

# MASFactory: 一种以图为中心的框架，通过氛围图技术协调基于大语言模型的多智能体系统

Yang Liu<sup>1</sup>, Jinxuan Cai<sup>1</sup>, Yishen Li<sup>1</sup>, Qi Meng<sup>1</sup>, Zedi Liu<sup>1</sup>,  
Xin Li<sup>1</sup>, Chen Qian<sup>2</sup>, Chuan Shi<sup>1</sup> and Cheng Yang<sup>1\*</sup>

<sup>1</sup>Beijing University of Posts and Telecommunications

<sup>2</sup>Shanghai Jiao Tong University

{liuyang1999, yangcheng}@bupt.edu.cn

## 摘要

基于大语言模型 (LLM) 的多智能体系统 (MAS) 正越来越多地通过角色特化与协作来扩展智能体的问题求解能力。MAS 工作流可自然建模为有向计算图，其中结点执行智能体/子工作流，边则编码依赖关系和消息传递。

然而，在现有框架中实现复杂的图工作流仍需要大量手动操作，复用性有限，并且难以集成异构的外部上下文源。

为克服这些局限，我们提出 MASFactory，一个以计算图为中心的框架，用于编排基于 LLM 的多智能体系统。该框架引入 Vibe Graphing，一种人机协同方法，将自然语言意图编译为可编辑的工作流规范，进而生成可执行的计算图。此外，该框架提供可复用组件与可插拔的上下文集成能力，并配备可视化工具，支持拓扑预览、运行时追踪及人机协同交互。

我们在七个公开基准上评估了 MASFactory，验证了代表性 MAS 方法的复现一致性以及 Vibe Graphing 的有效性。我们的代码<sup>1</sup> 和视频<sup>2</sup> 是公开可用的。

## 1 引言

随着工具使用和反馈驱动的纠错机制的引入，大模型 (LLMs) 通常被封装为智能体，遵

循一种感知-推理-动作环，从而在外部环境中实现长时程任务执行 (Yao et al., 2023; Schick et al., 2023; Shinn et al., 2023)。随着任务复杂度的提升，单一智能体往往难以同时实现端到端覆盖、鲁棒的错误恢复以及多步骤协调。因此，基于大模型的多智能体系统 (MAS) 已成为一种广泛应用的方法，通过角色特化、交叉验证和迭代协作来扩展智能体的问题求解能力 (Li et al., 2023; Wu et al., 2023; Hong et al., 2024; Qian et al., 2024; Chen et al., 2024)。

对于 MAS 建模，一种日益常见的编排抽象是有向计算图。图由承载计算单元 (智能体、工具或子工作流) 的结点和编码执行依赖关系与消息传递方向的边组成。纯有向非循环图 (DAG) 工作流非常适合流水线式协作，而循环结构对于反思、修订和重试等迭代过程至关重要。最近的框架已开始显式地将多智能体工作流表示为图。例如，LangGraph 将工作流建模为具有显式执行语义的有状态图 (LangChain, 2024)，而 Dify 提供了一个工作流画布，其中 DAG 依赖关系同时定义执行顺序和数据流 (LangGenius, 2024)。然而，实现复杂的 MAS 仍然是工程密集型的。例如，开发者必须手动为智能体结点编写角色提示，连接结点间的路由逻辑，并建立智能体间的通信协议。此外，真实应用场景依赖于异构上下文源，包括记忆层 (Chhikara et al., 2025; Kang et al., 2025; Packer et al., 2024)、检索增强生成 (RAG) (Lewis et al., 2020; Edge et al.,

\* Corresponding author.

<sup>1</sup><https://github.com/BUPT-GAMMA/MASFactory>. 根据 Apache-2.0 许可授权；在该许可条款下允许使用、修改和分发。

