Computing Center, IHEP, CAS

National HEP Data Center

AI智能体和MCP的简介及应用

报告人：熊东波

中国科学院高能物理研究所-计算中心

On behalf of HepAI Group

第二十一届全国科学计算与信息化会议暨第十二届科研信息化联盟会议

2025.08.27 中国 长春

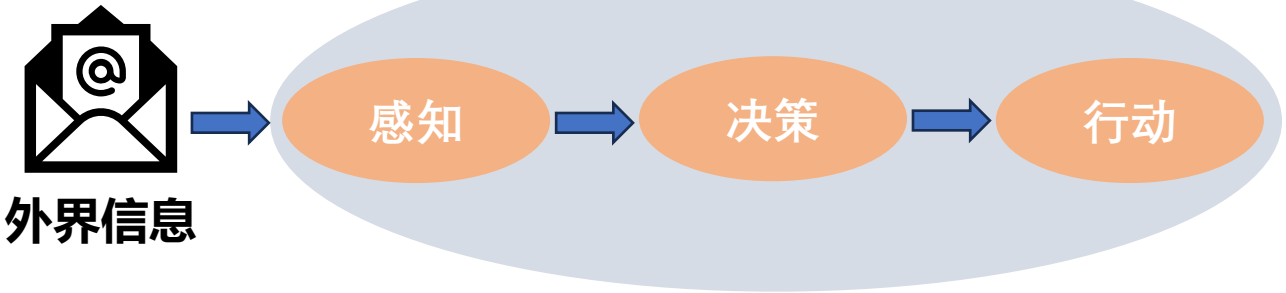
Outline

01-AI智能体简介

1.1.AI智能体简介

符号逻辑 | 响应式 | 强化学习 | 迁移学习和元学习

◆ 智能体的核心：
- ✓ 感知
- ✓ 决策
- ✓ 行动



◆ 传统AI智能体：
- 泛化性差
- 可解释性差
- 协调能力有限

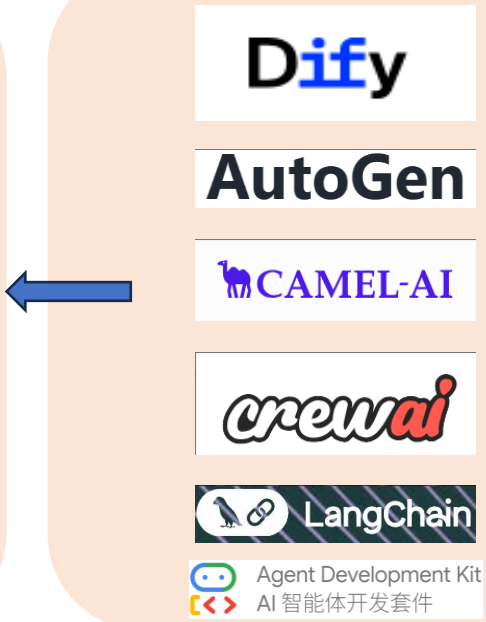
◆ 大模型时代的智能体：
- ✓ 感知
- ✓ 知识与记忆
- ✓ 思考和决策
- ✓ 行动
- ✓ 自主学习
- ✓ …

◆ 基于大模型的智能体：
- ✓ 动态生成任务流程
- ✓ 基于语言等人类可读的方式进行推理和决策
- ✓ 可驱动多 Agent 协作
- ✓ 可根据外部知识进行学习和反馈
- ✓ …

1. https://arxiv.org/abs/2401.03428, 2.https://aima.cs.berkeley.edu, 3.https://arxiv.org/abs/2503.12687

1.2. AI智能体框架及应用

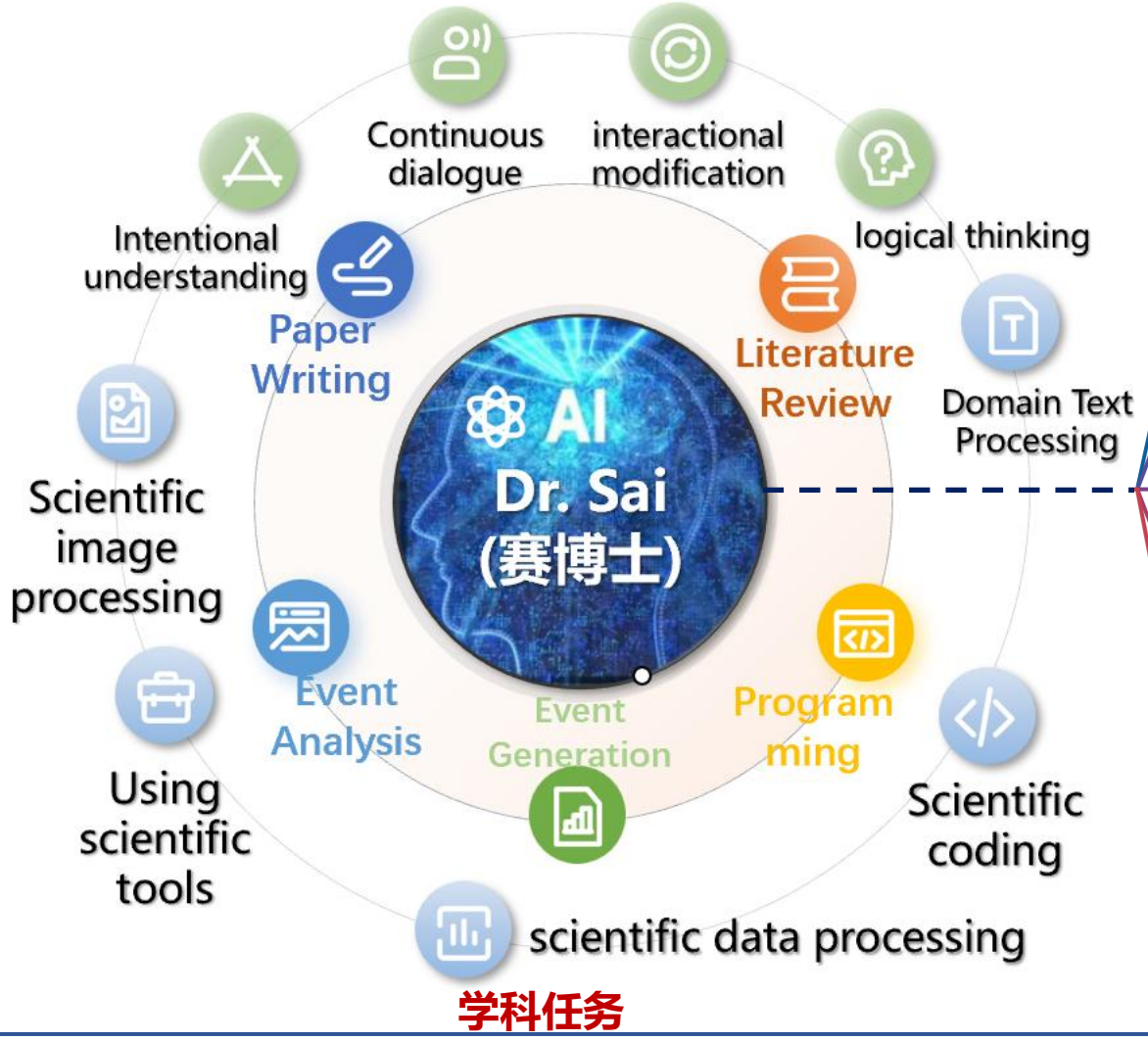
- **通用智能体：**
  - Manus/Magentic-one：多智能体协作
  - OWL/Open Manus： Manus开源竞品
  - Genspark Super Agent：数据库+LLM+Tools的经典智能体
  - DeepResearch：深度信息检索
- **AI智能体框架：**
  - AutoGen：多智能体协作
  - Dify：工作流设置
  - CAMEL AI/CrewAI：分布式协作，角色对话优化
  - LangChain：组件式开发
- **专业（垂直领域）智能体：**
  - Claude-Code、Cursor：可进行规划和代码执行的代码开发助手
  - 机器化学家：智能化学分析与实验
  - Dr.Sai/chATLAS：为物理分析开发的专业智能体

