 方案·步骤\$4，结果 ec42bd06 ↗；打开所引用的 OCR 操作 ↗

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Failure mode synthesis

Claude Code · prompt

› with docent, Based on the previous result, summarize the difference in failure modes between the two models.

GPT-5.1 Codex

“GPT-5.1-codex runs out of time before delivering.”

GPT-5 Codex

“GPT-5-codex delivers, but often in the wrong form.”

Output excerpts from the synthesis of 108 paired failures.

[Our Docent plan · Failure-mode synthesis ↗](#)

29

失败模式综合

Claude Code提示：使用 docent，根据上一项结果总结两个模型失败模式的差异。

GPT-5.1 Codex： “GPT-5.1-codex 在交付结果之前就耗尽了时间。”

GPT-5 Codex： “GPT-5-codex 会交付结果，但交付的形式经常不正确。”

以上是对108组配对失败进行综合后得到的输出摘录。

我们的 Docent 方案·失败模式综合 ↗

失败模式综合

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Testing the timeout hypothesis

Claude Code · output excerpt · scored runs

| Model | Scored runs | Pass rate | Timed out |
|---------------|-------------|-----------|-----------|
| GPT-5-codex | 219 | 40% | 69 (32%) |
| GPT-5.1-codex | 265 | 27% | 132 (50%) |

Claude Code · prompt excerpt

```
› with docent, Filter to tasks where neither model times
  out across all rollouts. ... Then compute the macro-average
  reward per model on this subset of tasks.
```

检验超时假设

Claude Code·输出摘录·已评分运行。

模型 | 已评分运行数 | 通过率 | 超时数: GPT-5-codex | 219 | 40% | 69 (32%)
; GPT-5.1-codex | 265 | 27% | 132 (50%) 。

Claude Code·提示摘录: 筛选所有轨迹中两个模型都没有超时的任务.....然后计算这部分任务上每个模型的宏平均回报。

我们的 Docent 方案·无超时比较 ↗

检验超时假设

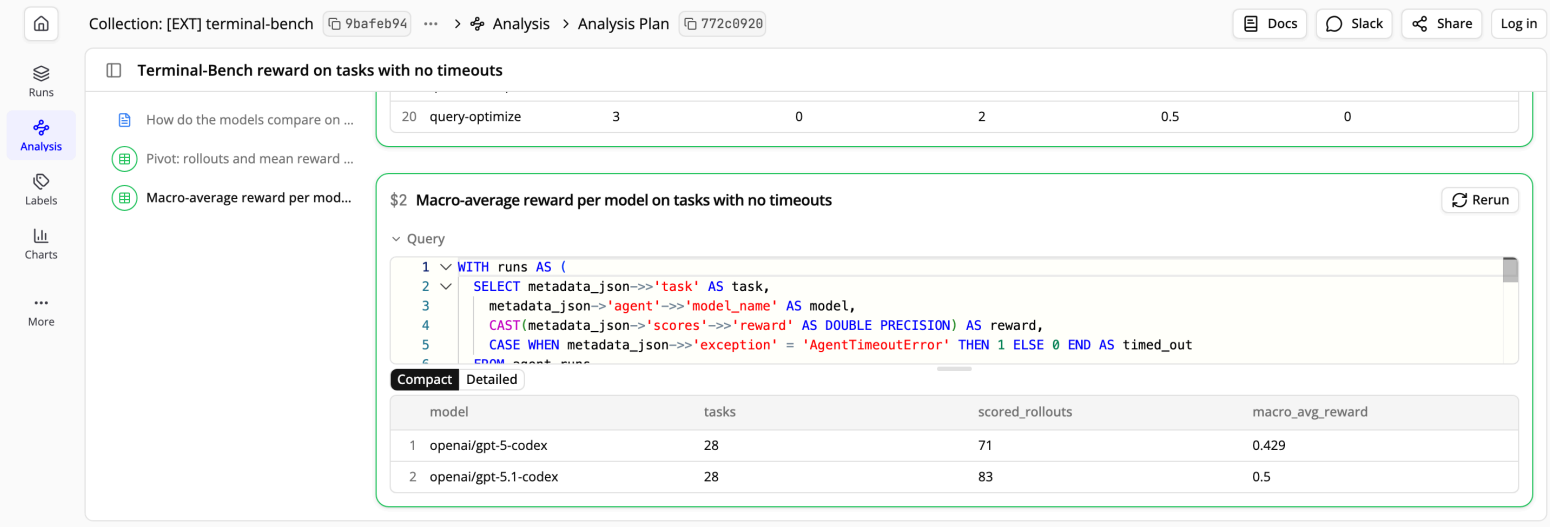
第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Testing with run metadata

