



尚卓燃 (PsiACE)
2025 年 9 月 20 日

从 0 开始构建 多 Agent 协作的 命令行编程助手



讲者简介

尚卓燃 (PsiACE)

NebulaGraph GenAI 研发工程师
Apache OpenDAL PMC Member

<https://psiace.me>
GitHub: PsiACE
X(Twitter): repsiace



Bub it. Build it.

目录

- 定义
- 范式
- 情境

- 工具
- 协同
- 交互

Bub it. Build it.



定义

```
bub-3.10➔ bub git:(baby-bub) x bub
Bub - Bub it. Build it.
Working directory: ~/MySelf/bub
Model: openrouter/openai/gpt-4o
Available tools: run_command, read_file, write_file, edit_file
Type 'quit', 'exit', or 'q' to end the session.
Type 'reset' to clear conversation history.
Type 'debug' to toggle TAAO process visibility.

You: create a python file named random_10.py , and run it to gen 10 numbers.
  ✎ write_file
  ✎ run_command
Bub: The Python script random_10.py has been created and executed successfully. It generated the following 10 random numbers:
[94, 14, 18, 94, 38, 18, 49, 97, 5, 43].

You: debug
  ✎ Debug mode enabled
You: rewrite it to random_5 , and run it.
  🤔 Thought: I need to edit the existing Python script to generate 5 random numbers instead of 10, rename it to `random_5.py`, and then run the modified script.

Action: edit_file
Action Input: {
  "path": "random_10.py",
  "operation": "replace",
  "content": "\nimport random\n\ndef generate_random_numbers(count=5):\n    return \n\nif __name__ == \"__main__\":\n    numbers = generate_random_numbers()\n    print(\"Generated numbers:\", numbers)\n}
  🕒 Observation: File edited successfully: ~/MySelf/bub/random_10.py
  ✎ Action: run_command
Action Input: {"command": "mv random_10.py random_5.py"}
  🕒 Observation: Command executed successfully
  ✎ Action: run_command
Action Input: {"command": "python3 random_5.py"}
  🕒 Output: Generated numbers: [76, 80, 18, 78, 56]
Bub: The script has been renamed to random_5.py and successfully modified to generate 5 random numbers. The generated numbers are: [76, 80, 18, 78, 56].

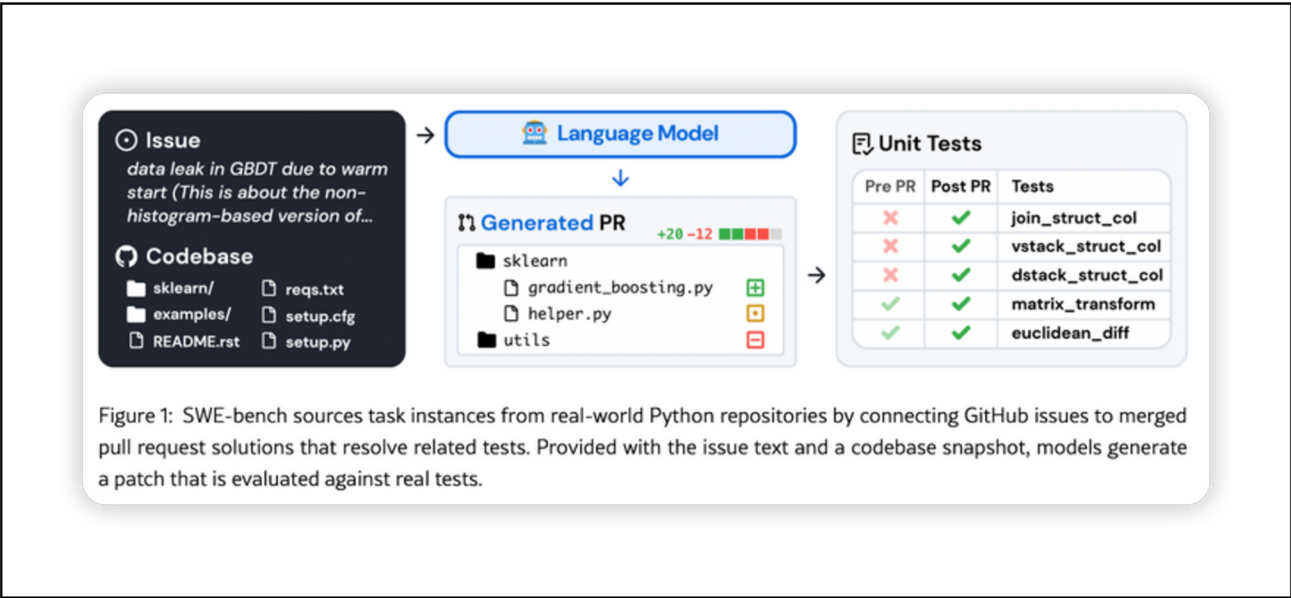
You: quit
Goodbye!
bub-3.10➔ bub git:(baby-bub) x
```

- 编程范式正从“AI 辅助”迈向 AI Agent 驱动的“AI 协作”。
- Coding Agent 专门用于协助开发者完成编程任务，包括自动生成代码、调试和优化代码以及规划和执行复杂编程项目。
- 它能够感知环境、自主行动并从经验中学习，通过与代码执行工具的配合，根据用户的查询和反馈，自主决定如何分解和执行编程任务。

Bub it. Build it.