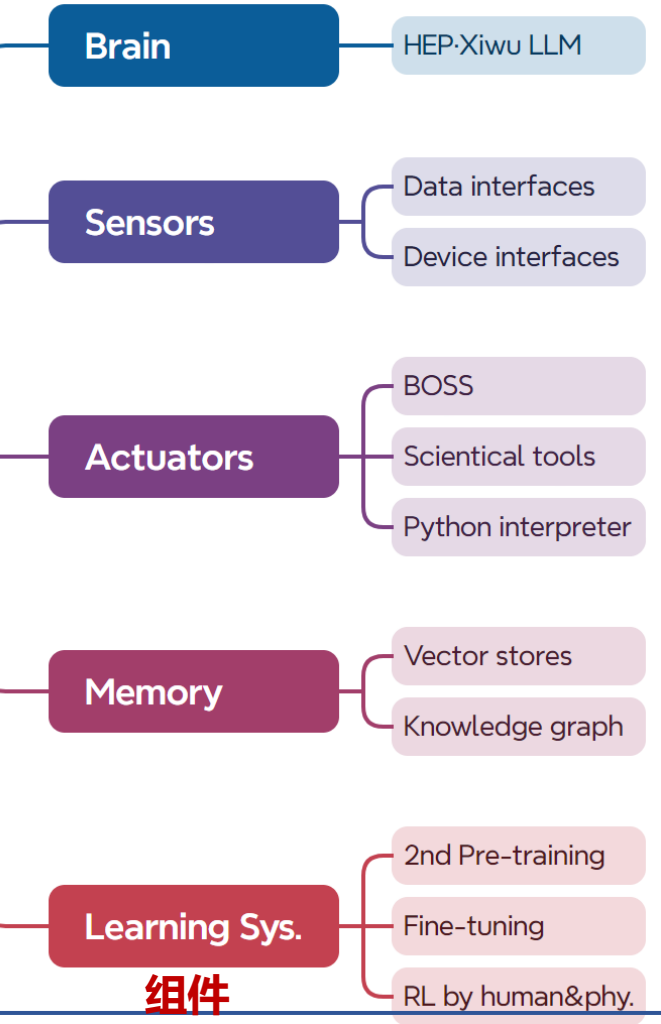


https://www.analyticsvidhya.com/, https://huggingface.co/gaia-benchmark

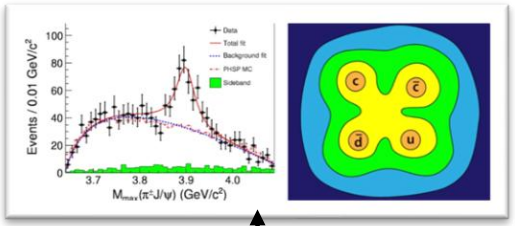
1.3. OpenDrSai 智能体和多智能体框架

◆ **OpenDrSai-为科学数据分析等复杂科学任务而开发的智能体和多智能体系统开发框架**





Dr.Sai BESIII 数据分析

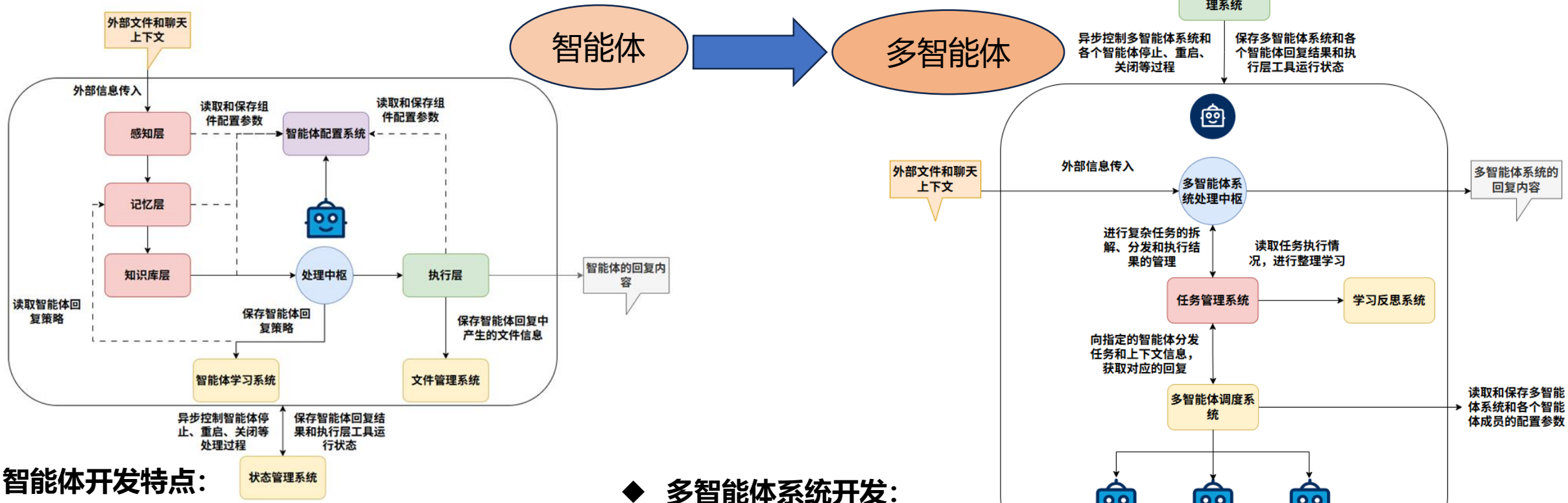


光源类大装置数据分析

1.https://code.ihep.ac.cn/hepai/drsai

1.3. OpenDrSai 智能体和多智能体框架

◆ OpenDrSai 智能体和多智能体系统开发框架



◆ 智能体开发特点：

- ✓ 复杂的科学数据感知
- ✓ 专业知识和记忆管理
- ✓ 专业任务处理过程学习
- ✓ 状态控制和过程文件管理

◆ 多智能体系统开发：

- ✓ 复杂任务管理
- ✓ 复杂多任务处理过程学习
- ✓ 多智能体系统状态控制

https://code.ihep.ac.cn/hepai/drsai

1.3. OpenDrSai 智能体和多智能体框架及应用

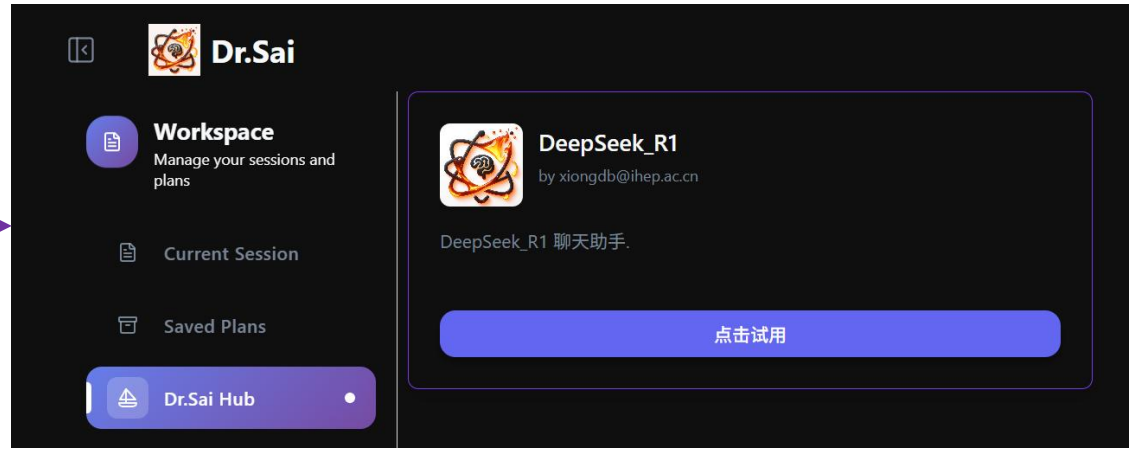
◆ **OpenDrSai 支持开发部署一体化（Power by DDF2）**

```python
def create_agent() -> AssistantAgent:

    # Define a model client. You can use other model client that implements
    # the `ChatCompletionClient` interface.
    model_client = HepAIChatCompletionClient(
        model="deepseek-ai/deepseek-r1:671b",
        api_key=os.environ.get("HEPAI_API_KEY"),
        base_url="https://aiapi.ihep.ac.cn/apiv2",
    )

    # Define an AssistantAgent with the model, tool, system message, and reflection enabled.
    # The system message instructs the agent via natural language.
    return AssistantAgent(
        name="weather_agent",
        model_client=model_client,
        system_message="You are a helpful assistant.",
        reflect_on_tool_use=False,
        model_client_stream=True,  # Enable streaming tokens from the model client.
    )
```

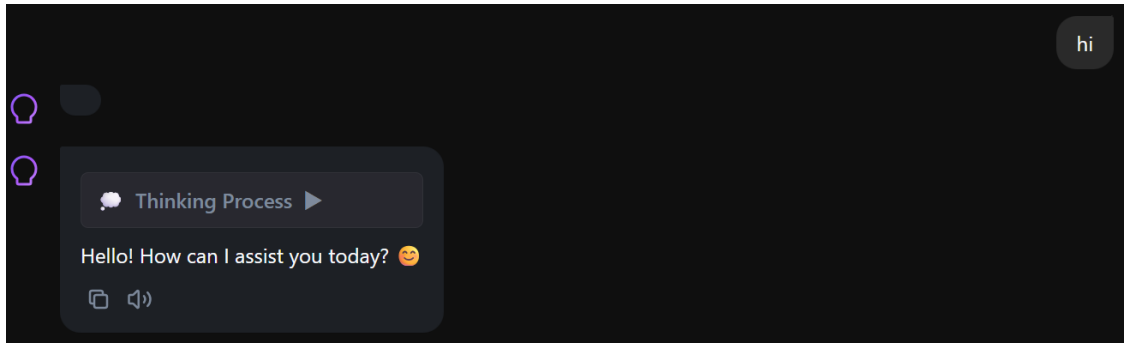
简单定制

一键部署

```python
asyncio.run(
    run_worker(
        # 智能体注册信息
        agent_name="DeepSeek_R1",
        permission='groups: drsai; users: admin, xiongdb@ihep.ac.cn
        description = "DeepSeek_R1 聊天助手.",
        version = "0.1.0",
        logo="https://aiapi.ihep.ac.cn/apiv2/files/file-8572b27d093
        # 智能体实体
        agent_factory=create_agent,
        # 后端服务配置
        port = 42812,
        no_register=False,
        enable_openwebui_pipeline=True,
        history_mode = "backend",
        # use_api_key_mode = "backend",
    )
)
```