<sup>2</sup><https://youtu.be/ANynzVFY32k>

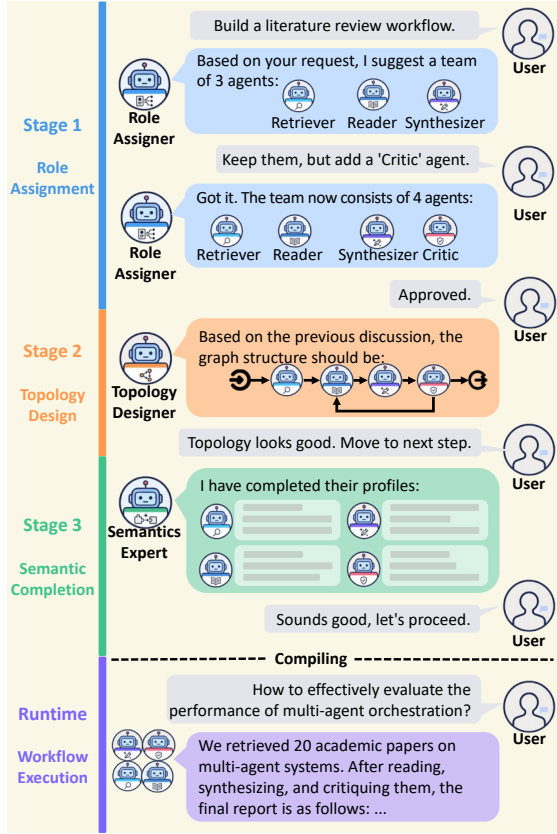


图 1: MASFactory 中的氛围图构建。MASFactory 通过一个三阶段的人机协同过程，将用户的自然语言意图转化为可执行的多智能体工作流，随后在运行时编译并执行生成的工作流。

2024)，以及标准化工具/上下文集成协议（如模型上下文协议（MCP））（Anthropic, 2025）。当前框架通常通过特定于工作流的胶水代码集成这些组件，使得在不同环境中移植和复用实现变得困难。最后，MAS 开发经常涉及重复的子图，这些子图全局相似但局部配置略有不同。现有框架对这类子图的版本控制、模板化复用支持有限。

为应对这些挑战，我们提出 **MASFactory**，一个可组合的多智能体系统编排框架，配备 **Vibe Graphing**（如图 1 所示）。首先，MASFactory 引入了 Vibe Graphing：用户以自然语言描述设计意图，系统将其编译为可读、可编辑且支持版本控制的结构化中间表示，该表示进一步被编译为可执行的工作流。此过程融入 人机协同交互，允许用户通过可视化界面审查、修改并提供反馈。其次，MASFactory

通过上下文适配器（Context Adapter）提升可操作性，隐藏不同上下文源之间的异构性。第三，MASFactory 通过提供可配置、可复用的图模块来提升复用性，这些模块对应常见的多智能体协作模式，使开发者能够快速组装工作流，而无需重复实现结构相似的子图。此外，MASFactory 提供了一个可视化工具，支持对工作流拓扑的视觉检查、运行时迹的可视化以及人机协同交互。

贡献。（1）我们提出 **MASFactory**，一个以图为中心的 MAS 编排框架，具备可重用组件和可插拔上下文管理，能够实现自然语言工作流生成并保证一致且可复现的结果。（2）我们引入 **Vibe Graphing**，一种人机协同方法，将自然语言意图编译为可执行图，降低实现成本的同时，在性能上达到与手动实现工作流相当的水平。（3）我们使用 MASFactory 复现了五个代表性 MAS，并在七个公开基准上展示了具有竞争力的性能。

## 2 相关工作

**基于大语言模型的多智能体系统。** 近期工作表明，协调多个大语言模型智能体可通过角色特化、交叉验证和迭代协作提升鲁棒性和覆盖范围（Li et al., 2023; Chen et al., 2024; Shen et al., 2023）。例如，AutoGen 推广了可编程的多智能体对话模式与群体协作机制（Wu et al., 2023）。MetaGPT 通过角色定义和类似标准操作流程的步骤来组织复杂任务的协作（Hong et al., 2024）。ChatDev 将软件开发协作组织为分阶段的角色交互（Qian et al., 2024）。然而，这些工作的实现方式差异显著，其代码库通常包含数千至数万行代码，使得开发者难以在此基础上进行构建。

**多智能体系统开发框架。** 除了研究原型之外，一个日益增长的生态系统正致力于实际的多智能体系统（MAS）实现。Google 的智能体开发工具包（ADK）（Google, 2024）和 CrewAI（CrewAI Inc., 2024）强调以代码为先、流程驱动的

方法：前者提供了一套工具用于部署智能体应用，而后者则专注于协同智能体团队的程序化编排。另一方面，以图为中心的框架如 LangGraph (LangChain, 2024) 和 Dify (LangGenius, 2024) 将智能体工作流表示为显式的结点/边结构，以实现执行流程的细粒度控制。然而，构建复杂的 MAS 仍然具有较高的工程复杂性，需要大量的手动配置以及紧密耦合的上下文集成。为了简化编排过程，我们设计了 MASFactory，能够通过 Vibe Graphing 将用户意图转化为可执行的图结构。