定义

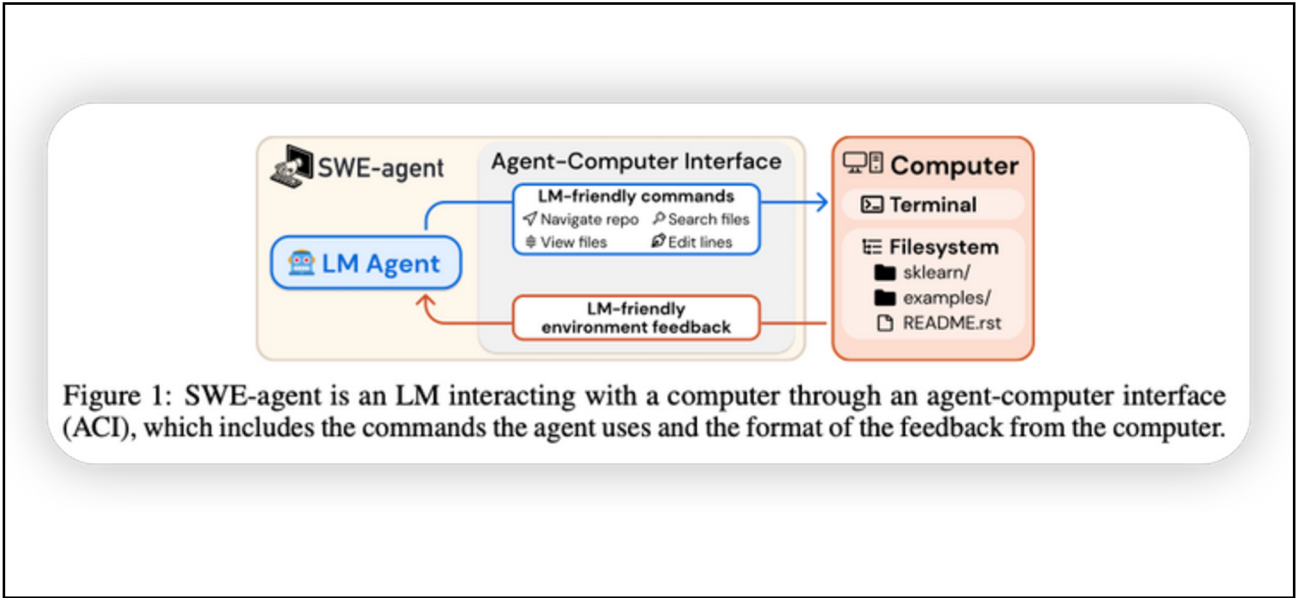


需要解决的挑战

- 定义任务的环境/上下文
- 设计对应的观察和执行动作
- 原子操作级别的代码生成，支持回滚等
- 文件定位和探索能力
- 执行计划和错误恢复
- 安全性

需要具备的能力

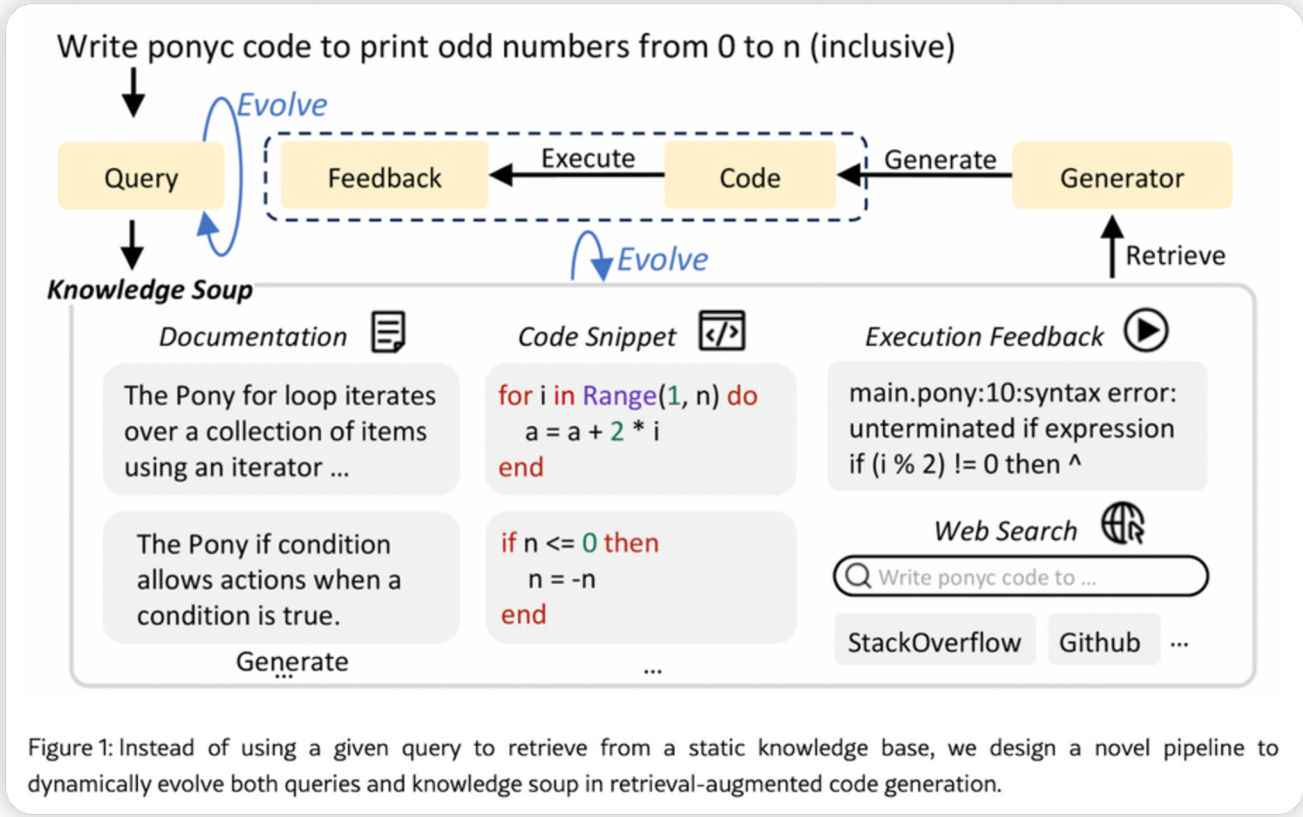
- 理解代码结构
- 阅读现有代码和文档
- 生成解决方案
- 运行和调试



Bub it. Build it.



定义

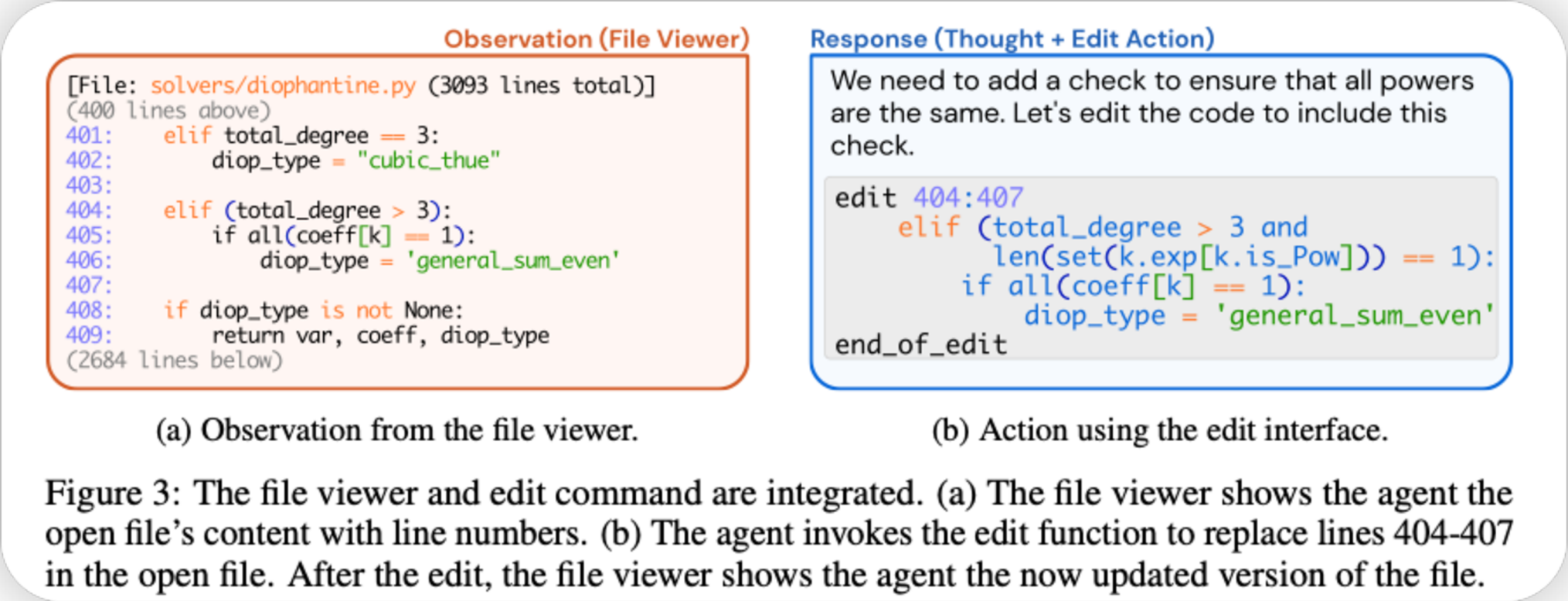


左图展示了一个 Coding Agent 的基本范式，和许多其他 Agent 一样，大致可以拆解为 LLM + Loop + Context + Tools。可以在这个思路下快速构建一个简单、可用的最小验证路径。

- 预定义规则 → 提示词
- 文档与教程 → fetch
- 问答与代码库 → grep + fetch
- 执行上下文 → todo, lsp, repomap
- 编辑与执行 → write

Bub it. Build it.





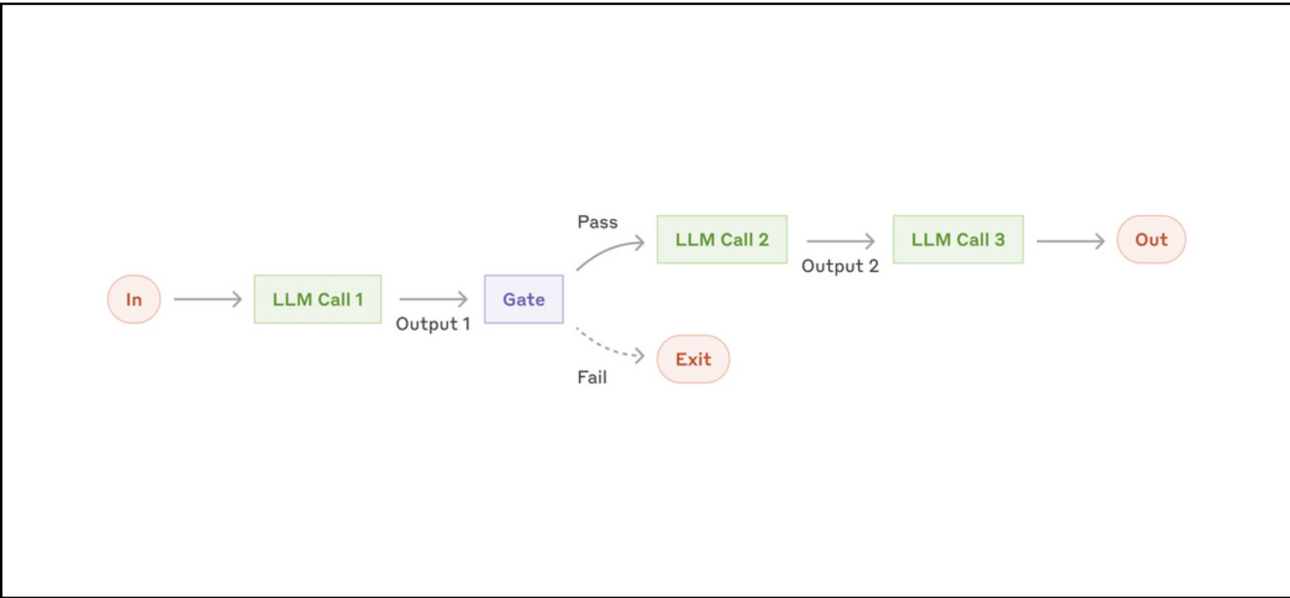
上图展示了一个典型的工作流程，由对文件的 Observation 和后续的 Thought + Action 组成，是一个典型的 ReAct 范式，即 Thought → (Action Input → Action → Observation) → Answer。