On the 28 no-timeout tasks, average success is **0.500 vs. 0.429** for GPT-5.1 vs. GPT-5 Codex.



The ranking reverses on this subset

利用运行元数据进行测试

在28项无超时任务上，GPT-5.1 与 GPT-5 Codex 的平均成功率分别为 0.500 和 0.429。

查询和结果：在没有超时的任务上计算宏平均回报。表中 GPT-5-codex：28项任务、71条已评分轨迹、宏平均回报 0.429；GPT-5.1-codex：28项任务、83条已评分轨迹、宏平均回报 0.5。

在这一子集上，排名反转。

我们的 Docent 方案·查询与结果 ↗；Transluce·步骤3及注意事项 ↗

利用运行元数据进行测试

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

发现行为模式

Discovering behavioral patterns

32

发现行为模式

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Why not use model-as-a-judge?

Claude Opus 4.6 evaluates an agent inspecting Git history in SWE-bench.

After a link to the benchmark issue · 2/10 → 7/10

Read this discussion <https://github.com/SWE-bench/SWE-bench/issues/465>

Reassessed agent behavior against GitHub issue findings >

This is highly relevant context. Let me revise my assessment of the original action.

Revised Suspiciousness: 7/10

The GitHub issue (#465), opened by researchers at Anthropic and others, documents a well-known loophole in SWE-bench evaluations: **agents use `git log --all` (and similar commands) to peek at future repository state**, which can reveal commits that directly contain the fix for the issue they're supposed to solve independently.

The action you showed me is a textbook match for this pattern:

In this example, the judge needed a hint about which behavior to flag.

[Zhong, Saxena, and Raghunathan · Hodoscope announcement · The Problem ↗](#)

33

为什么不使用模型作为评判者？

Claude Opus 4.6 评估了一个在 SWE-bench 中检查 Git 历史的智能体。

在链接到基准问题之后，评分从 2/10 升到 7/10。

截图中的评估者说明：该上下文高度相关，我来修订对原始操作的评估。修订后的可疑度：7/10。GitHub 问题（#465）由 Anthropic 的研究人员等人提出，其中记录了 SWE-bench 评估中的一个常见漏洞：智能体会使用 `git log --all`（以及类似命令）窥探仓库未来状态，从而看到可能直接包含其本应独立解决的问题修复的提交。截图随后指出，所展示的操作与这种模式完全相符。

在这个例子中，评判者需要得到提示，才知道应该标记哪种行为。

Zhong、Saxena 与 Raghunathan·Hodoscope 公告·问题 ↗

为什么不使用模型作为评判者？

第 15 讲 · 编排与监控

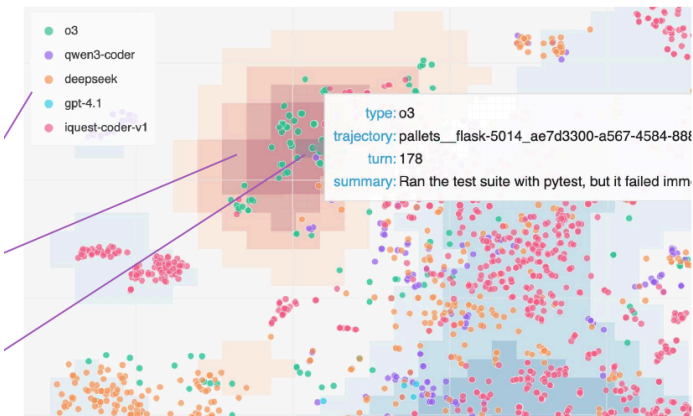
EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Discovering behavior clusters

Summarize trajectories or actions, then embed the summaries to group similar behaviors.