### 3 系统设计

在本节中，我们介绍 MASFactory 的架构。如图 2 所示，在底层系统采用由结点和边组成的骨干结构来建模协作图。在此骨干之上，MASFactory 提供了可组合的组件和模块，以实现灵活的协作过程。此外，MASFactory 统一了 MAS 关键机制（通信协议和上下文管理），并将其构建为可插拔模块。因此，系统能够方便地集成外部框架，如 Mem0 (Chhikara et al., 2025) 和 LlamaIndex (LlamaIndex, 2024)，从而提升用户体验。MASFactory 还提供了三种编排接口：通过 Vibe 图示实现自动化编排，以及通过声明式或命令式编程实现手动工作流编写。为了便于开发与调试，MASFactory 还提供了一个可视化工具，用于拓扑预览、运行时追踪以及在整个工作流中的人机协同交互。

#### 3.1 基本组件

**基于图形的组织。** MASFactory 将协作建模为由 Node 和 Edge 组成的有向图 (Zhuge et al., 2024)。Node 是基本计算单元，可扩展为自由组合的组件，包括 Graph、Loop、Agent、CustomNode、Interaction 和 Switch。这些组件提供一致的接口，但在执行逻辑上有所不同。Edge 表示结点间的依赖关系，并作为消息传递的载体。

**协作流程。** MASFactory 通过将协作信号分解为三个流程，使其更加明确。**控制流**沿边

传播，以推进调度和依赖关系，确保执行的因果约束。**消息流**沿边水平传播，将结点输出传递给下游结点。**状态流**沿图与子图之间的层次结构传播，以同步图级别的上下文和运行时状态。

**结点生命周期。**运行时，每个结点遵循统一的生命周期。结点首先沿其 in edges 聚合传入的消息到其输入，然后从父级 Graph 读取状态。随后执行结点特定的逻辑，例如对 Agent 调用大语言模型 (LLM)，对 Graph 和 Loop 按拓扑顺序调度内部结点。执行完成后，结点通过其 ou edges 将生成的消息分发给下游结点，并最终将更新后的状态写回父级 Graph，以同步共享状态并支持下游执行。