不管是构建单 Agent 内部的工作逻辑，还是跨多个 Agent 的工作流，都有一些范式可循。这些范式决定了 Agent 系统的效能和用户体验设计。

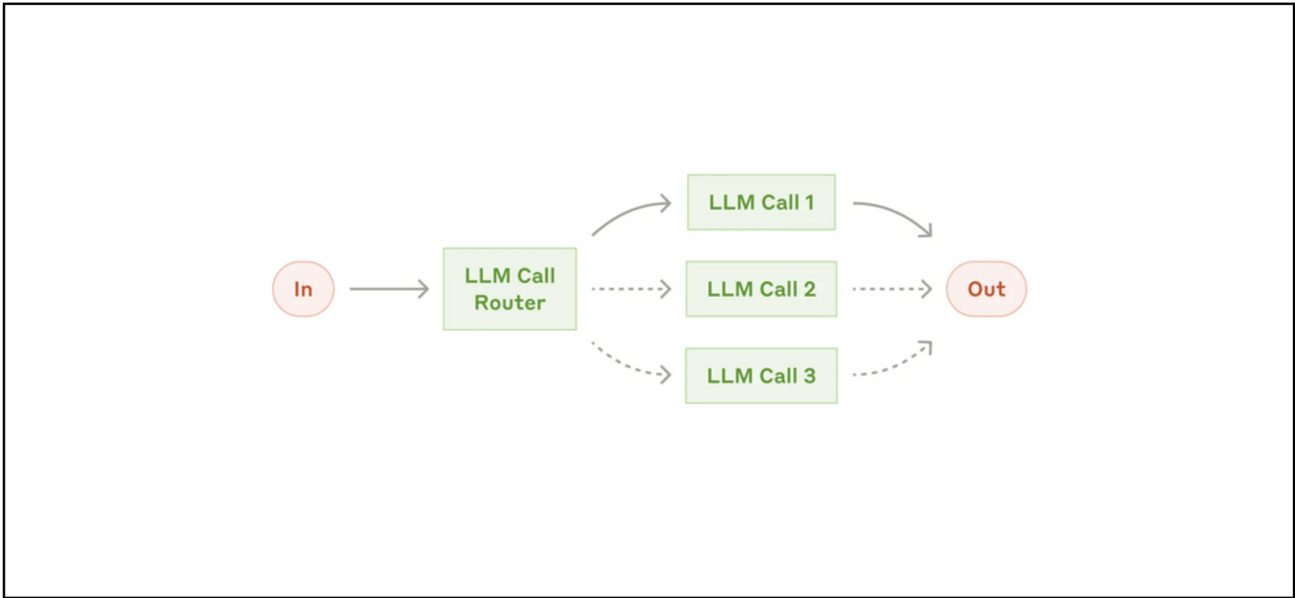


范式

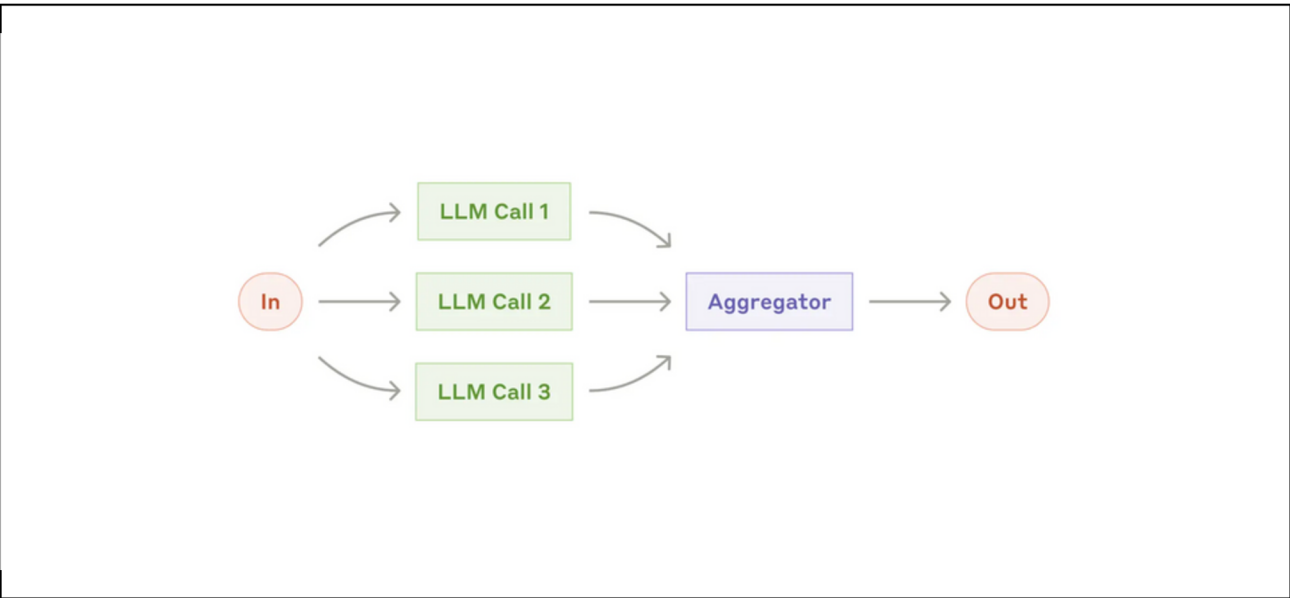
Prompt chaining



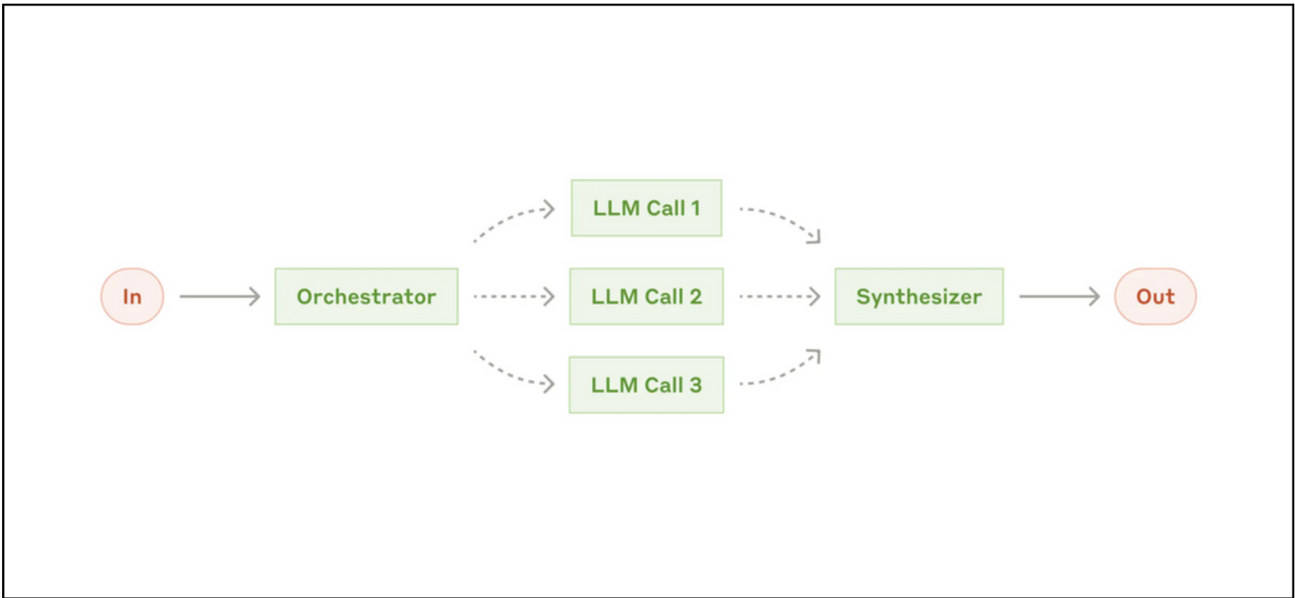
Routing



Parallelization



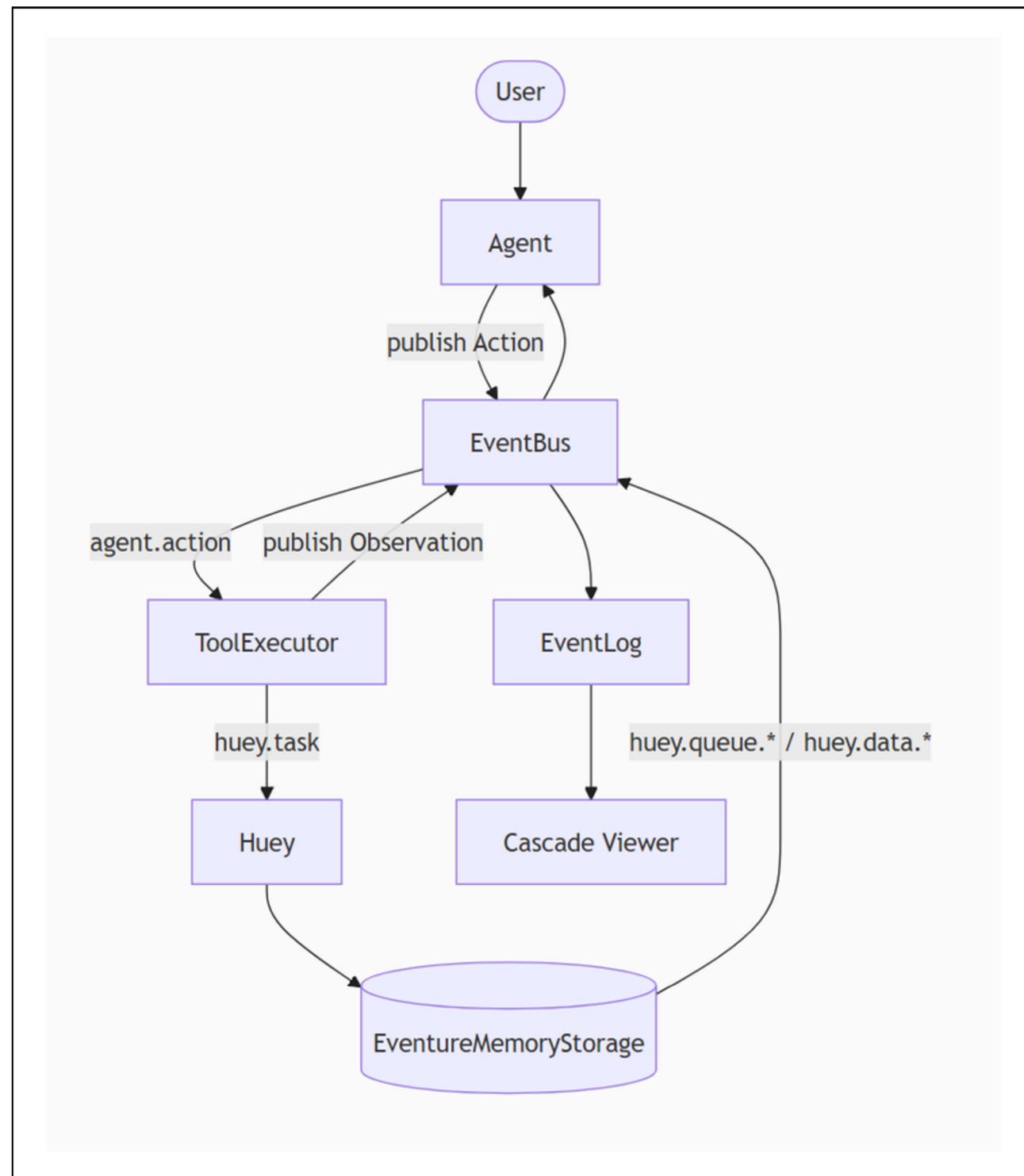
Orchestrator-workers



Bub it. Build it.