开发即应用



https://code.ihep.ac.cn/hepai/drsai

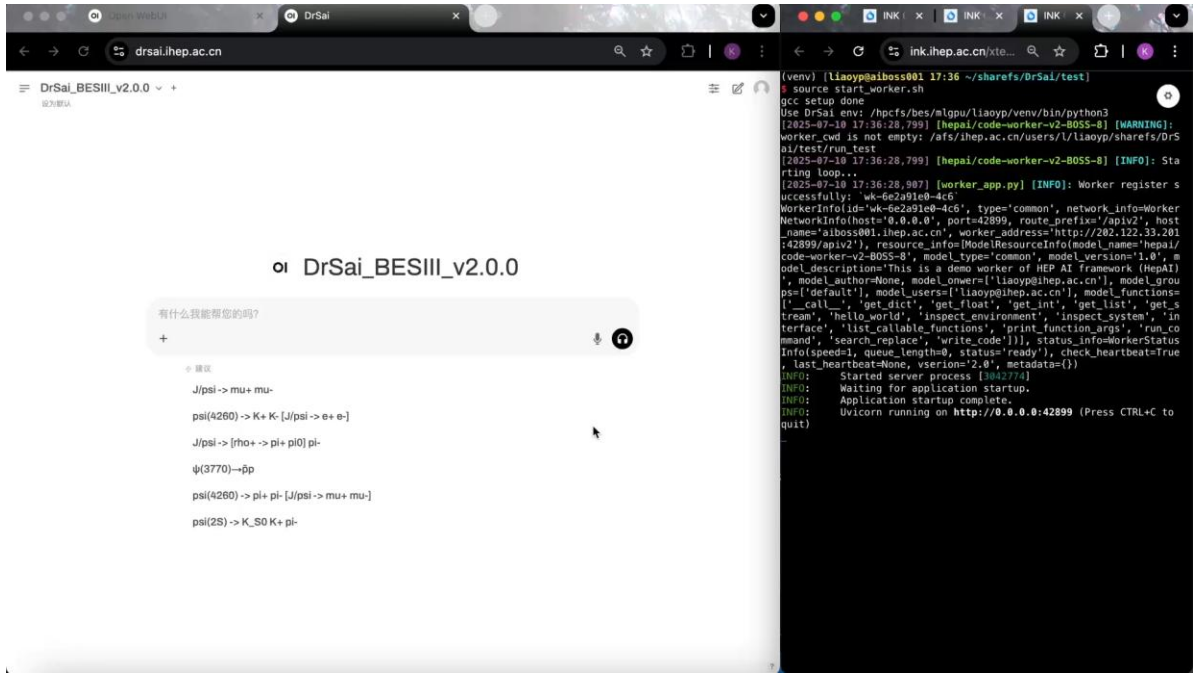
1.3. OpenDrSai 智能体和多智能体框架及应用

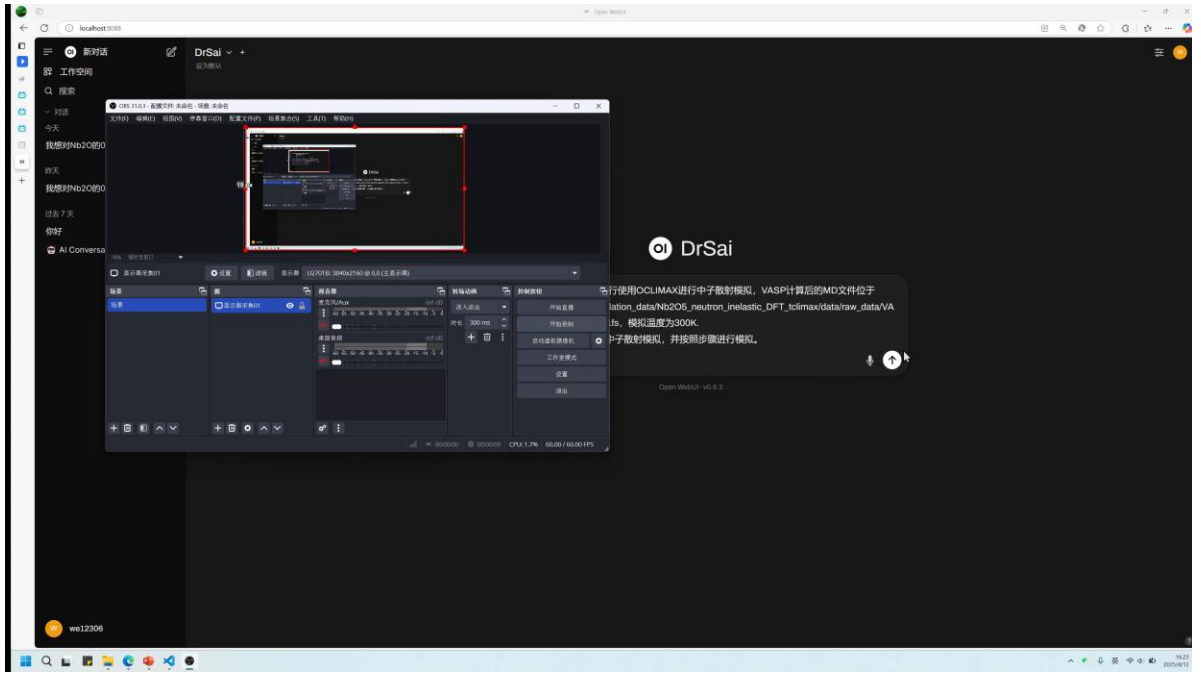
- **BESIII物理分析：**

  通过Planner、Host、Tester、Coder等多智能体进行BESIII数据分析

- **光源类大科学装置数据分析：**

  通过构建不同方法学专业数据处理智能体和管理智能体进行多方法科学数据处理





1. https://code.ihep.ac.cn/hepai/drsai

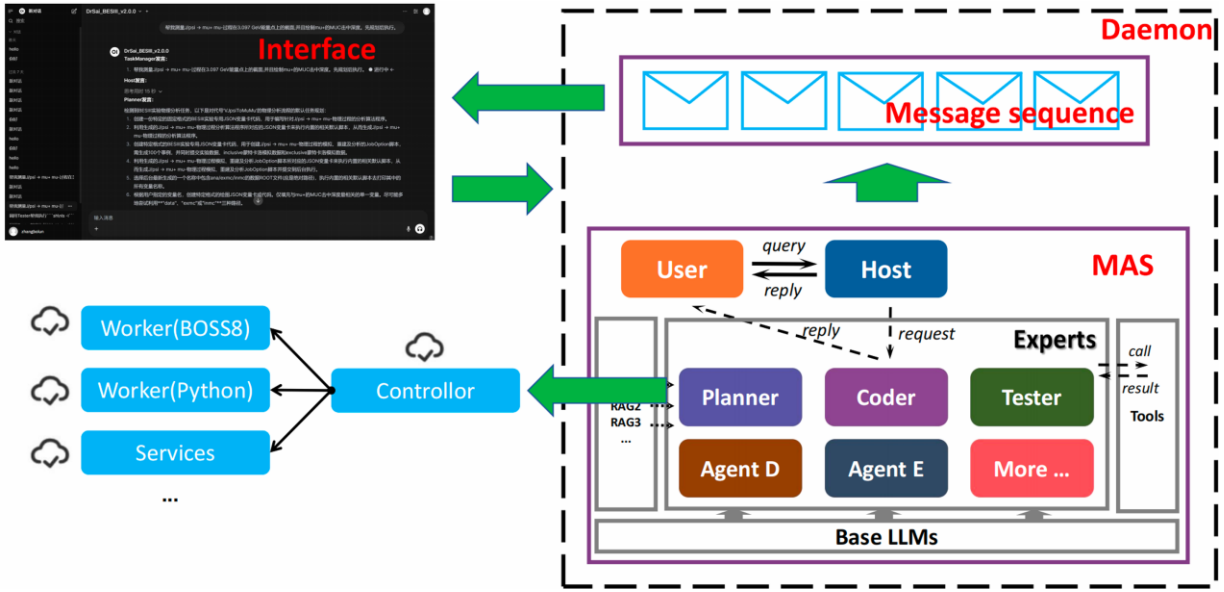
1.3. OpenDrSai 智能体和多智能体框架及应用

- **BESIII物理分析智能体：**

  特点：分析方法和步骤明确

  核心：

  - **树状任务系统，Host任务指派**
  - **分布式使用专业工具**
  - **长任务守护与状态管理**

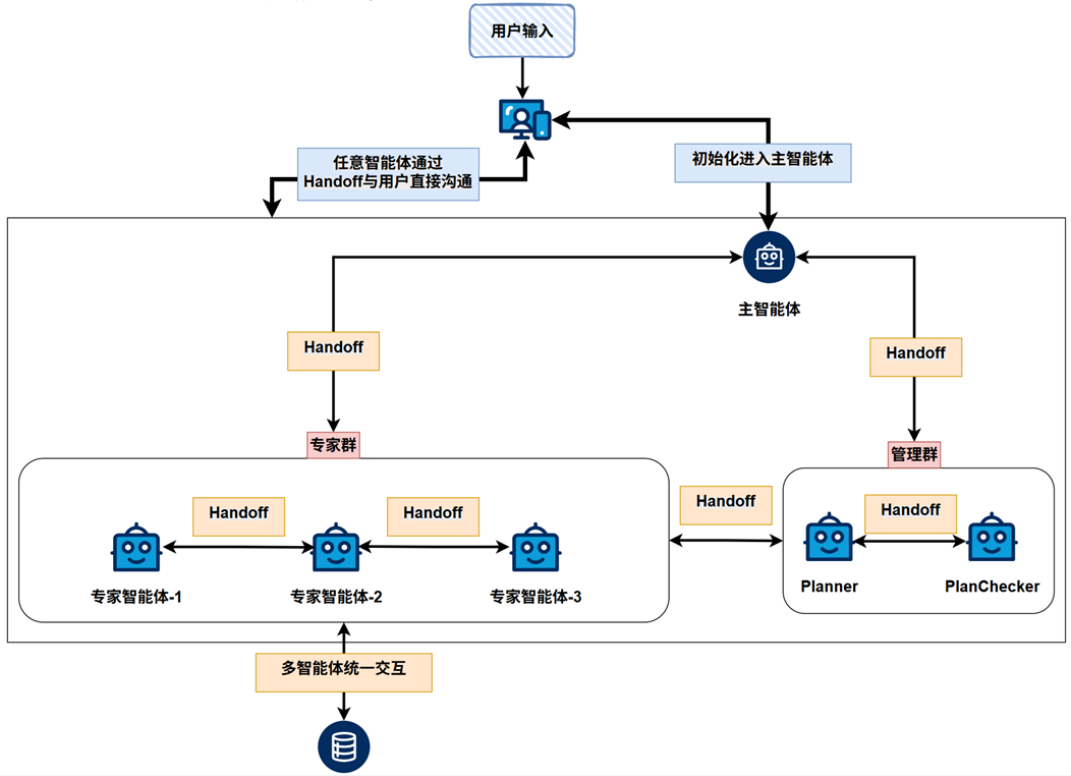


- **光源类大科学装置数据分析智能体：**

  特点：任务和分析方法多样

  核心：

  - **Handoff任务转移机制**
  - **可扩展的多智能体系统**

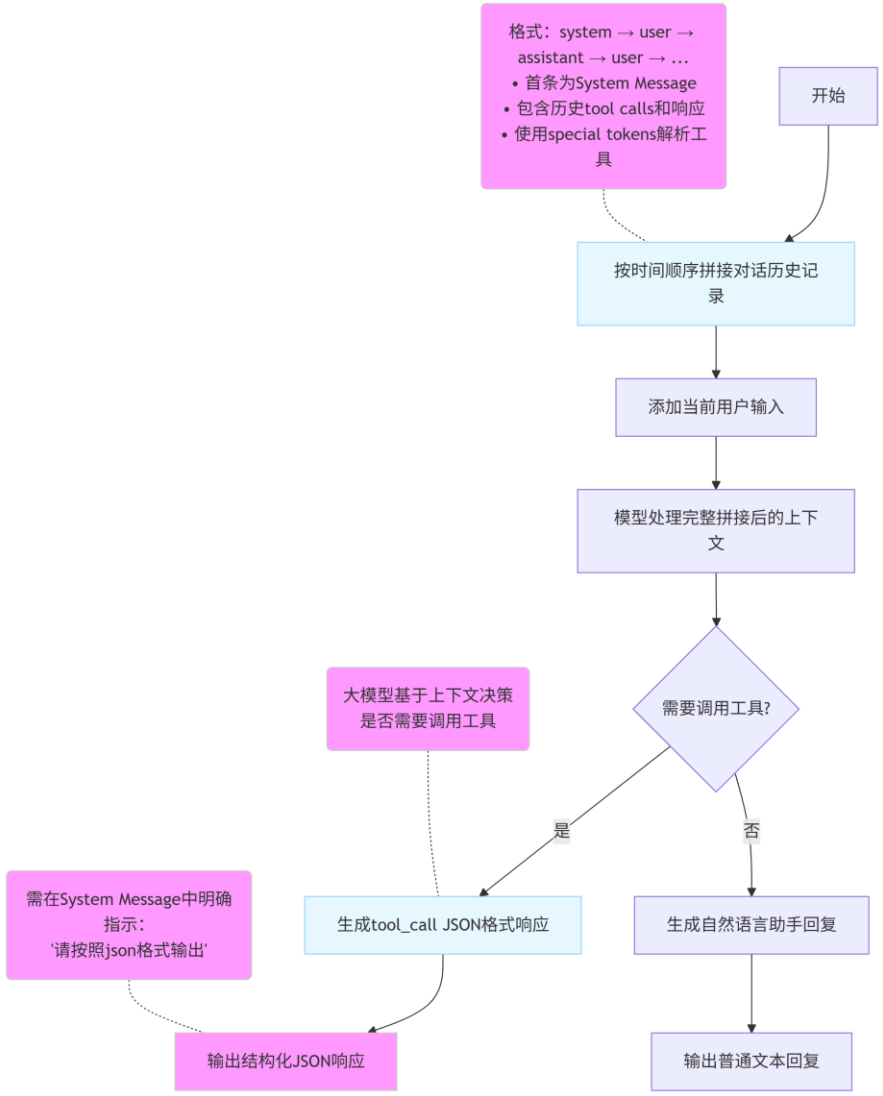


1. https://code.ihep.ac.cn/hepai/drsai

02-MCP简介

2.1 MCP (Model Context Protocol)

◆ 大模型进行tool call调用的过程：



◆ 问题：

- 不同API厂商的tool call构造格式差异较大

OpenAI

```python
from openai import OpenAI
client = OpenAI()

resp = client.chat.completions.create(
    model="gpt-4o",
    messages=[
        {"role": "user", "content": "查一下明天新加坡的天气"}
    ],
    tools=[
        {
            "type": "function",
            "function": {
                "name": "get_weather",
                "description": "获取某个城市的天气",
                "parameters": {
                    "type": "object",
                    "properties": {
                        "city": {"type": "string"},