**运行时调度。** MASFactory 采用基于就绪状态的调度策略，允许多个就绪结点并发执行。此统一机制支持顺序、并行、分支和循环控制结构。

#### 3.2 关键部件

**图与环。** Graph 和 Loop 负责内部结点的拓扑表示和调度。Graph 表示并调度有向非循环图 (DAG) 工作流。Loop 表示并调度循环结构，常用于迭代协作模式，如反射、修订和重试。

**开关。** Switch 在图中实现控制流的路由。与普通结点默认向所有下游结点广播消息并触发信号不同，Switch 根据运行时状态动态选择并激活一个或多个下游路径。

**交互。** 一个 Interaction 结点作为人机协同机制的入口点。它可以在执行过程中主动向用户查询，收集反馈，并将用户输入重新注入工作流中。

#### 3.3 智能体组件

**感知-推理-动作。** 智能体采用经典的感知-推理-动作范式，并通过可插拔的设计对通信和上下文等关键步骤进行模块化处理。如图 2 所示，智能体依赖于可插拔的消息适配器进行消息处理，以及可插拔的上下文适配器



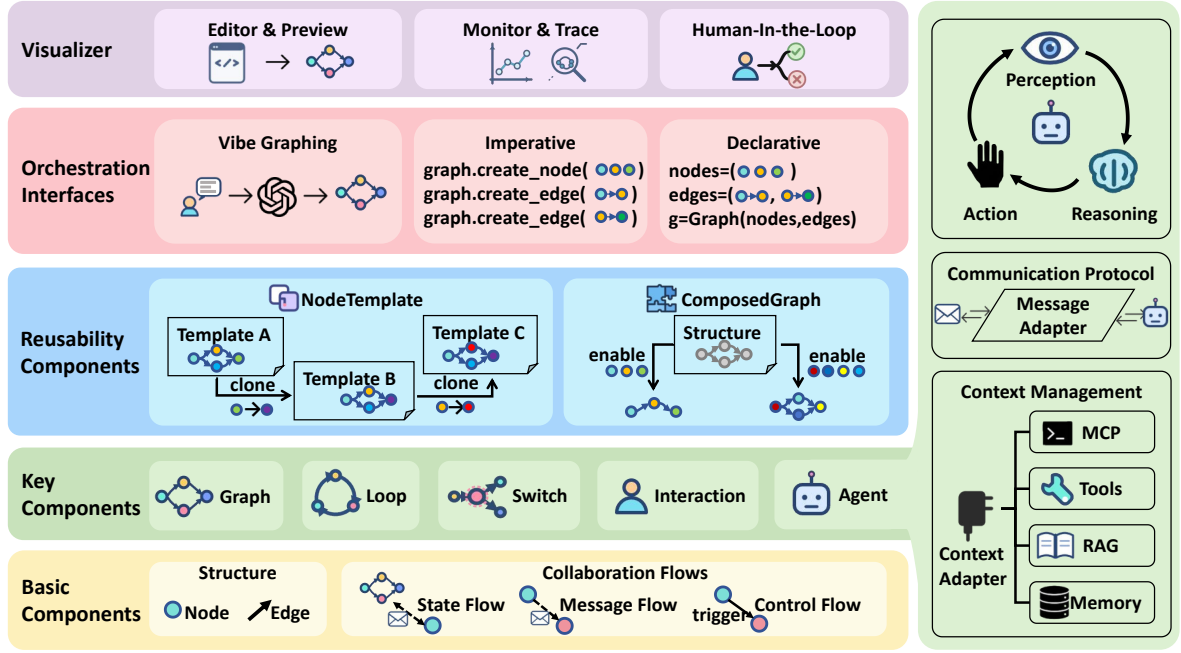


图 2: MASFactory 框架的架构概述。

进行上下文管理。

**消息适配器。**为了实现通信，消息适配器根据给定的通信协议对智能体的输入和输出进行格式化。

MASFactory 提供了一套常用的适配器，包括基于 JSON 模式、结构化 Markdown 段落以及纯文本段落格式的协议。它还提供了用户自定义协议的接口。通过引入消息适配器，MASFactory 将协作图与协议定义解耦，使得在不修改协作拓扑结构的情况下，能够扩展或替换协议。

**上下文适配器。**为了管理上下文，屏蔽来自不同信息源（如 Memory、MCP 和 RAG (Packer et al., 2024; Kang et al., 2025; Anthropic, 2025; Lewis et al., 2020; Edge et al., 2024)) 的异构性，MASFactory 使用一个上下文适配器来提供标准化接口。该适配器将不同的外部上下文分割成标准化单元，从而实现与不同上下文框架（如 Mem0 和 LlamaIndex）的统一无缝集成。

### 3.4 可重复使用性与模板化原型设计

**结点模板。**结点模板允许用户先声明一个结构模板，然后在后续实例化为具体的图。这种声明与实例化之间的解耦使得用户能够克隆模板以构建多个具有相似全局结构但局部配置不同的图，支持分支式复用和版本化管理。此外，开发人员在实例化图时可以复用图级模板，同时调整结点点级设置。

**ComposedGraph.** ComposedGraph 是一种专门的 Graph 形式，用于表示一类预定义的结构。它可以通过填充结点配置或根据用户参数激活特定分支来实例化具体的图结构，从而隐藏底层构建细节。通过 ComposedGraph，用户可以将设计封装为可重用的复合组件。同时，MASFactory 使用 ComposedGraph 来封装难以用静态图表达的结构，例如 DyLan 风格的动态调度模式 (Liu et al., 2024)，以及常用的协作子图。这不仅支持更广泛的应用场景，还降低了开发成本。

### 3.5 编排接口

**情绪图谱构建。**在情绪图谱构建中，MASFactory 通过分阶段编译将自然语言意图转化

为可执行的多智能体 workflow，同时每个阶段提供人机协同的审查与修改。在此过程中，系统生成一种可读、可编辑且结构化的中间表示，随后将其编译为可执行的 workflow。如图 1 所示，该流水线完成三项核心任务。角色分配将任务意图映射为一组具有明确责任边界的候选智能体。结构设计基于智能体间的信息依赖关系和控制约束，生成有向图拓扑骨架，确定连接性以及消息与控制传播的方向。语义补全通过对骨架进行参数化实例化，为每个结点配置提示词和工具，生成可直接编译与执行的 workflow。这一意图-结构-实例化编译链将多智能体系统构建从手动 workflow 配置提升为迭代式设计过程，在降低实现成本的同时保留对拓扑结构和语义的控制权。

**命令式接口。**命令式接口以代码为中心：开发者通过程序化地实例化结点和边，并使用显式的控制逻辑将它们连接起来构建图。这种风格提供了高度的灵活性以及对拓扑结构、参数和运行时行为的精确控制，适用于精心设计的工作流以及需要与应用逻辑紧密耦合的场景。