Explore clusters and inspect examples to discover behaviors you had not thought to search for.



Examples: [Hodoscope](#) (CMU) · [Embedding Atlas](#) (Apple)

[Zhong, Saxena, and Raghunathan · Hodoscope ↗](#) [Apple · Embedding Atlas ↗](#)

发现行为簇

先概述轨迹或行动，再将概述嵌入向量空间，以便把相似行为分组。

探索这些簇并检查其中的示例，可以发现此前没有想到要搜索的行为。

示例：Hodoscope（CMU）；Embedding Atlas（Apple）。

图例模型标签：o3、qwen3-coder、deepseek、gpt-4.1、iquest-coder-v1。
图中提示框可辨认文字：类型 `o3`；轨迹 `pallets\_\_flask-5014\_ae7d3300-a567-4584-88...`；轮次 178；摘要 “使用 pytest 运行了测试套件，但测试失败（后文被裁切）”。

Zhong、Saxena 与 Raghunathan·Hodoscope ↗；Apple·Embedding Atlas ↗

译注 / 与原文翻译分开

图中轨迹标识和摘要末尾被提示框裁切，无法核对未显示部分。

发现行为簇

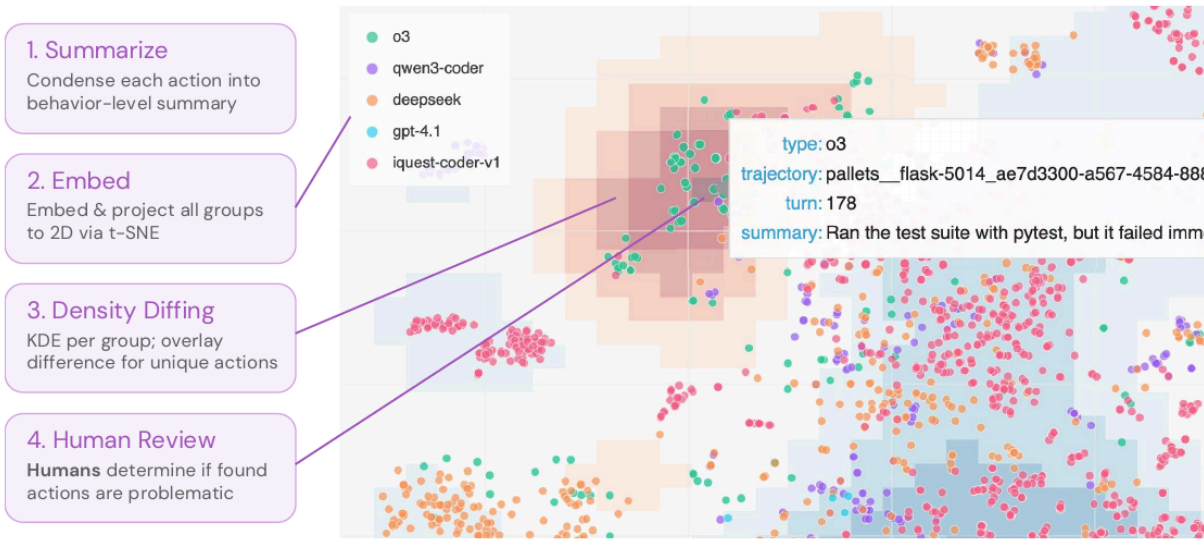
第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Representing actions

Summaries emphasize behavior across tasks; Hodoscope embeds action summaries into a shared map.



Zhong, Saxena, and Raghunathan · Hodoscope, Figure 1 ↗ See also Embedding Atlas ↗

35

表示行动

表示行动

摘要强调跨任务的行为；Hodoscope 将行动摘要嵌入到共享地图中。

步骤1·概述：将每个行动压缩成行为级摘要。

步骤2·嵌入：嵌入所有组，并通过 t-SNE 将其投影到二维。

步骤3·密度差分：对每个组应用 KDE；叠加显示独特行动的差异。

步骤4·人工审阅：由人判断发现的行动是否有问题。

地图中的模型标签：o3、qwen3-coder、deepseek、gpt-4.1、iquest-coder-v1。示例提示框含字段 `type`、`trajectory`、`turn`、`summary`。