                        "date": {"type": "string"}
                    },
                    "required": ["city", "date"]
                }
            }
        }
    ]
)
print(resp.choices[0].message.tool_calls)
```

Anthropic

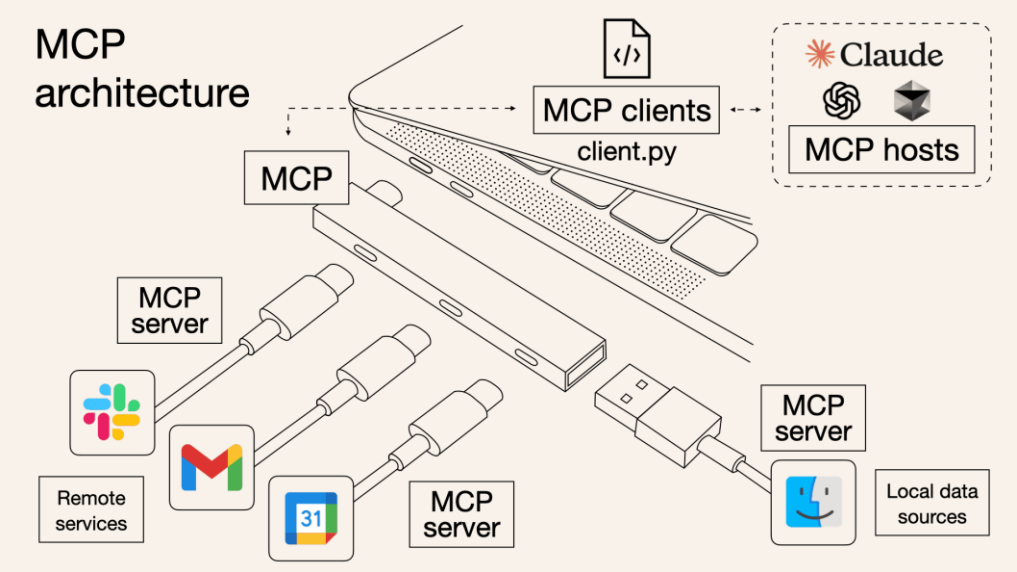
```
# Anthropic
{"tools": [
      {
        "name": "get_stock_price",
        "description": "Get the current stock price for a given ticker symbol.",
        "input_schema": {
          "type": "object",
          "properties": {
            "ticker": {
              "type": "string",
              "description": "The stock ticker symbol, e.g. AAPL for Apple Inc."
            }
          },
          "required": ["ticker"]
        }
      }
    ]
}
```

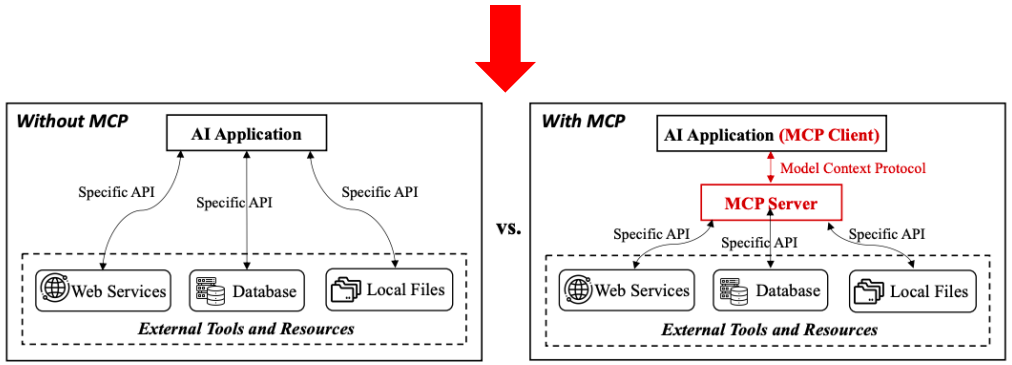
Gemini

```
# Gemini
{tools": [
      {
        "functionDeclarations": [
          {
            "name": "get_current_temperature",
            "description": "Gets the current temperature for a given location.",
            "parameters": {
              "type": "object",
              "properties": {
                "location": {
                  "type": "string",
                  "description": "The city name, e.g. San Francisco"
                }
              },
              "required": [
                "location"
              ]
            }
          }
        ]
      }
    ]
}
```