**声明式接口。**声明式接口将 workflow 指定为一种结构化配置：开发者声明图的拓扑结构和结点属性，MASFactory 会据此构建可执行的图。这种风格使编程简洁且易于审查，适用于固定 workflow 和轻度动态结构。

### 3.6 可视化工具

可视化工具是一个作为 VS Code 插件实现的可视化集成环境。它在单一视图中将静态 workflow 拓扑与运行时迹对齐，并支持以下功能：

**编辑器 & 预览。**编辑器 & 预览支持在开发过程中实时查看拓扑结构并进行结构检查。

**监控 & 迹。**该可视化工具在执行过程中跟踪结点状态演化和消息传播，以支持调试和诊断。

**人机协同。**人机协同与 Interaction 结点配合，用于可视化运行时的人机交互，并将外部反馈或输入整合到 Vibe Graphing workflow 中，实现交互式干预和迭代。

## 4 评估与分析

我们进行了大量实验，以回答以下问题：(1) MASFactory 能否以一致的有效性复现具有代表性的 MAS 方法；(2) Vibe Graphing 生成的 workflow 是否能与人工设计的 workflow 相媲美；(3) MASFactory 能否通过组件复用和意图驱动的编排降低实现成本。

### 4.1 实验设置

**基准。**我们在面向编码的基准 (HumanEval、MBPP、BigCodeBench、SRDD) (Chen et al., 2021; Austin et al., 2021; Zhuo et al., 2025; Qian et al., 2024) 上进行评估，以及通用推理/工具使用基准 (MMLU-Pro、GAIA、GPQA) (Wang et al., 2024; Mialon et al., 2024; Rein et al., 2024)。所有得分均以百分制 (0-100) 报告。除 SRDD 外，其他编码基准报告 pass@1；SRDD 报告综合质量得分 (SRDD)；MMLU-Pro/GAIA/GPQA 报告准确率。

**大模型骨干。**在 Vibe Graphing 中，workflow 构建阶段使用 gpt-5.2 将自然语言意图编译为可执行的 workflow。在所有方法的工作流执行阶段，我们使用 gpt-4o-mini。

### 4.2 重现工作流的性能

表 1 比较了在 MASFactory 中实现的五种代表性多智能体系统 (MAS) 方法与其原始实现，包括 ChatDev (Qian et al., 2024)、MetaGPT (Hong et al., 2024)、AgentVerse (Chen et al., 2024)、CAMEL (Li et al., 2023) 以及 HuggingGPT (Shen et al., 2023)。总体而言，MASFactory 的复现结果在不同基准上与原始结果基本一致甚至更优。这表明 MASFactory 能够覆盖多样化的多智能体架构与协作设计，且未出现系统性回归。

### 4.3 Vibe Graphing 的性能

我们在表 1 中报告了两种 Vibe Graphing 情景。在 Vibe Graphing-ChatDev 情景中，我

表 1: 主要结果以百分比形式报告 (0-100)。符号 “-” 表示由于专注于编程的 MAS 与通用推理基准之间不兼容, 因此不适用。

Method	HumanEval	MBPP	BigCodeBench	SRDD	MMLU-Pro	GAIA	GPQA
ChatDev (original)	82.50	71.40	50.70	82.91	-	-	-
ChatDev (MASFactory)	81.30	74.20	53.30	84.23	-	-	-
MetaGPT (original)	67.07	36.03	50.10	78.19	-	-	-
MetaGPT (MASFactory)	89.02	59.14	51.70	72.77	-	-	-
AgentVerse (original)	85.00	74.54	65.92	87.55	64.64	12.12	38.39
AgentVerse (MASFactory)	85.00	75.15	64.12	91.06	64.16	12.73	37.50
CAMEL (original)	62.20	60.60	63.51	89.42	50.08	9.70	32.59
CAMEL (MASFactory)	71.85	57.80	78.16	89.69	63.04	12.73	24.78
HuggingGPT (original)	82.32	68.60	28.42	87.96	65.59	9.09	56.67
HuggingGPT (MASFactory)	80.49	64.40	29.91	83.26	63.66	10.91	47.32
Vibe Graphing-ChatDev	83.50	74.20	45.30	88.13	-	-	-
Vibe Graphing-Task Specific	84.76	72.37	51.67	90.71	51.73	12.12	39.51