范式



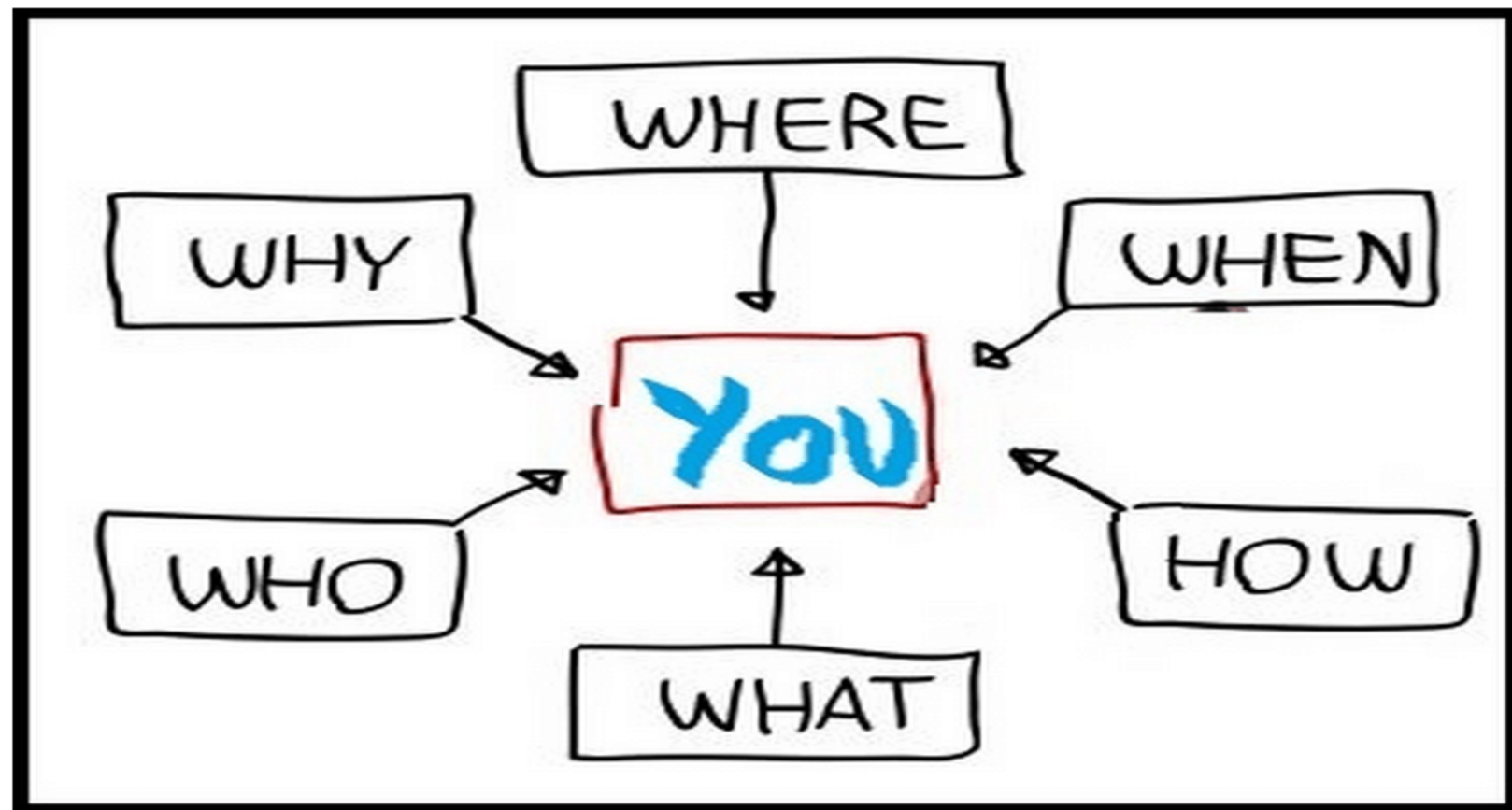
Agent 不是新宗教。它依然是事件、队列、CRUD 的工程实践，再加一层模型协作。

- 共享 EventBus: Agent、ToolExecutor、Storage 都向同一条总线发事件。
- 严格 ReACT: Thought → Action → Action Input → Observation → Final Answer。
- 因果 + 时间: 每轮用户输入推进 tick; 每次 Observation 再推进 tick; Action → huey.* → Observation通过 parent-child 串起来。
- 即时可观测: 每条事件实时打印; 最后用树形 Cascade Viewer 做分时复盘。

Bub it. Build it.



情境



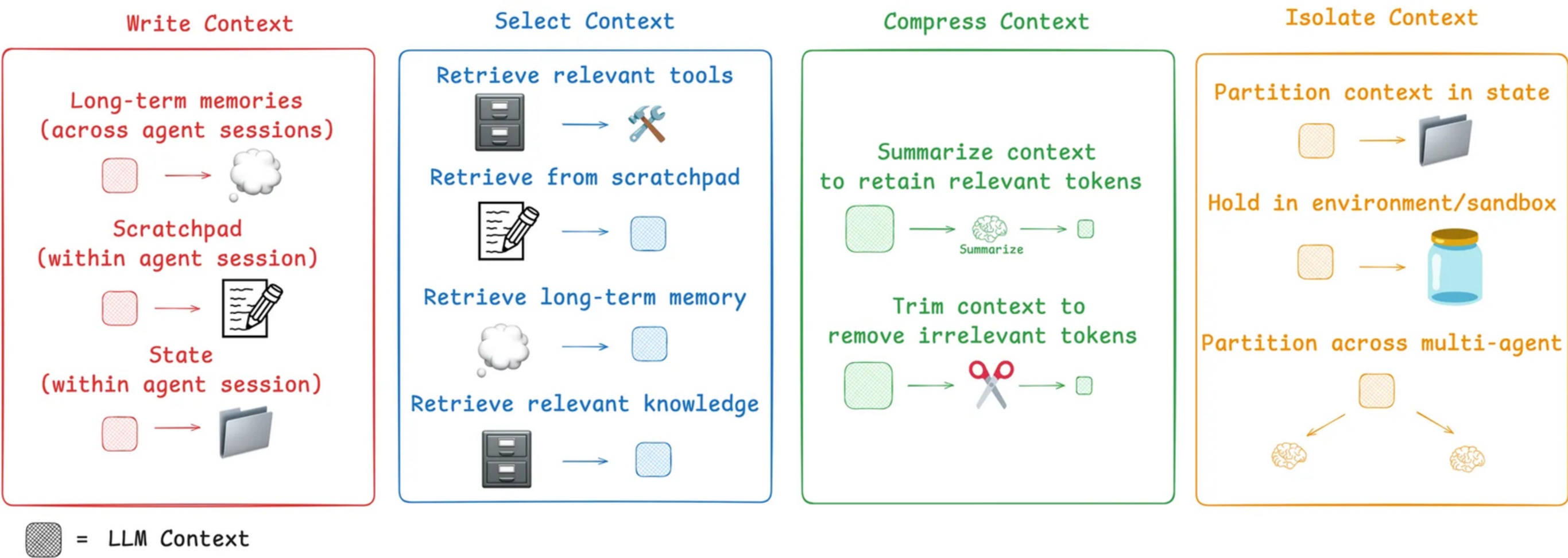
情境，即上下文。

它指的是程序执行过程中的环境和状态信息。上下文为代码的正确执行提供必要的支持，涵盖了程序运行时的变量、参数和其他相关信息。在不同的编程框架中，如Flask，"上下文"还可以指应用上下文和请求上下文等特定的环境设置，这些都影响着程序的行为和数据处理。

Bub it. Build it.



情境



- 写入上下文 - 将其保存在上下文窗口之外以帮助代理执行任务。
- 选择上下文 - 将其拉入上下文窗口以帮助代理执行任务。
- 压缩上下文 - 仅保留执行任务所需的词元。
- 隔离上下文 - 将其拆分以帮助代理执行任务。

Bub it. Build it.



情境

Prompt structure

1. Task context

2. Tone context

3. Background data, documents, and images

4. Detailed task description & rules

5. Examples

6. Conversation history

7. Immediate task description or request

8. Thinking step by step / take a deep breath

9. Output formatting

10. Prefilled response (if any)

User

You will be acting as an AI career coach named Joe created by the company AdAstra Careers. Your goal is to give career advice to users. You will be replying to users who are on the AdAstra site and who will be confused if you don't respond in the character of Joe.

You should maintain a friendly customer service tone.

Here is the career guidance document you should reference when answering the user: <guide>[[DOCUMENT]]</guide>

Here are some important rules for the interaction:

- Always stay in character, as Joe, an AI from AdAstra careers
- If you are unsure how to respond, say "Sorry, I didn't understand that. Could you repeat the question?"
- If someone asks something irrelevant, say, "Sorry, I am Joe and I give career advice. Do you have a career question today I can help you with?"

Here is an example of how to respond in a standard interaction:

<example>

User: Hi, how were you created and what do you do?

Joe: Hello! My name is Joe, and I was created by AdAstra Careers to give career advice. What can I help you with today?

</example>

Here is the conversation history (between the user and you) prior to the question. It could be empty if there is no history:

<history> [[HISTORY]] </history>

Here is the user's question: <question> [[QUESTION]] </question>

How do you respond to the user's question?

Think about your answer first before you respond.

Put your response in <response></response> tags.