2.1 MCP (Model Context Protocol)

- **MCP：AI模型与外部工具和资源之间交互的统一协议**

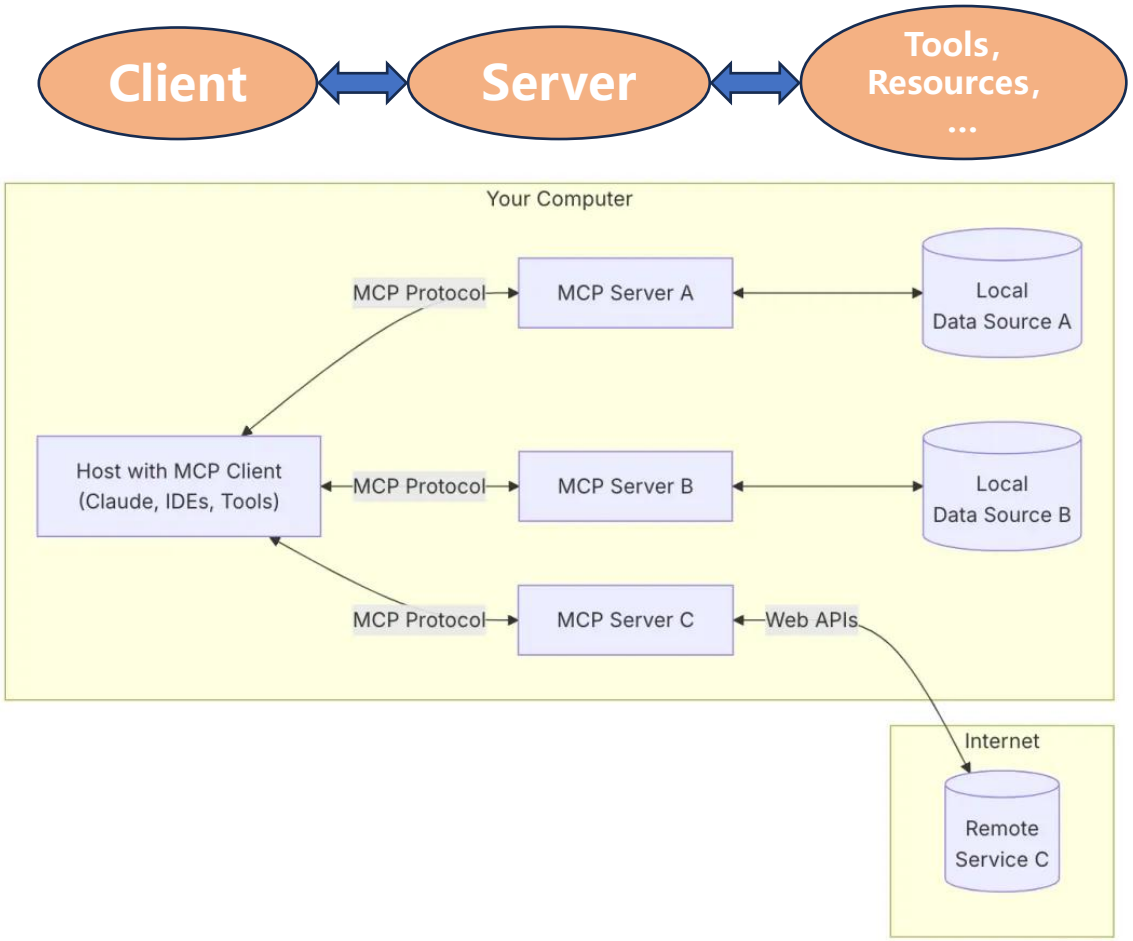




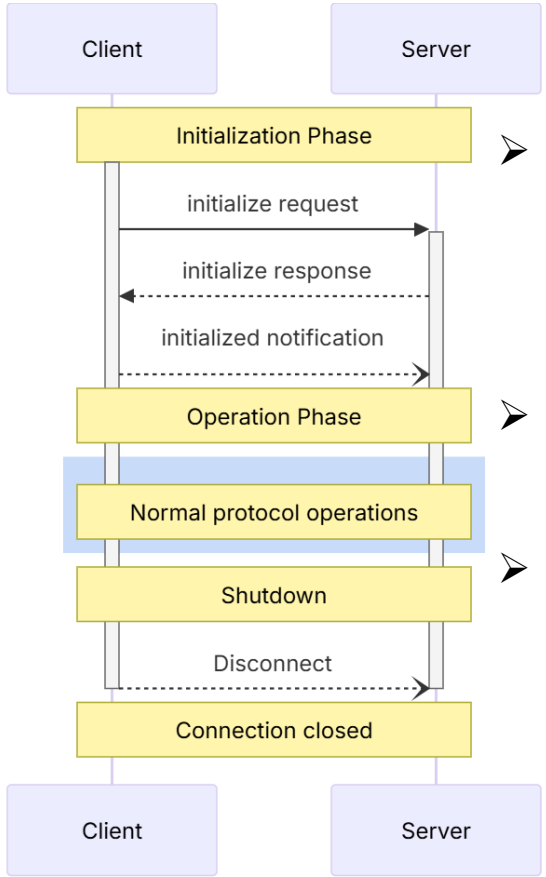
- **MCP解决的问题**
  - ✓ **1.不同API厂商和智能体框架兼容统一的工具和资源接入方式**
  - ✓ **2.有利于形成跨模型、跨平台的生态，促进更多第三方工具/服务的加入**

2.2 MCP的工作逻辑

◆ MCP的工作逻辑



◆ MCP生命周期：



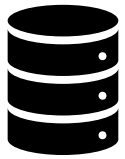
- **Initialization（初始化阶段）：能力协商与协议版本一致性确认**
- **Operation（运行阶段）：常规协议通信**
- **Shutdown（关闭阶段）：连接的优雅关闭**

1. https://arxiv.org/abs/2503.23278, 2.https://modelcontextprotocol.io/docs, 3.https://mcpcn.com

2.3 MCP的开发应用

◆ MCP服务开发的核心概念：

资源（Resources）



类似 REST API 的 GET 接口，只读、无副作用。常用于暴露文件内容、配置信息、数据库查询结果、外部 API 返回值等，供 LLM 获取上下文。

工具（Tools）

类似 POST 接口，可执行逻辑。适用于执行计算、调用第三方 API、数据写入、自动化任务等。支持结构化输出、参数校验、进度回报、与用户的动态交互

提示模板（Prompts）

可复用的交互模板，帮助标准化生成文本、消息或代码片段。可注册为命令或菜单选项，LLM 调用时自动填充。

◆ MCP服务开发的中的其他内容：

图片（Images）

参数补全（Completions）

日志与通知（Logging & Notifications）

认证与权限（Authentication & Authorization）

1.https://modelcontextprotocol.io/docs

2.4 如何构建MCP服务

- **MCP服务构建和应用的逻辑**
  - **MCP服务开发者开发Resources/Tools/Prompts等具体的服务应用**
  - **在python端可以通过简单修饰器启动MCP Server，可以直接执行程序，或者启动为streamable-http/sse服务**
  - **各个API和智能体框架支持直接接入MCP Clients，作为大模型tool call等**

AutoGen:

```
playwright_server_params = SseServerParams(
    url="http://localhost:8931/sse",
)
```

OpenAI:

```
from openai import OpenAI

client = OpenAI()

resp = client.responses.create(
    model="gpt-4.1",
    tools=[
        {
            "type": "mcp",
            "server_label": "deepwiki",
            "server_url": "https://mcp.deepwiki.com/mcp",
            "require_approval": "never",
        },
    ],
    input="What transport protocols are supported in the 2
)
```

```
from mcp.server.fastmcp import FastMCP

# Create an MCP server
mcp = FastMCP("Demo")

# Add an addition tool
@mcp.tool()
def add(a: int, b: int) -> int:
    """Add two numbers"""
    return a + b


# Add a dynamic greeting resource
@mcp.resource("greeting://{name}")
def get_greeting(name: str) -> str:
    """Get a personalized greeting"""
    return f"Hello, {name}!"


# Add a prompt
@mcp.prompt()
def greet_user(name: str, style: str = "friendly") -> str:
    """Generate a greeting prompt"""
    styles = {
        "friendly": "Please write a warm, friendly greeting",
        "formal": "Please write a formal, professional greeting",
        "casual": "Please write a casual, relaxed greeting",
    }

    return f"{styles.get(style, styles['friendly'])} for someone named {name}."
```

服务描述

Tools

Resources

Prompt

2.4 如何构建MCP服务

- **MCP服务其他开发内容**
  - 图像(Image)
  - 鉴权(Authentication)
  - 结构化输出(Structured Output)
  - 采样(Sampling)
  - 日志与通知(Logging and Notifications)