们将 ChatDev 的每个关键阶段替换为一个独立的 VibeGraph 组件, 并使用轻量级胶水代码将这些阶段连接成一个工作流。相比之下, *Vibe Graphing-Task Specific* 是一种任务驱动的变体: 针对每个基准数据集, 开发者编写一个自然语言描述的工作流, 然后将该指令传递给单个 VibeGraph 组件, 由其编译为可执行的工作流。表 1 的最后两行报告了使用 Vibe Graphing 生成的工作流的结果。尽管仅在工作流构建阶段使用了不同的模型 (gpt-5.2), 但生成的工作流仍以 gpt-4o-mini 执行, 在编码基准测试中取得了具有竞争力的表现, 同时在适用的一般推理/工具使用任务中也表现良好。这些结果表明, 分阶段的意图到图编译方法能够生成可行的多智能体工作流, 其性能接近手动设计的基准, 同时显著减少了底层图连接和框架特定的工程工作量。

#### 4.4 实施成本案例研究

ChatDev 的原始实现包含 1,511 行 Python 代码用于工作流定义。通过基于 ComposedGraph 的复用, 我们的 MASFactory 实现将总代码量减少至 1,114 行, 同时保持了相

当的性能, 如表 1 所示。如果我们进一步使用 Vibe Graphing 实现每个 ChatDev 阶段的交互式生成 (即 Vibe Graphing-ChatDev), 则连接这些阶段仅需 203 行代码。当我们完全依赖 Vibe Graphing 实现端到端的工作流生成 (即 Vibe Graphing-特定任务), 工作流规范缩减至仅 45 行代码。

此外, 在相同的 gpt-5.2 后端下, 如表 2 所示, Vibe Graphing 相较于 Vibe Coding 将 API 成本降低了约一个数量级。通过 Vibe Coding 生成的工作流在构建的图中经常表现出逻辑缺陷, 无法返回正确的执行结果。因此, 我们将与 Vibe Coding 的比较限定在成本分析上, 并将其排除在性能评估之外。

总而言之, 组件化设计提升了可重用性, 而 Vibe Graphing 通过减少样板代码和手动配置显著降低了开发门槛。我们在附录 A 中提供了一个 Vibe Graphing 的案例研究, 详细介绍了从意图到可执行工作流的流水线构建过程。

表 2: Vibe Graphing (VG) 与 Vibe Coding (VC) 的货币成本比较。VC-L 和 VC-M 分别表示 VC 的低推理长度和中等推理长度。

Metric	ChatDev			AgentVerse		
	VG	VC-L	VC-M	VG	VC-L	VC-M
Cost (\$)	0.26	3.49	3.02	0.59	4.43	6.08

## 5 结论

我们提出 MASFactory，一个以图为中心的框架，用于编排基于大语言模型的多智能体系统。MASFactory 将多智能体工作流建模为可执行的有向图，并集成 Vibe Graphing 技术，通过人机协同优化，将自然语言意图编译为多智能体工作流。该框架还提供可复用组件、可插拔上下文以及拓扑预览、运行时追踪和可视化人机协同交互的可视化工具。在七个公开基准上的实验表明，MASFactory 能够一致地复现代表性多智能体系统，且 Vibe Graphing 生成的工作流具有竞争力，同时显著降低了实现开销。

## 局限性

MASFactory 目前不提供内置的检查点功能，无法在中断后从中间状态恢复执行。此外，我们将在未来的更新中持续丰富和完善内置的组件库。

## References

- Anthropic. 2025. [Model context protocol \(MCP\) specification](#). Accessed: 2026-02-23.
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. [Program synthesis with large language models](#). *arXiv preprint arXiv:2108.07732*.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. [Evaluating large language models trained on code](#). *Preprint*, arXiv:2107.03374.
- Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2024. [AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors](#). In *The Twelfth International Conference on Learning Representations (ICLR)*. OpenReview.net.
- Prateek Chhikara, Dev Khant, Shreyas Aryan, Tushar Singh, and Deshraj Yadav. 2025. [Mem0: Building production-ready AI agents with scalable long-term memory](#). *Preprint*, arXiv:2504.19413.
- CrewAI Inc. 2024. [CrewAI](#). Accessed: 2026-02-23.
- Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. [From local to global: A graph RAG approach to query-focused summarization](#). *Preprint*, arXiv:2404.16130.
- Google. 2024. [Agent development kit \(ADK\)](#). Accessed: 2026-02-23.
- Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xianwu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. [MetaGPT: Meta programming for a multi-agent collaborative framework](#). In *The Twelfth International Conference on Learning Representations (ICLR)*. OpenReview.net.
- Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. 2025. [Memory OS of AI agent](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
- LangChain. 2024. [LangGraph](#). Accessed: 2026-02-23.
- LangGenius. 2024. [Dify: An LLM app development platform](#). Accessed: 2026-02-23.
- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. [Retrieval-augmented generation for knowledge-intensive NLP tasks](#). In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. [CAMEL: Communicative agents for "mind"](#)