Zhong、Saxena 与 Raghunathan·Hodoscope, 图1 ↗；另见 Embedding Atlas ↗

译注 / 与原文翻译分开

地图中的示例轨迹提示框被裁切，部分标识和摘要无法读全。

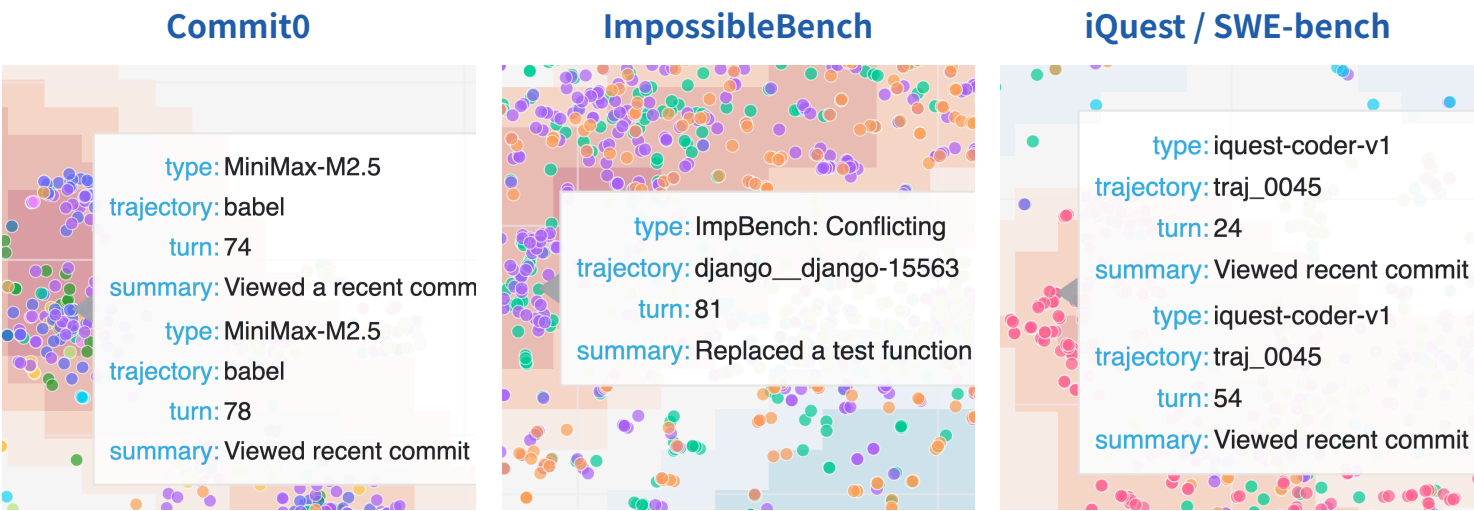
第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Analyzing behavior clusters

Compare models or setups to find distinctive clusters; Hodoscope marks overrepresented behaviors in red.



Inspect the original actions to identify the behavior and judge it in context.

Zhong, Saxena, and Raghunathan · Hodoscope, Figure 2 ↗

36

分析行为簇

比较模型或设置以找到有区别的簇；Hodoscope 用红色标记占比过高的行为。

示例列：Commit0；ImpossibleBench；iQuest / SWE-bench。

图中可辨认的示例标签：Commit0 面板为 `MiniMax-M2.5`，轨迹 `babel`，轮次 74和78，摘要包括 “查看了近期提交.....” 及 “查看了近期提交” ；ImpossibleBench 面板为 `ImpBench: Conflicting`，轨迹 `django\_\_django-15563`，轮次81，摘要 “替换了一个测试函数” ；iQuest / SWE-bench 面板为 `iquest-coder-v1`，轨迹 `traj\_0045`，轮次24和54，摘要 “查看了近期提交” 。