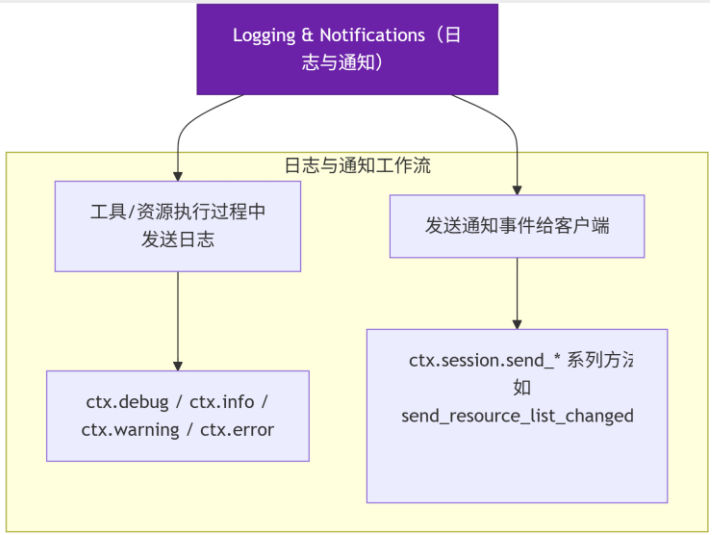
基于 OAuth 2.1 的令牌鉴权机制：

```python
class SimpleTokenVerifier(TokenVerifier):
    """Simple token verifier for demonstration."""

    async def verify_token(self, token: str) -> AccessToken | None:
        pass  # This is where you would implement actual token validation


# Create FastMCP instance as a Resource Server
mcp = FastMCP(
    "Weather Service",
    # Token verifier for authentication
    token_verifier=SimpleTokenVerifier(),
    # Auth settings for RFC 9728 Protected Resource Metadata
    auth=AuthSettings(
        issuer_url=AnyHttpUrl("https://auth.example.com"),  # Authorization Server URL
        resource_server_url=AnyHttpUrl("http://localhost:3001"),  # This server's URL
        required_scopes=["user"],
    ),
)
```

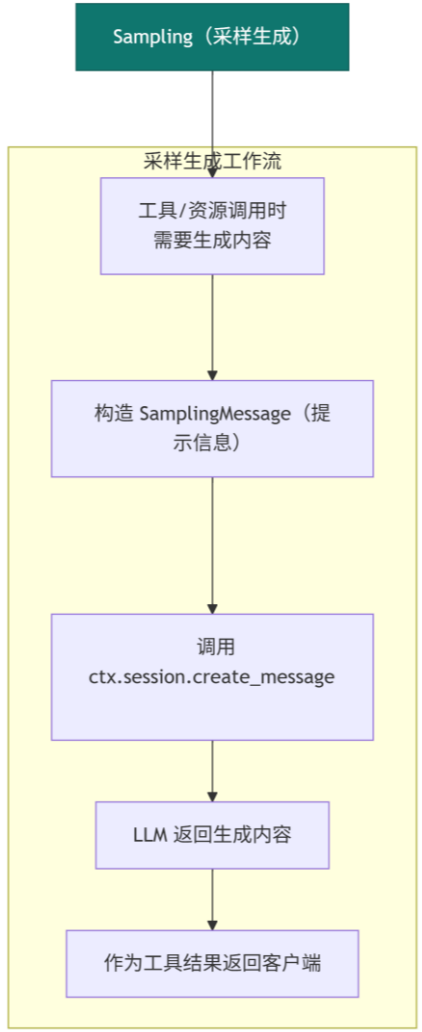
处理Image对象：

```python
from PIL import Image as PILImage

from mcp.server.fastmcp import FastMCP, Image

mcp = FastMCP("Image Example")


@mcp.tool()
def create_thumbnail(image_path: str) -> Image:
    """Create a thumbnail from an image"""
    img = PILImage.open(image_path)
    img.thumbnail((100, 100))
    return Image(data=img.tobytes(), format="png")
```

日志与通知：



采样：通过客户端向大语言模型请求文本补全



1. https://github.com/modelcontextprotocol/

2.5 如何启动MCP服务

- **MCP服务三种运行方式**
  - **stdio**：使用标准输入输出（Standard Input/Output）进行通信，客户端和服务器直接通过进程管道交换 JSON 消息
  - **sse**：基于 HTTP 长连接的单向推送协议，服务器可以实时推送事件给客户端
  - **streamable-http**：基于 HTTP 的全双工流式通信，支持有状态 / 无状态模式，可同时发送和接收流式数据

HTTP服务启动：

- **定义启动端口，通过/sse或者/mcp访问**
- **使用Starlette进行HTTP 层路由、挂载、生命周期管理**

```python
# Create an MCP server
mcp = FastMCP(
    name="Demo",
    instructions="Welcome to the demo server!",
    host="localhost",
    port=42996,
    )
```

```
INFO:     Started server process [3054314]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://localhost:42996 (Press CTRL+C to quit)
```

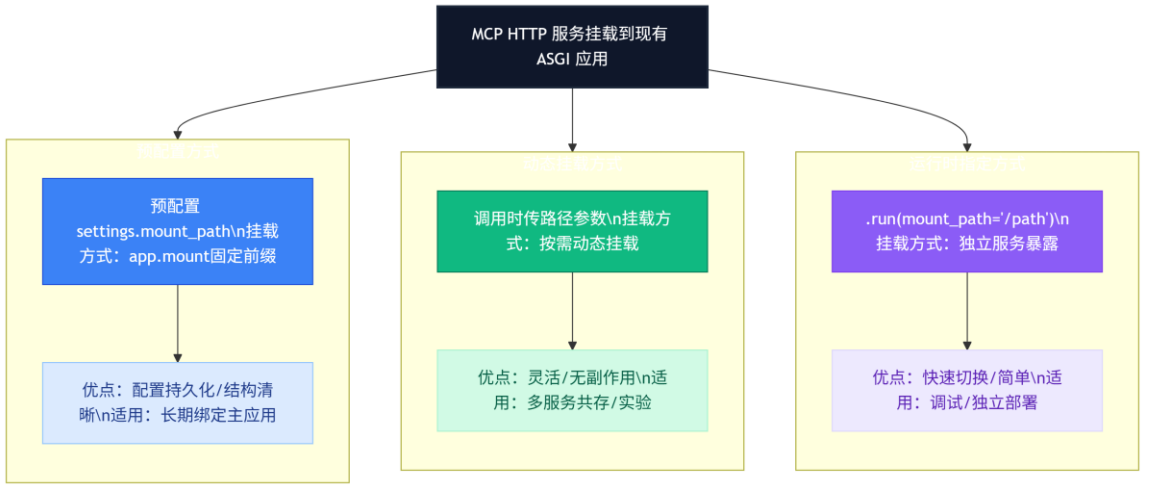
```python
from mcp.server.fastmcp import FastMCP

mcp = FastMCP("My App")


@mcp.tool()
def hello(name: str = "World") -> str:
    """Say hello to someone."""
    return f"Hello, {name}!"


def main():
    """Entry point for the direct execution server."""
    mcp.run()


if __name__ == "__main__":
    main()
```

```python
# Add a simple tool to demonstrate the server
@mcp.tool()
def greet(name: str = "World") -> str:
    """Greet someone by name."""
    return f"Hello, {name}!"


# Run server with streamable_http transport
if __name__ == "__main__":
    mcp.run(transport="streamable-http")
```

直接执行

HTTP服务

挂载到现有ASGI服务器：



1. https://github.com/modelcontextprotocol/

2.6 如何构建MCP Client

- MCP sse/streamable服务的获取：

```python
import asyncio

from mcp import ClientSession
from mcp.client.streamable_http import streamablehttp_client

async def main():
    # Connect to a streamable HTTP server
    async with streamablehttp_client("http://localhost:42996/mcp") as (
        read_stream,
        write_stream,
        _,
    ):
        # Create a session using the client streams
        async with ClientSession(read_stream, write_stream) as session:
            # Initialize the connection
            await session.initialize()
            # List available tools
            tools = await session.list_tools()
            print(f"Available tools: {[tool.name for tool in tools.tools]}")
```

通过URL获取

```python
import asyncio

from mcp import ClientSession
from mcp.client.sse import sse_client
# from mcp.client.streamable_http import streamablehttp_client

async def main():
    # Connect to a streamable HTTP server
    async with sse_client("http://localhost:42996/sse") as (
        read_stream,
        write_stream
    ):
        # Create a session using the client streams
        async with ClientSession(read_stream, write_stream) as session:
            # Initialize the connection
            await session.initialize()
            # List available tools
            tools = await session.list_tools()
            print(f"Available tools: {[tool.name for tool in tools.tools]}")
```

- MCP应用执行
  - get_prompt
  - call_tool
  - read_resource
  - complete

- MCP stdio服务的获取：

通过执行MCP server文件获取

```python
import asyncio
import os
from pydantic import AnyUrl
from mcp import ClientSession, StdioServerParameters, types
from mcp.client.stdio import stdio_client
from mcp.shared.context import RequestContext

# Create server parameters for stdio connection
server_params = StdioServerParameters(
    command="conda",  # Using uv to run the server
    args=["run","-n","drsai","--live-stream","python", "examples/tools/MCP/mcp_sever01.py"
          ],  # Use relative path since we're in the same directory
    env=None,
)

# Optional: create a sampling callback
async def handle_sampling_message(
    context: RequestContext[ClientSession, None], params: types.CreateMessageRequestParams
) -> types.CreateMessageResult:
    print(f"Sampling request: {params.messages}")
    return types.CreateMessageResult(
        role="assistant",
        content=types.TextContent(
            type="text",
            text="Hello, world! from model",
        ),
        model="gpt-3.5-turbo",
        stopReason="endTurn",
    )


async def run():
    async with stdio_client(server_params) as (read, write):
        async with ClientSession(read, write, sampling_callback=handle_sampling_message) as sessio
            # Initialize the connection
            await session.initialize()

            # List available prompts
            prompts = await session.list_prompts()
            print(f"Available prompts: {[p.name for p in prompts.prompts]}")
```