- exploration of large language model society. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. 2024. [A dynamic LLM-powered agent network for task-oriented agent collaboration](#). In *First Conference on Language Modeling (COLM)*.
- LlamaIndex. 2024. [LlamaIndex](#). Accessed: 2026-02-23.
- Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2024. [GAIA: a benchmark for general AI assistants](#). In *The Twelfth International Conference on Learning Representations (ICLR)*.
- Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. [MemGPT: Towards LLMs as operating systems](#). In *The Twelfth International Conference on Learning Representations (ICLR)*.
- Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. 2024. [ChatDev: Communicative agents for software development](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*.
- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. [GPQA: A graduate-level google-proof Q&A benchmark](#). In *First Conference on Language Modeling (COLM)*.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. [Toolformer: Language models can teach themselves to use tools](#). In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. [Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face](#). In *Advances in Neural Information Processing Systems*, pages 38154–38180. Curran Associates, Inc.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. [Reflexion: Language agents with verbal reinforcement learning](#). In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyang Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhua Chen. 2024. [MMLU-Pro: A more robust and challenging multi-task language understanding benchmark](#). *Preprint*, arXiv:2406.01574.
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryan W White, Doug Burger, and Chi Wang. 2023. [AutoGen: Enabling next-gen LLM applications via multi-agent conversation](#). *Preprint*, arXiv:2308.08155.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. [ReAct: Synergizing reasoning and acting in language models](#). In *The Eleventh International Conference on Learning Representations (ICLR)*. OpenReview.net.
- Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. [GPTSwarm: Language agents as optimizable graphs](#). In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pages 62743–62767.
- Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, Thong Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, and 14 others. 2025. [BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions](#). In *The Thirteenth International Conference on Learning Representations (ICLR)*.



## A 氛围图谱案例研究

在此，我们展示一个具体的 Vibe 绘图示例，说明从自然语言意图到结构化 workflow 规范及可执行 workflow 定义的端到端流水线。

### A.1 意图与结构约束

我们从一个自然语言构建指令开始，该指令既指定了任务意图（撰写周报）又包含一个显式的结构约束： $START \rightarrow A, B, C \rightarrow D \rightarrow END$ ，其中  $A$ 、 $B$  和  $C$  是并行起草智能体， $D$  是评估/选择智能体，负责生成最终输出。

本案例研究中使用的构建指令为 设计一个撰写每周报告的工作流。我将首先提供本周的工作内容。然后并行运行三个智能体分别起草报告，再将所有草稿交给第四个智能体进行评估并选择最佳版本作为最终输出。预期的工作流结构为： $START \rightarrow A, B, C \rightarrow D \rightarrow END$ 。

### A.2 交互式 workflow 构建

如图 5 所示，我们编写了一个简短的程序，构建一个具有拓扑结构  $ENTRY \rightarrow Vibe\ Graph \rightarrow EXIT$  的最小封装图。其中，Vibe Graph 是 MASFactory 中内置的一个复合图，其在图构建时会激活 Vibe Graphing 流水线。如图 1 所示，该流水线包含三个阶段。每个阶段均由一个 Loop 组件封装，并围绕一个带有辅助机制的 Agent 结点展开，这些辅助机制用于修正、审查以及通过 Interaction 实现人机协同交互。在每个阶段生成中间设计后，系统会请求用户反馈，并迭代优化结果，直至用户接受为止，此时流水线才进入下一阶段。在第 2 阶段和第 3 阶段的人机协同交互过程中，用户可选择 (i) 通过可视化工具直接编辑结构化的中间表示，或 (ii) 在交互面板中提供反馈。无论是手动修改还是文本反馈，均会被记录并作为后续修订的参考信息回传给 Agent。图 3 展示了可视化工具中的一个交互迹例，而图 4(a) 展示了工作流预览与编辑界面。