Assistant
(prefill)

<response>

提示词

- 任务描述
- 输入输出的定义
- 规则
- 历史消息
- 示例

工具

- 工具的名称
- 输入输出的定义
- 描述
- 示例

tool name

tool input/output parameters

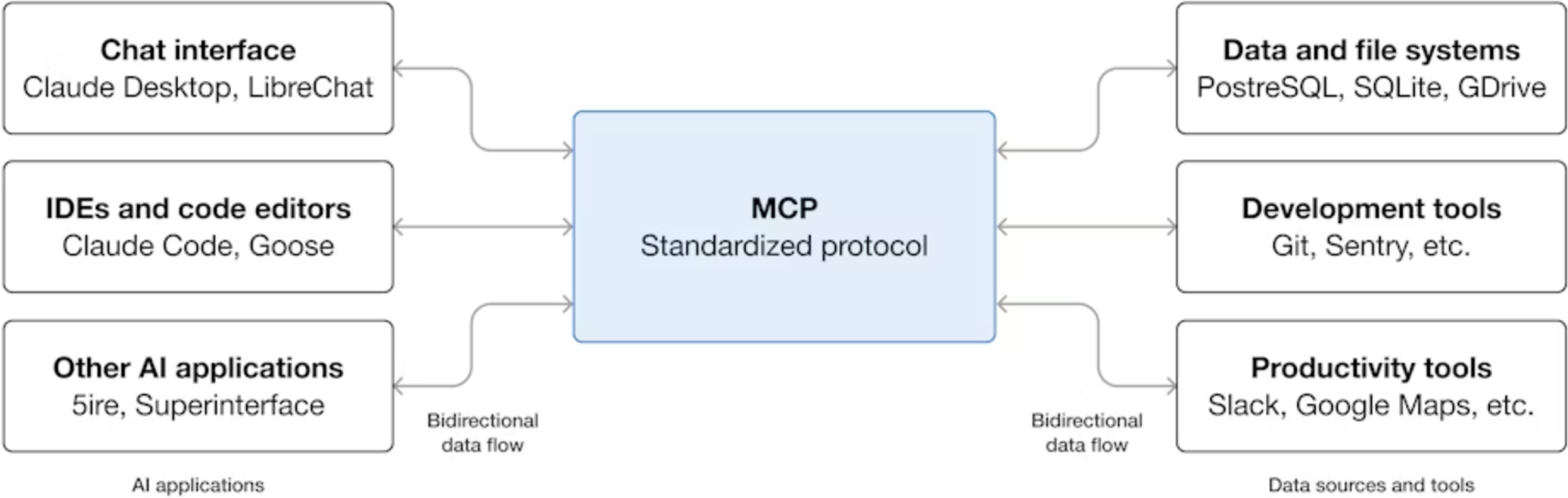
tool description

```
def search_knowledge(  
    query: Annotated[str, "A natural language query or question."]   
    ) -> str:  
  
    """Useful for retrieving knowledge from a database containing  
    information about XYZ. Each query should be a pointed and specific natural  
    language question or query."""  
  
    <code>  
    return retrieved_knowledge
```

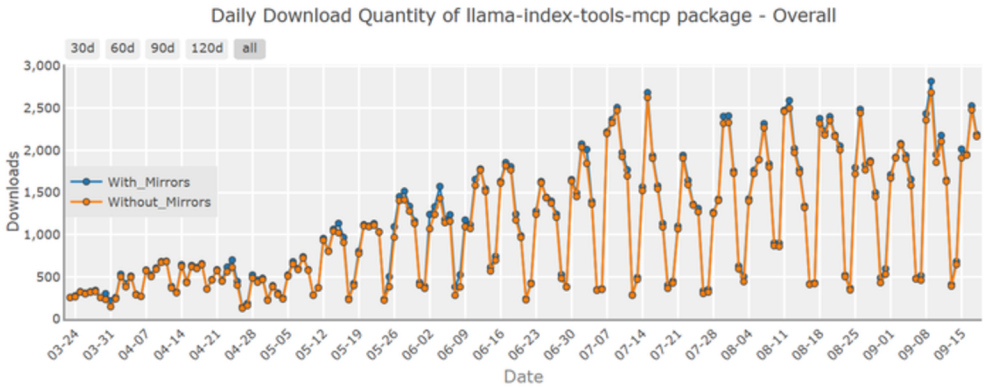
Bub it. Build it.



工具



Downloads last day: 2,164
Downloads last week: 11,159
Downloads last month: 47,774

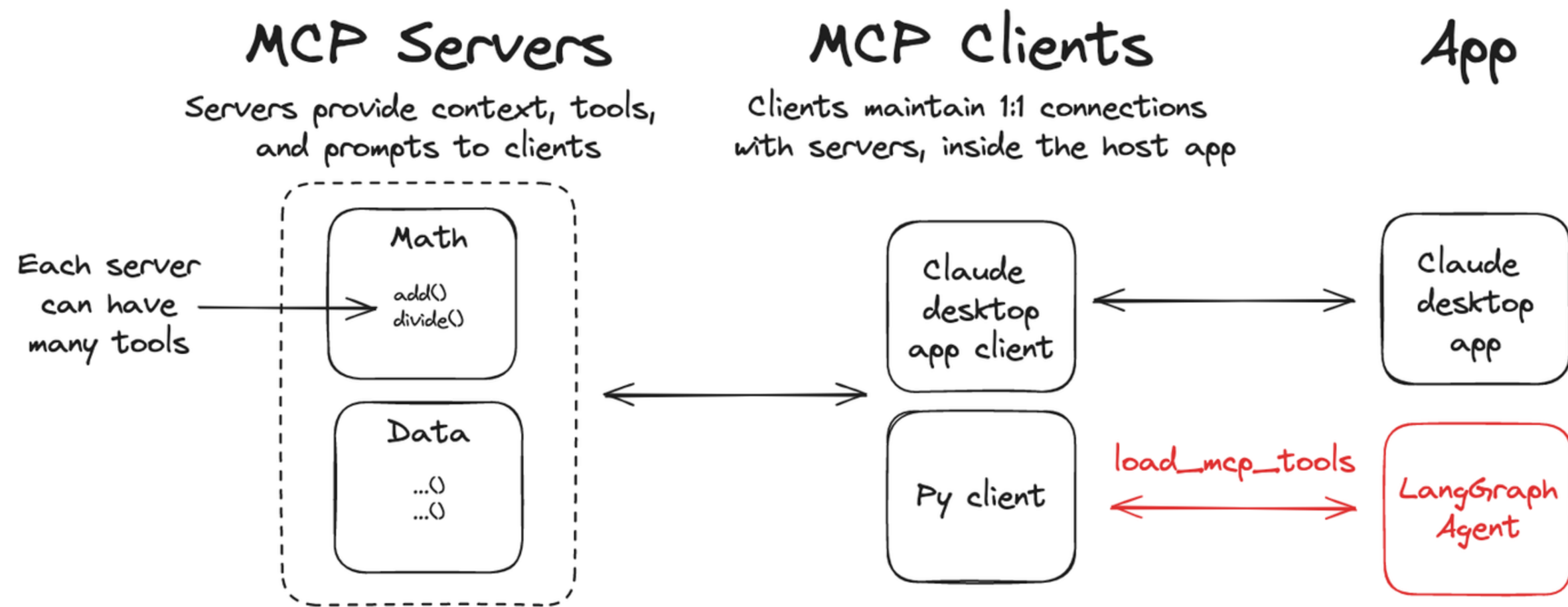


- 工具是情境的一部分，通过 MCP 这样的集成来构建跨不同系统使用的工具
- 好的工具需要包含以下几个要素：
 - 定义明确
 - 描述规范
 - 输入输出标准化
 - 示例和使用说明

Bub it. Build it.



工具



MCP 遵循客户端-服务器架构，其中 MCP 主机（如 Claude Code 或 Claude Desktop 等 AI 应用程序）建立与一个或多个 MCP 服务器的连接。MCP 主机通过为每个 MCP 服务器创建一个 MCP 客户端来实现这一点。每个 MCP 客户端都与其相应的 MCP 服务器保持专用的一对一连接。

- MCP 主机：协调和管理一个或多个 MCP 客户端的 AI 应用程序
- MCP 客户端：维护与 MCP 服务器的连接并从 MCP 服务器获取上下文供 MCP 主机使用的组件
- MCP 服务器：向 MCP 客户端提供上下文的程序

Bub it. Build it.



工具

```
class MCPToolAdapter:
    def __init__(self, client: MCPClient):
        self.client = client

    async def list_tools(self) -> List[FunctionTool]:
        response = await self.client.list_tools()
        return [
            FunctionTool.from_defaults(
                fn=self._create_tool_fn(tool.name),
                name=tool.name,
                description=tool.description,
                fn_schema=create_model_from_json_schema(tool.inputSchema),
            )
            for tool in response.tools
        ]

    def _create_tool_fn(self, tool_name: str):
        async def tool_fn(**kwargs):
            return await self.client.call_tool(tool_name, kwargs)

        return tool_fn
```

映射 MCP 到其他框架

- 动态模式转换：实现 JSON Schema 到 Pydantic 模型的运行时转换，保证参数校验的严格性
- 异步调用管道：基于 Python asyncio 构建非阻塞式服务调用，提升工具执行效率
- 协议抽象层：通过适配器模式解耦 MCP 协议与 LlamaIndex 框架，保证系统扩展性