03-基于MCP的AI智能体开发

3.1 构建自己的MCP应用

◆ 开发一个arXiv文献查询助手

具体见：https://code.ihep.ac.cn/xdb/openagents

1.为MCP服务定义简介和启动端口

```python
# 访问arXiv所需要的包
import requests
import re
# 构建MCP对象
from mcp.server.fastmcp import FastMCP
mcp=FastMCP(
    name="tools",
    instructions="A collection example of tools for MCP.",
    host="0.0.0.0",
    port="42700"
    )
```

3.一键启动（或者使用stdio模式执行）

```python
if __name__ == "__main__":
    # mcp.run(transport="stdio")
    mcp.run(transport="sse")
```

2.写一个包含完整参数描述和类型注释的功能函数

```python
# 注册MCP工具
@mcp.tool()
async def get_arxiv_home(
    search_key: str,
    max_results: int = 10,
    ) -> str:
    """
    根据关键字获取arxiv首页
    params:
        search_key: ArXiv搜索逻辑词，只接受英文字符，如果是中文也需要翻译成英文再搜索

    search_key构造规则：
        - 字段前缀：在查询中，使用字段前缀来指定搜索的字段。例如，ti：表示标题，au：表示作者，abs：表示摘要。
        - 逻辑运算符：使用布尔运算符（AND、OR、ANDNOT）来构建逻辑关系。请注意，AND 和 OR 是大小写敏感的，必须使用大写字母。
        - URL 编码：在构造查询字符串时，确保对特殊字符进行 URL 编码。例如，空格编码为 +，冒号编码为 %3A，括号编码为 %28（左括号）和 %29（右括号）。
        - 括号使用：在使用多个逻辑运算符时，使用括号来明确运算顺序。
    例如：用户需要搜索标题包含"AI Agent"和"material"的论文：search_key = "ti%3AI+Agent+AND+ti%3Amaterial"

    """

    key=search_key.replace(' ','+')
    base_url=f"https://export.arxiv.org/api/query?search_query={key}&start=0&max_results={max_results}"
    results: list[dict] = await async_requests_arxiv_home(base_url)
    result='\n'.join([f"**Title:{i['title']}**\n-----------\n**Abstract**:{i['abstract']}\n**URL**:{i['base_url']}\n" for i in results])
    return result
```

3.2 构建智能体

◆ 模块化定制智能体-基于OpenDrSai框架

1.导入必要的依赖包

```python
# -*- coding: utf-8 -*-
import os, asyncio

# Agent framework
from drsai import AssistantAgent

# Model client
from drsai import HepAIChatCompletionClient

# MCP tools
from autogen_ext.tools.mcp import StdioServerParams, mcp_server_tools, SseServerParams

# backend thread
from drsai import run_worker, run_console


# get path
there = os.path.dirname(os.path.abspath(__file__))
```

3.将创建的智能体应用注册到AI平台

```python
asyncio.run(
    run_worker(
        # 智能体注册信息
        agent_name="ArXiv研究助手",
        # permission='groups: payg; users: admin, xiongdb@ihep.ac.cn, ddf_free, yqsun@ihep.ac.cn; o
        permission={
            "groups": "payg",
            "users": ["admin", "ddf_free", "xiongdb@ihep.ac.cn", "yqsun@ihep.ac.cn"],
            "owner": "11192333767@qq.com"
            },
        description = "一个可调用arXiv的AI助手。",
        version = "0.1.0",
        logo="https://aiapi.ihep.ac.cn/apiv2/files/file-8572b27d093f4e15913bebfac3645e20/preview",
        # 智能体实体
        agent_factory=create_agent,
        # 后端服务配置
        port = 42813,
        no_register=False,
        enable_openwebui_pipeline=True,
        history_mode = "backend",
        # use_api_key_mode = "backend",
    )
)
```

或者在本地调试/OpenAI Chat格式

```python
# 启动一个控制台，你可以输入指令进行交互
# asyncio.run(run_console(agent_factory=create_agent, task="我需要调研关于AI Agent在材料领域的应用?"))
# 符合OpenAI格式访问后端和兼容OpenWebui前端的启动方式，这将会启动在42813端口，访问url为http://0.0.0.0:42813/apiv2
# 你可以直接使用OpenAI进行访问
```

2.模块化定制智能体

2.1.基座模型

2.2.MCP工具获取

2.3.组装智能体

```python
# 创建一个工厂函数，用于并发访问时确保后端使用的Agent实例是隔离的。
async def create_agent() -> AssistantAgent:

    # 定义一个大模型的访问client
    model_client = HepAIChatCompletionClient(
        model=os.environ.get("HEPAI_MODEL", "openai/gpt-4o"),
        api_key=os.environ.get("HEPAI_API_KEY"),
        base_url=os.environ.get("HEPAI_API_URL", "https://aiapi.ihep.ac.cn/apiv2"),
        )

    # 获取MCP工具
    MCP_tools=[]
    ## 本地运行MCP工具
    # MCP_tools.extend(await mcp_server_tools(
    #             StdioServerParams(
    #             command="conda",
    #             args=["run", "-n", "mcps", "--live-stream", "python", f"{there}/arxiv_mcp_tool.py"],
    #             env=None,
    #             )))

    MCP_tools.extend(await mcp_server_tools(
                SseServerParams(
                    url=f"http://localhost:42700/sse",
                    timeout=30,
                )))

    # 定义一个智能体，将大模型访问client和工具赋给智能体
    return AssistantAgent(
        name="ArXiv_agent",
        model_client=model_client,
        tools=MCP_tools,
        system_message=f"""你是一个可调用arXiv的AI助手。""",
        model_client_stream=True,  # Enable streaming tokens from the model client.
    )
```

3.3 开发及应用

- **直接对话MCP智能体：https://drsai.ihep.ac.cn**

- **OpenDrSai框架的开发及应用模式**
  - **智能体具有模块化的开发和加载方式**
  - **具有人机交互预设模式，直接兼容各种前端应用和人机交互模式**
  - **智能体代码开发者调试后即可发布应用**

我需要调研关于AI Agent在材料领域的应用

Title:Collaborative AI Enhances Image Understanding in Materials Science

Abstract: Recent breakthroughs in machine learning and artificial intelligence, fueled by scientific data, are revolutionizing the discovery of new materials. Despite the wealth of existing scientific literature, the availability of both structured experimental data and chemical domain knowledge that can be easily integrated into data-driven workflows is limited. The motivation to integrate this information, as well as additional context from first-principle calculations and physics-informed deep learning surrogate models, is to enable efficient exploration of the relevant chemical space and to predict structure-property relationships of new materials a priori. Ultimately, such a framework could replicate the expertise of human subject-matter experts. In this work, we present dZiner, a chemist AI agent, powered by large language models (LLMs), that discovers new compounds with desired properties via inverse design (property-to-structure). In specific, the agent leverages domain-specific insights from foundational scientific literature to propose new materials with enhanced chemical properties, iteratively evaluating them using relevant surrogate models in a rational design process, while accounting for design constraints. The model supports both closed-loop and human-in-the-loop feedback cycles, enabling human-AI collaboration in molecular design with real-time property inference, and uncertainty and chemical feasibility assessment. We demonstrate the flexibility of this agent by applying it to various materials target properties, including surfactants, ligand and drug candidates, and metal-organic frameworks. Our approach holds promise to both accelerate the discovery of new materials and enable the targeted design of materials with desired functionalities. The methodology is available as an open-source software on https://github.com/mehradans92/dZiner.
URL:http://arxiv.org/abs/2410.03963v1

Title:A Survey of AI for Materials Science: Foundation Models, LLM Agents, Datasets, and Tools

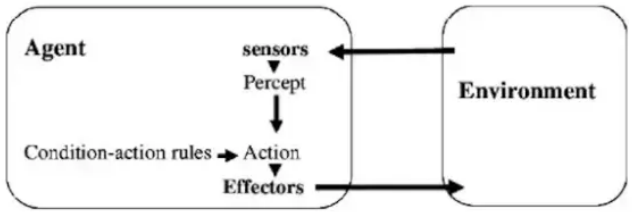
Abstract: Designing inorganic crystalline materials with tailored properties is critical to technological innovation, yet current generative computational methods often struggle to