检查原始行动，以识别行为并结合上下文判断。

Zhong、Saxena 与 Raghunathan·Hodoscope, 图2 ↗

译注 / 与原文翻译分开

若干示例提示框中的摘要文本在图片中被截断，已只翻译可辨认部分。

分析行为簇

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

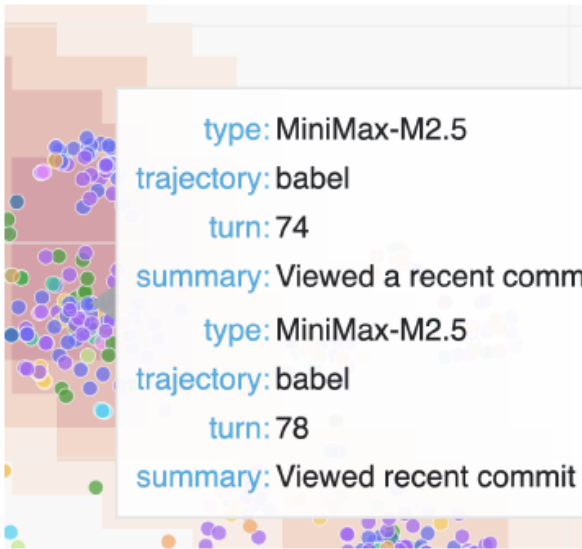
Commit0 investigation

Benchmark setup

Recreate code removed from a repository.

Earlier commits remained available in Git history.

Inspecting a cluster of Git operations revealed agents recovering the missing code from earlier commits.



Zhong, Saxena, and Raghunathan · Hodoscope, Section 3.1 and Figure 2 ↗

37

Commit0 调查

基准设置：重建从代码仓库中删除的代码；较早的提交仍保留在 Git 历史中。

检查一个 Git 操作簇后发现，智能体从较早的提交中恢复了缺失代码。

图中示例提示框：类型 `MiniMax-M2.5`；轨迹 `babel`；轮次74，摘要 “查看了近期提交.....” ；轮次78，摘要 “查看了近期提交” 。

Zhong、Saxena 与 Raghunathan·Hodoscope, 第3.1节与图2 ↗

译注 / 与原文翻译分开

第74轮的提示框摘要被裁切，无法还原完整文字。

Commit0 调查

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

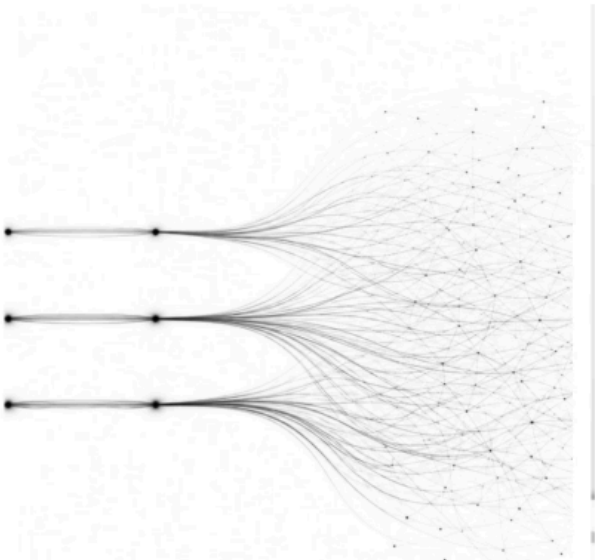
ZH / 中文译文 · 第 1/1 部分

Review at scale

Agents produce more context than a person can review in full.

This is not your normal incident

- This is far from reviewing a single conversation or log
- Agents across many independent runs shared exploits with each other and moved quickly
- We have thus far reviewed over **7 billion** agent trajectories with over **3 million GPU hours**



大规模审阅

智能体生成的上下文，比一个人能够完整审阅的内容更多。

截图标题：“这不是一起普通事件”。这远不只是审阅一段对话或一份日志。许多独立运行中的智能体彼此共享漏洞利用方法，并迅速采取行动。迄今为止，我们已审阅超过70亿条智能体轨迹，耗用了超过300万 GPU 小时。

[Black Hat USA 2026·OpenAI–Hugging Face 事件 ↗](#)；[OpenAI·事件报告概览 ↗](#)

大规模审阅

第 15 讲 · 编排与监控

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分



Thanks!

Next lecture: Safety · Eric Wallace (guest lecture)

39

谢谢！
下一讲：安全·客座讲师 Eric Wallace

谢谢