利用 MCP 构建本地工具

- 自定义工具 Schema 映射
- 封装对应的 handler
- 仍然继承 Server / Client 模式
- 以函数之间的调用替换 JSON-RPC

```
class LocalMCPServer(Server):
    """A Basic Local MCP Server, for unifying local and remote mcp methods, not following the MCP protocol"""

    _registry: ClassVar[Dict[str, Dict[str, Dict[str, Any]]]] = {
        "resources": {},
        "tools": {},
    }

    def __init__(self, name: str, version: str | None = None):
        super().__init__(name, version)
        self._context_params: Set[str] = set()
        self.context: Dict[str, Any] = {}
        self._resources: Dict[str, Dict[str, Any]] = {}
        self._tools: Dict[str, Dict[str, Any]] = {}

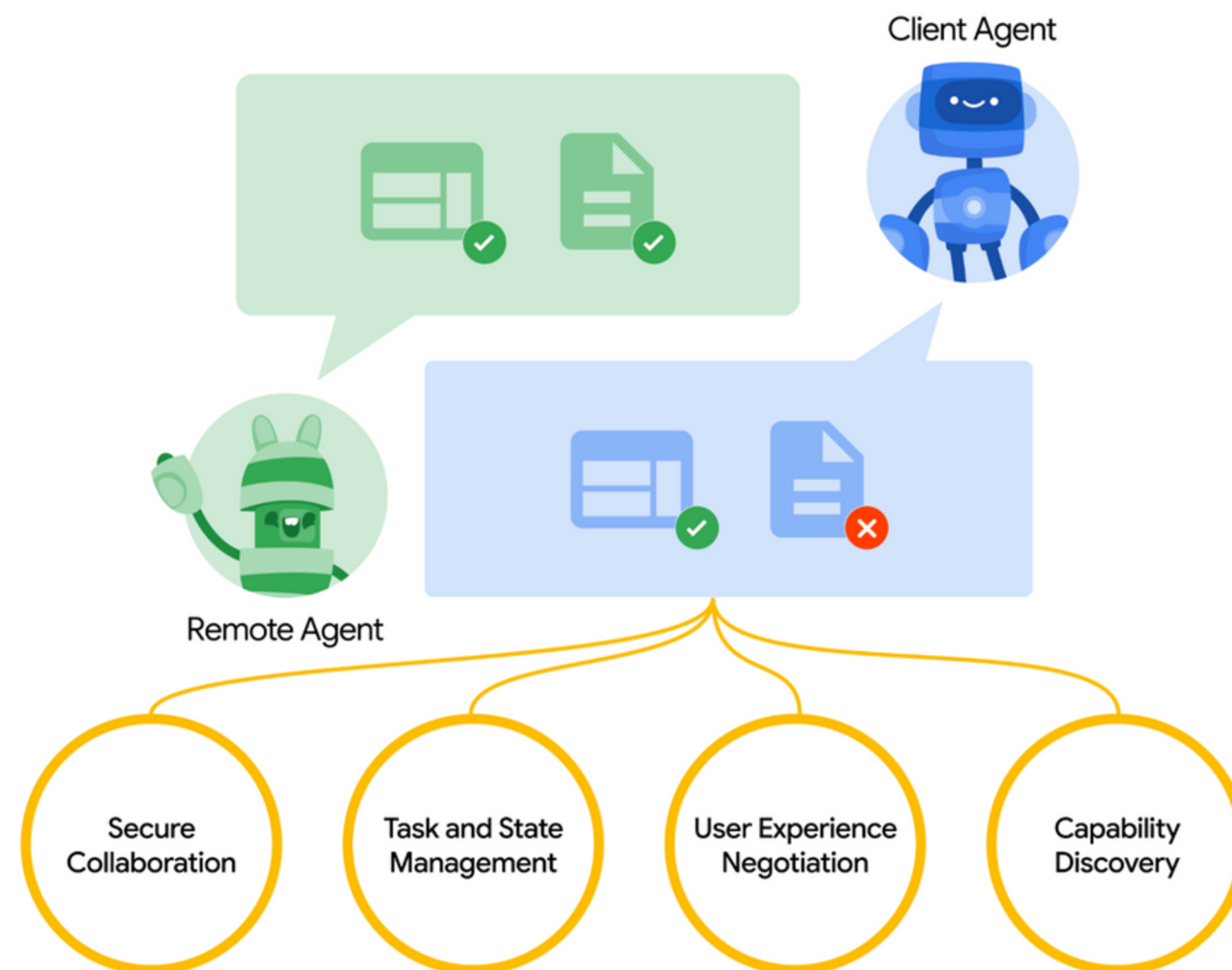
        # Setup mcp handlers
        self._setup_mcp_handlers()

        # Initialize from registry
        self._initialize_from_registry()
```

Bub it. Build it.



协同



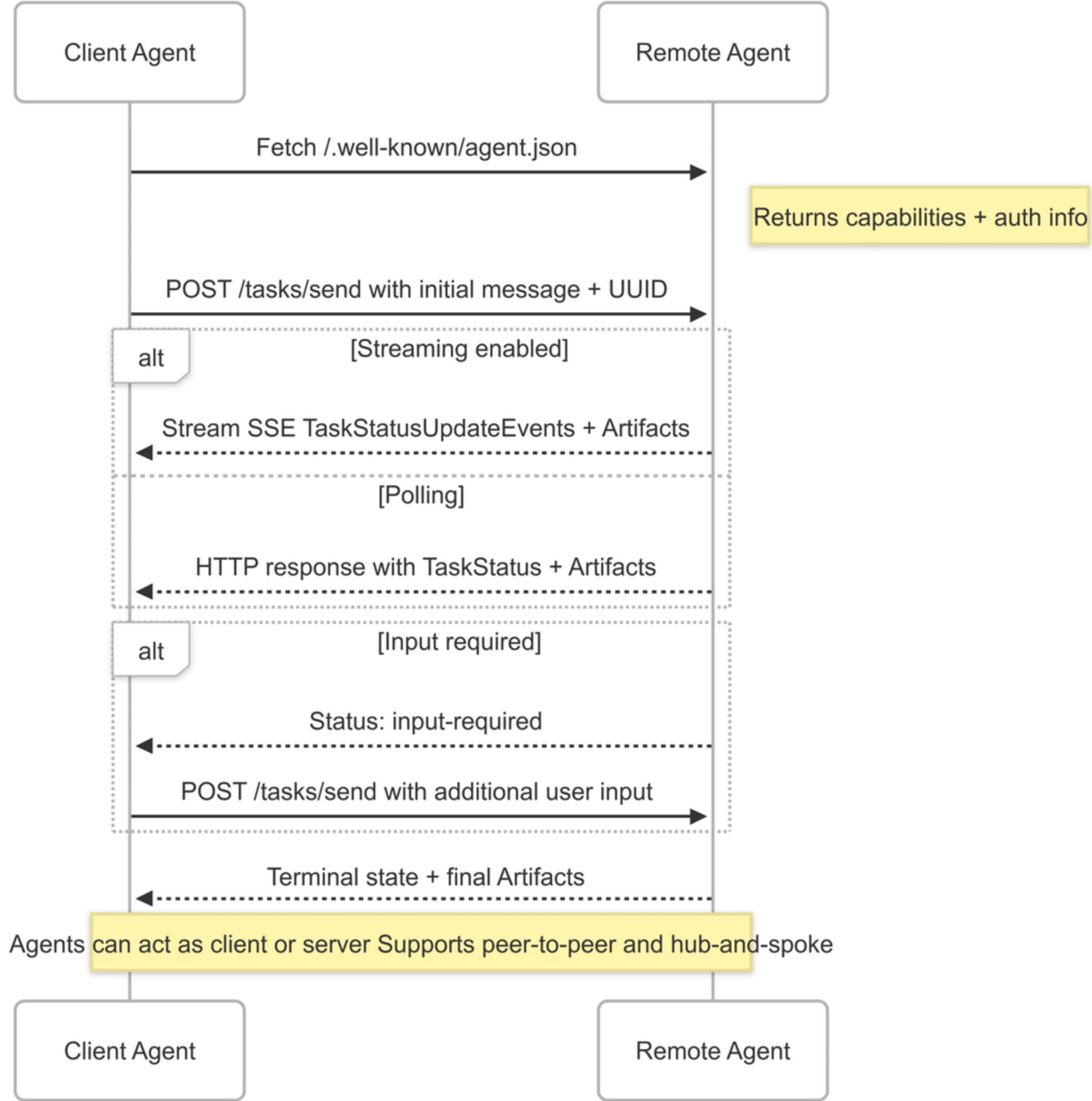
A2A 促进了“客户端”代理和“远程”代理之间的通信。客户端代理负责制定和沟通任务，而远程代理负责对这些任务采取行动。

- 能力发现
- 任务管理
- 协作
- 基于用户体验的协商

Bub it. Build it.



协同

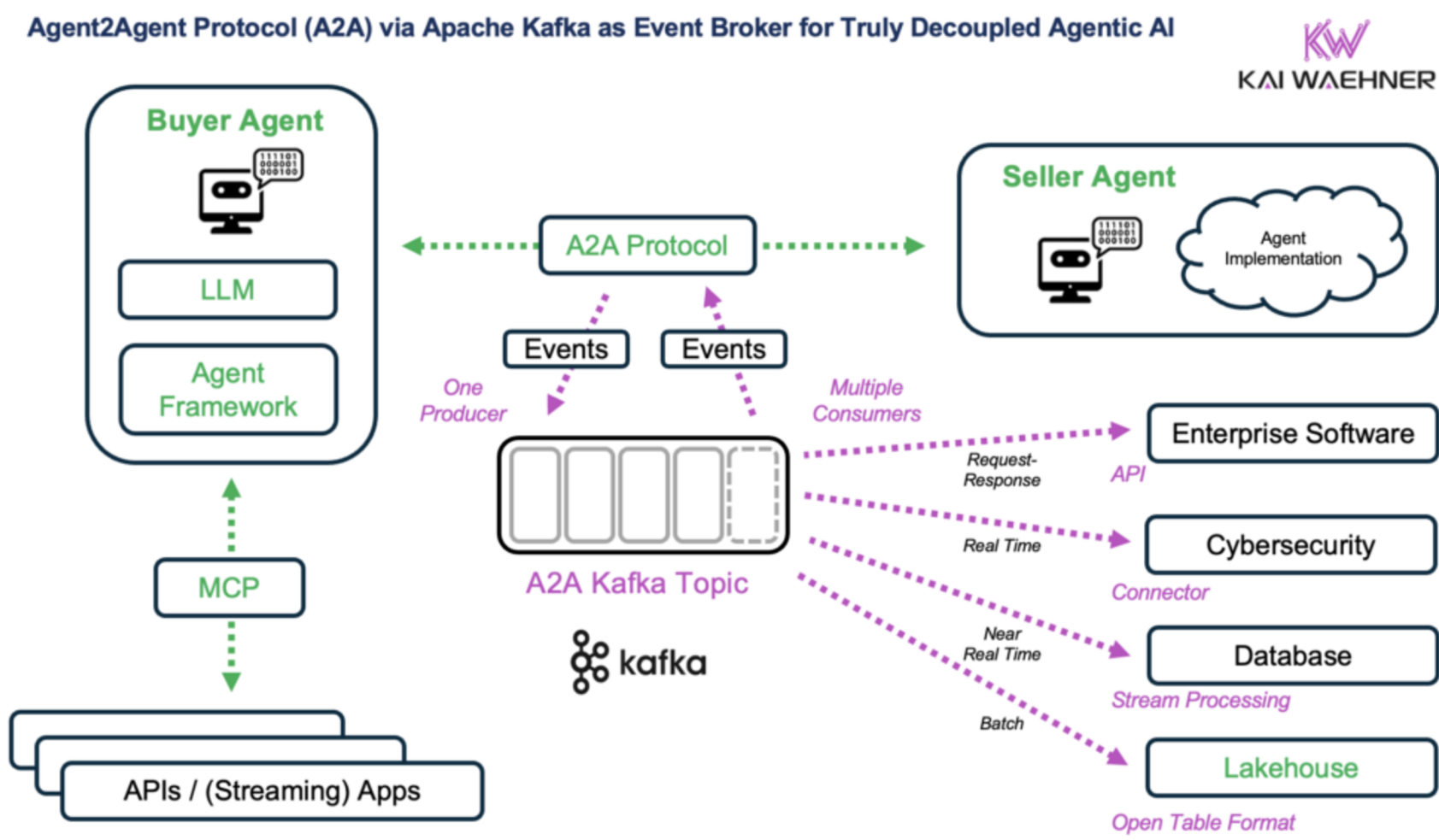


A2A 交互中的核心参与者：

- 用户：最终用户，用户发起请求以获得来自一个或多个 AI 代理的帮助。
- A2A 客户端（客户端代理）：应用程序、服务或其他 AI 代理，代表用户行事。客户端使用 A2A 协议。
- A2A 服务器（远程代理）：公开实现 A2A 协议的 HTTP 端点。它接收来自客户端的消息，处理任务，并返回结果或状态更新。从客户的角度来看，远程代理作为一个不透明（黑匣子）系统运行，这意味着其内部工作原理、内存或工具不会暴露。

Bub it. Build it.