Thank you for listening

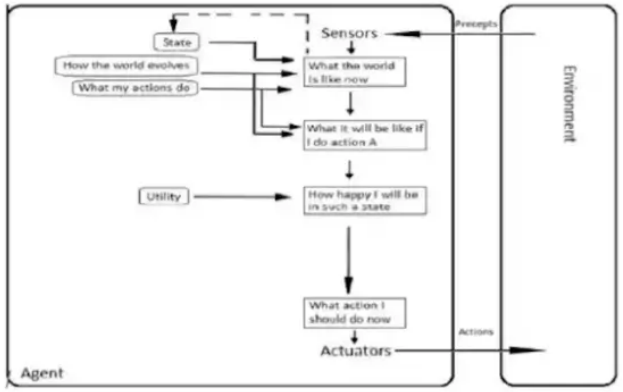
Backup

Types of AI Agent

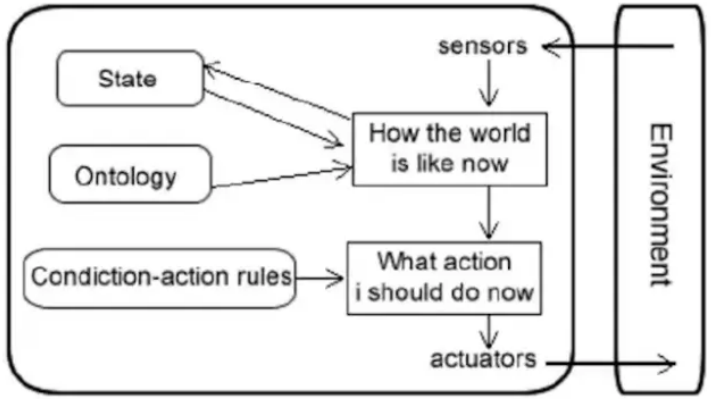
Simple Reflex Agents



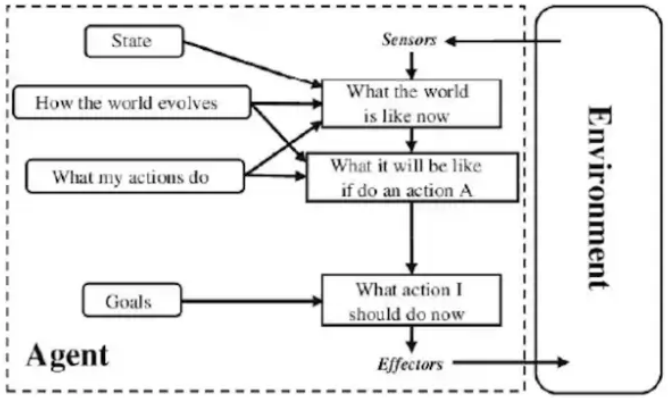
Utility-Based Agents



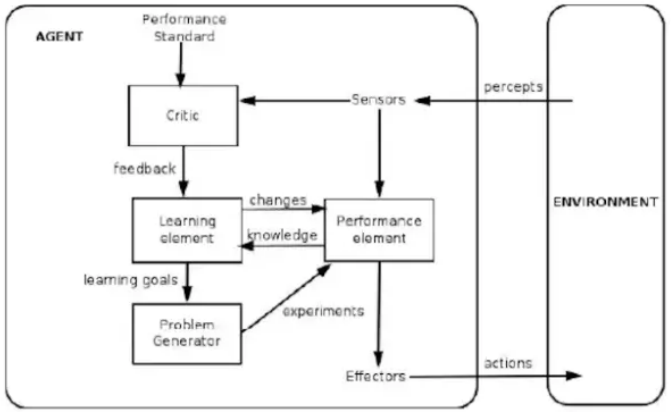
Model-Based Reflex Agents

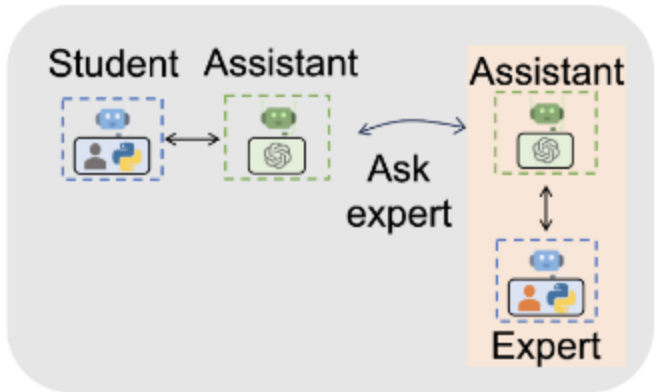


Goal-Based Agents

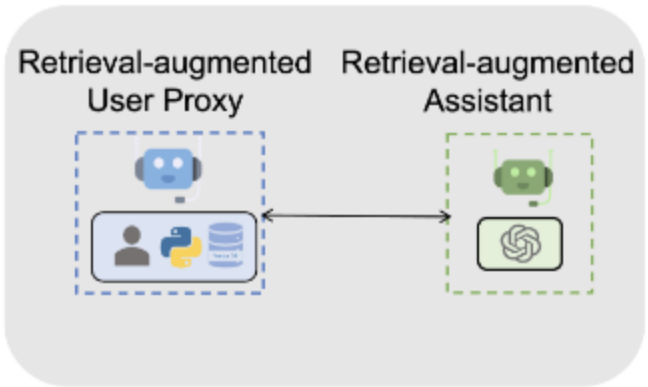


Learning Agents

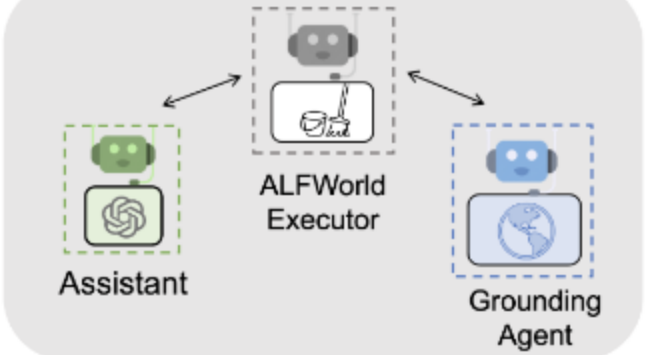




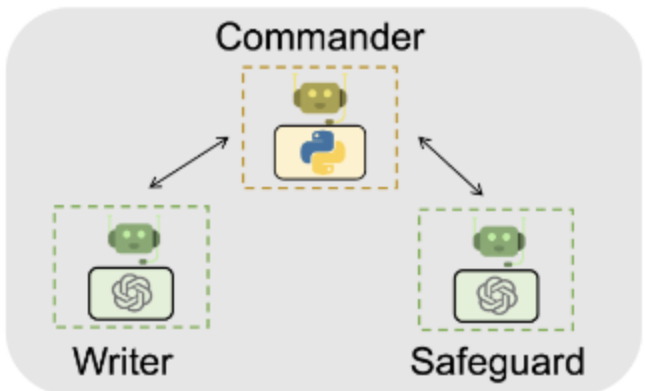
A1. Math Problem Solving



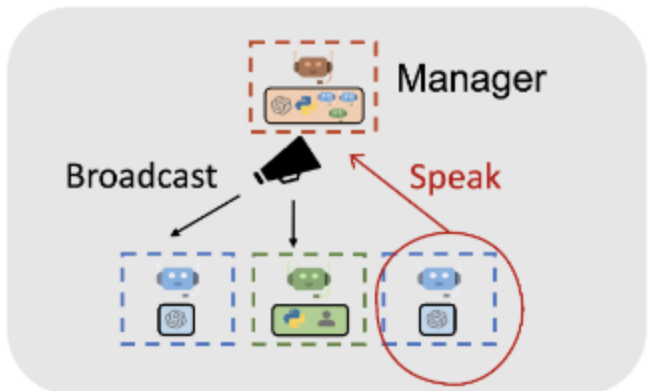
A2. Retrieval-augmented Chat



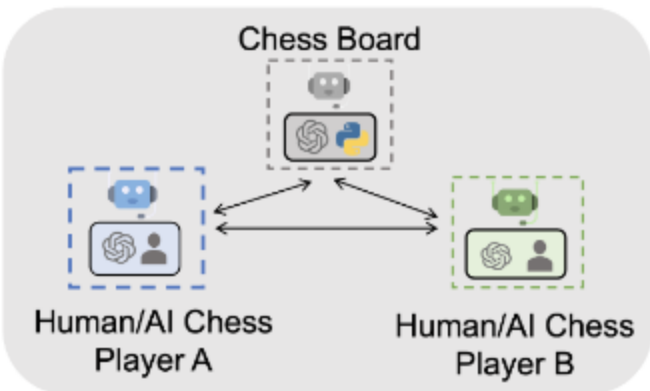
A3. Decision Making



A4. Multi-agent Coding



A5. Dynamic Group Chat



A6. Conversational Chess

Table 1. Comparison of Capabilities Across Agent Workflow Systems

System	Planning	Tool Use	Multi-agent	Memory	GUI	API	Self-Reflection	Custom Tools	Cross-Platform	Open-source	Year
AgentUniverse [54]	√	√	√	√	√	√	○	√	√	√	2023
Agentverse [55]	√	√	√	√	◑	√	√	√	√	√	2023
Agno [56]	√	√	√	√	×	√	√	√	√	√	2024
AutoGen [3]	√	√	√	√	×	√	√	√	◑	√	2023
CAMEL [57]	√	√	√	√	×	√	√	√	√	√	2023
ChatDev [58]	√	√	√	√	×	×	◑	√	√	√	2023
Coze [59]	√	√	√	√	√	√	○	√	√	√	2024
CrewAI [60]	√	√	√	○	×	√	√	√	◑	√	2024
DeepResearch [61]	√	√	○	√	√	√*	◑	√	√	×	2025
Dify [62]	√	√	×	√	×	√*	○	√	√	√	2023
DSPy [63]	√	√	○	√	√	√*	√	√	√	√	2023
ERNIE-agent [64]	√	√	√	○	√	√	×	√	√	√	2024
Flowise [65]	√	√	×	√	×	√	×	√	√	√	2023
LangGraph [66]	√	√	√	√	×	√	√	√	◑	√	2023
Magnetic-One [67]	√	√	√	√	√	√	√	√	◑	√	2024
Meta-GPT [2]	√	√	√	◑	×	√*	×	√	√	√	2023
n8n [68]	√	√	×	√	√	√	×	√	√	√	2019
OmAgent [69]	√	√	×	√	×	√	√	√	○	√	2024
OpenAI Swarm [70]	√	√	√	×	×	√	×	√	√	√	2024
Phidata [71]	√	√	√	√	×	√*	×	√	√	√	2024
Qwen-agent [72]	√	√	√	√	√	√	√	√	√	√	2024
ReAct [14]	√	√	×	×	×	√	×	√	×	√	2022
ReWoo [23]	√	√	×	×	×	√	×	√	×	√	2024
Semantic Kernel [73]	√	√	×	√	×	√*	×	√	√	√	2023