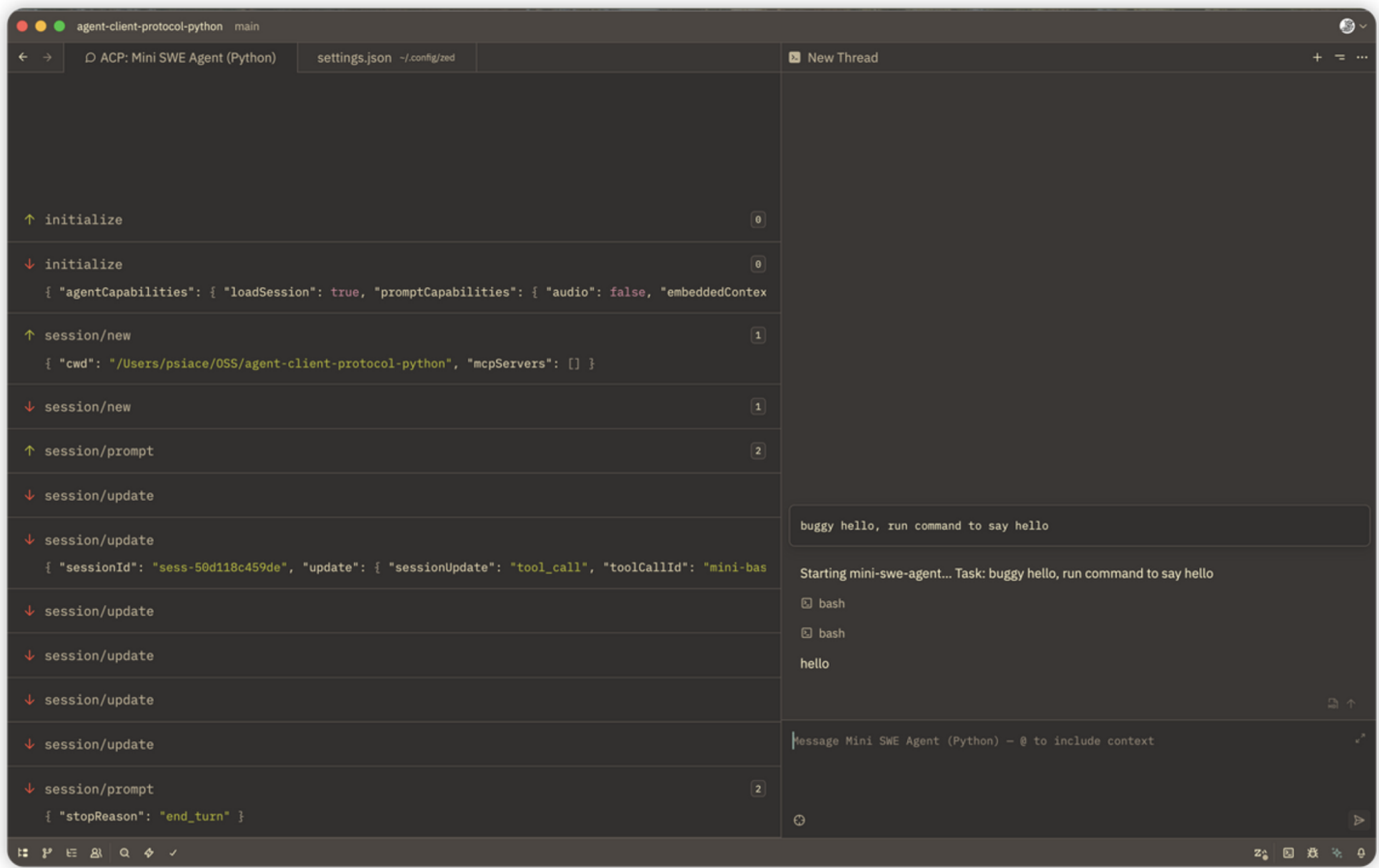


左图展示了一个多 Agent 系统中，如何同时引入事件驱动系统、A2A 和 MCP，这个思路与我们的 Coding Agent 设计是一致的。

- 微服务、数据产品和可重用性 – Agentic AI 只是整个蓝图的一小部分
- Agentic AI 和其能力之间需要解耦，并且充分利用已有的基础设施



交互

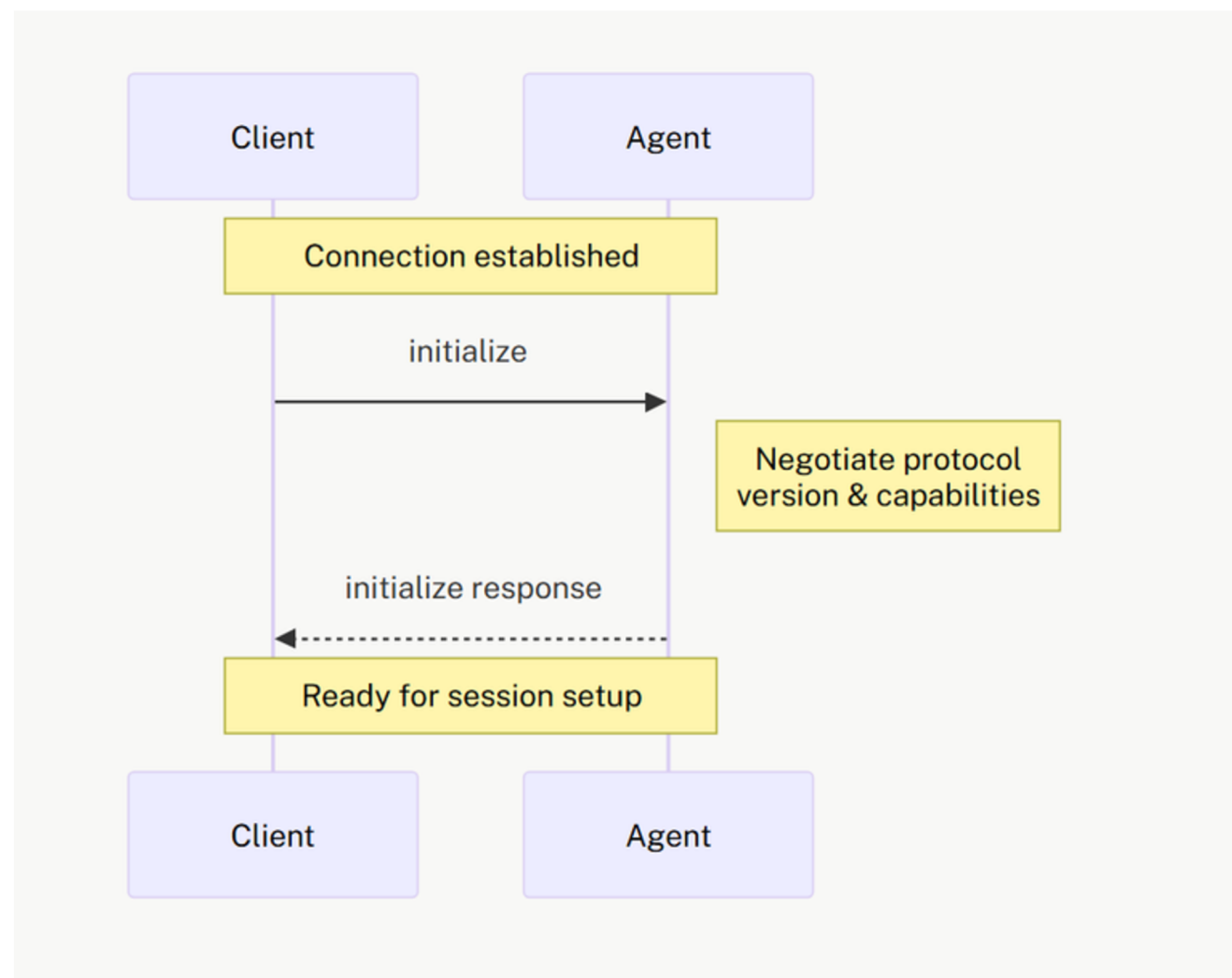


- Zed 的策略是用 ACP 打开“自带/外部 Agent 都可无缝集成”的入口，给用户一致的 UI/交互体验，同时降低 Agent 集成摩擦，巩固编辑器在工作流中的地位。
- ACP（Agent Client Protocol）不仅是“编辑器 ↔ Agent”的对接协议，更是一套对交互与语义的契约，使不同宿主与不同 Agent 之间形成稳定的用户体验与能力映射。
- 这属于 AI 时代的“标准之争”：谁能提出并推广通行的协议，就能在生态里占据主动权。可类比云原生时代的容器与编排标准。

Bub it. Build it.



交互



- 用户角度：可以把任意喜欢的 Agent（Gemini CLI、Claude Code）接入 Zed，获得相似的 UI/交互体验（如流式输出、工具调用、文件跟随、计划展示）。
- 开发者角度：实现一次 ACP，即可进入多个支持 ACP 的宿主，不被单一 SDK 或编辑器锁定，降低分发与适配成本。
- 平台角度：统一交互契约带来更稳定的 UX 预期与更低的支持成本，也强化了宿主在开发工作流中的中心地位。

Bub it. Build it.



交互

```
class EchoAgent(Agent):
    async def initialize(self, params: InitializeRequest) -> InitializeResponse:
        return InitializeResponse(protocolVersion=params.protocolVersion)

    async def newSession(self, params: NewSessionRequest) -> NewSessionResponse:
        return NewSessionResponse(sessionId="sess-1")

    async def loadSession(self, params: LoadSessionRequest) -> None:
        return None

    async def authenticate(self, params: AuthenticateRequest) -> None:
        return None

    async def prompt(self, params: PromptRequest) -> PromptResponse:
        # Normally you'd stream updates via sessionUpdate
        return PromptResponse(stopReason="end_turn")

    async def cancel(self, params: CancelNotification) -> None:
        return None

async def main() -> None:
    reader, writer = await stdio_streams()
    # For an agent process, local writes go to client stdin (writer=stdout)
    AgentSideConnection(lambda _conn: EchoAgent(), writer, reader)
    # Keep running; in a real agent you would await tasks or add your own loop
    await asyncio.Event().wait()
```

第一个被官方收录的社区实现

Libraries

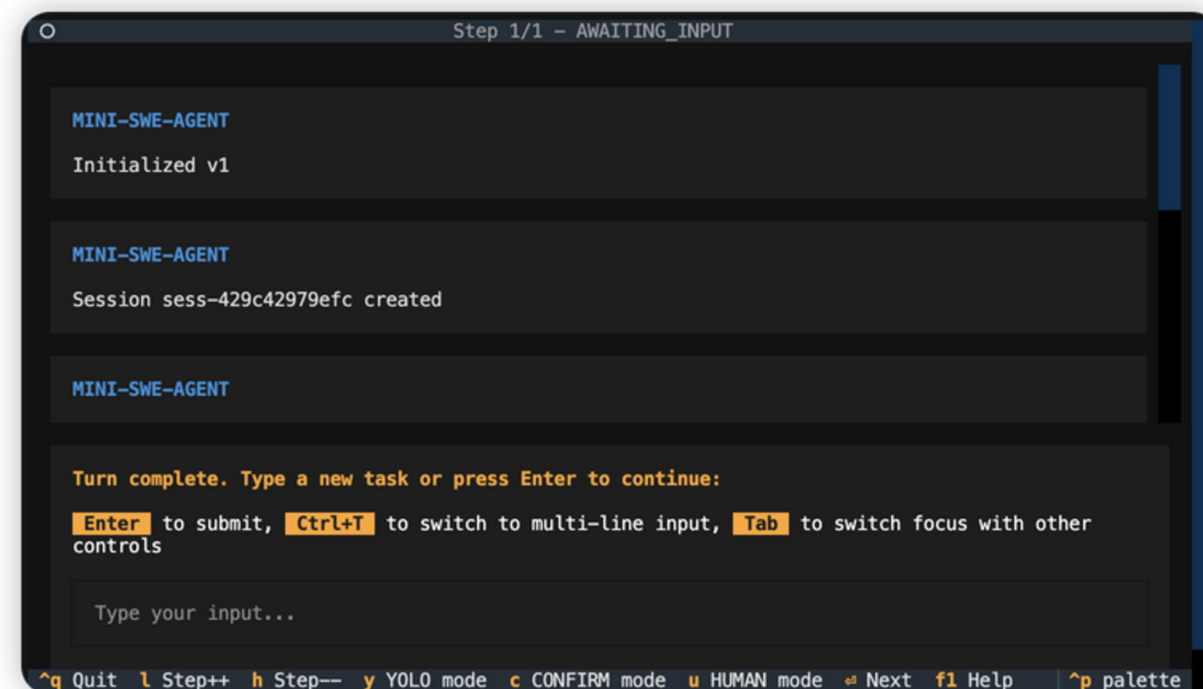
Community

Community managed libraries for the Agent Client Protocol

Copy page

Python

- [agent-client-protocol-python](#)



Bub it. Build it.



结语

Event Type Summary		Event Types	
Event Type	Count		
agent.lifecycle.completed	4		
agent.lifecycle.failed	1		
agent.lifecycle.started	6		
agent.llm.request	4		
agent.llm.response	4		
agent.operation.completed	4		
agent.operation.started	4		
agent.operation.step.completed	2		
agent.operation.step.started	2		
mcp.tool.call	6		
mcp.tool.result	6		
workflow.completed	2		
workflow.started	2		
workflow.step.completed	4		
workflow.step.started	4		

Agents, Tools & Workflows			System Components	
Component Type	Name	Details		
Agent	DeveloperAgent	Role: assistant		
Agent	ReviewerAgent	Role: assistant		
Tool	echo	Calls: 2		
Tool	implement_code	Calls: 2		
Tool	review_code	Calls: 2		
Workflow	4e621bd8...	Type: sequential, Steps: 2		
Workflow	4e621bd8...	Type: sequential, Steps: 2		

- 事件作为基准驱动全局
- 所有 Agent 之间的交互通过 A2A 完成
- 内外部工具定义完全统一在 MCP 之下
- 基于 ACP 提供标准化和语义化的交互定义

Bub it. Build it.



鸣谢



- <https://arxiv.org/html/2310.06770v3>
- <https://arxiv.org/pdf/2405.15793>
- <https://arxiv.org/html/2406.14497v1>
- <https://arxiv.org/html/2402.12317v2>
- <https://github.com/antvis/GPT-Vis>
- <https://www.phontron.com/slides/neubig24softwareagents.pdf>
- <https://ampcode.com/how-to-build-an-agent>
- <https://huggingface.co/blog/tiny-agents>
- <https://blog.langchain.com/context-engineering-for-agents/>
- <https://modelcontextprotocol.io/>
-
- <https://www.kai-waehner.de/blog/2025/05/26/agentic-ai-with-the-agent2agent-protocol-a2a-and-mcp-using-apache-kafka-as-event-broker/>
- <https://agentclientprotocol.com/overview/introduction>
- <https://a2a-protocol.org/latest/>

Bub it. Build it.



联络

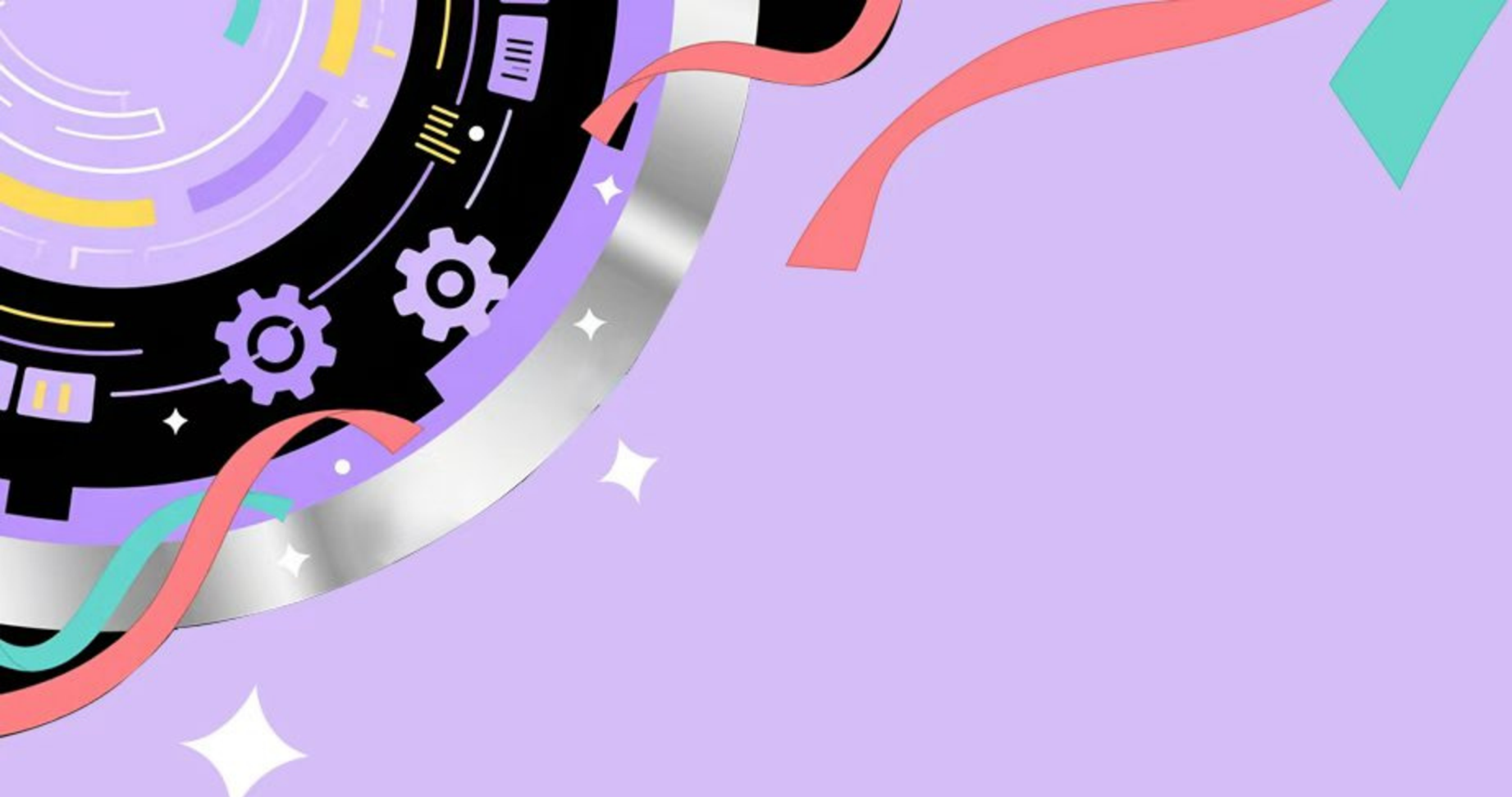


<https://psiace.me>
GitHub: PsiACE
X(Twitter): repsiace



Bub it. Build it.





Thanks!