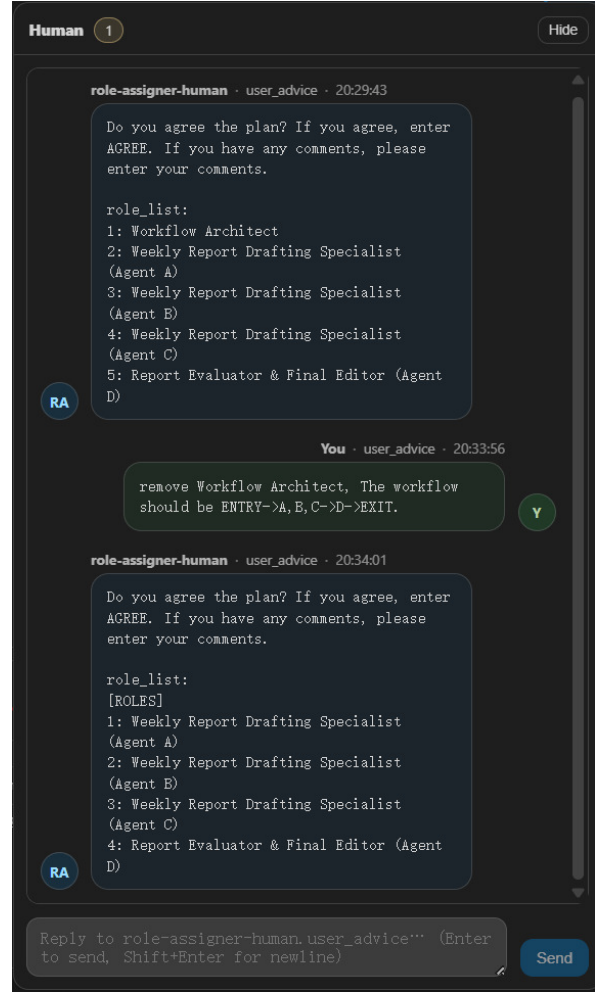


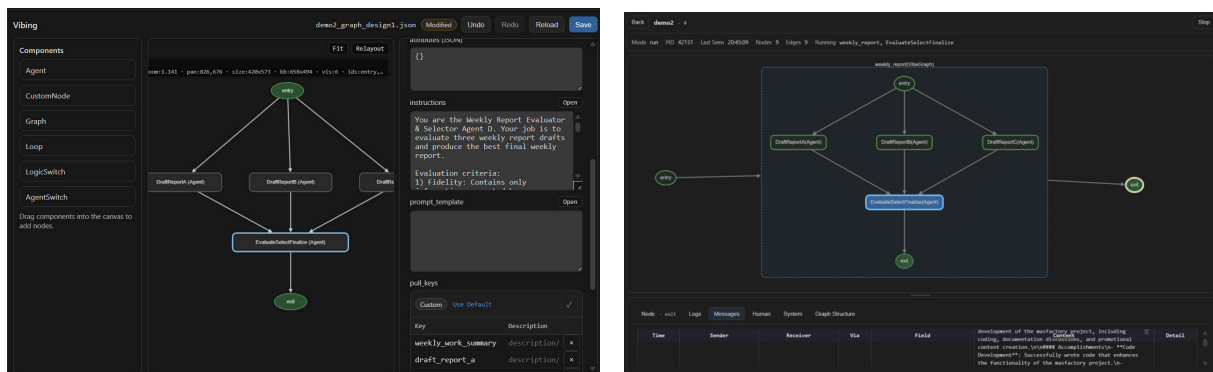
图 3: 在可视化工具中进行振动图绘制时的人机协同交互。用户提供的反馈用于优化各阶段的角色分配与结构。

### A.3 工作流规范快照

如图 6 所示，我们展示了 Vibe Graphing 生成的结构化中间表示的一个快照。该表示显式地编码了：(i) 结点语义（例如 label 和 instructions），(ii) 输入/输出契约（例如 input\_fields 和 output\_fields），以及 (iii) 结点之间的有向依赖关系（即 edges）。

### A.4 工作流执行

运行时，可视化工具提供执行状态和消息迹的同步视图，如图 4(b) 所示。工作流首先收集用户输入并将其归一化为 weekly\_work\_summary，然后并行执行三个起草分支，最后汇总草案以进行选择 and 轻度编



(a) MASFactory 可视化工具中的红外预览与编辑。 (b) MASFactory 可视化工具中的运行时预览与迹对齐。

图 4: Vibe 绘图中使用的可视化视图。

辑。

```

from masfactory import VibeGraph, NodeTemplate, OpenAIModel, RootGraph
invoke_model = OpenAIModel(api_key=os.environ.get("OPENAI_API_KEY", ""))
    base_url=os.environ.get("OPENAI_BASE_URL",""), 模型 _ 名称 ="gpt-4o-mini")
build_model = OpenAIModel(api_key=os.environ.get("OPENAI_API_KEY", ""))
    base_url=os.environ.get("OPENAI_BASE_URL",""), model_name="gpt-5.2")
weekly_report = NodeTemplate(
    VibeGraph,
    invoke_model= 调用 _ 模型,
    build_model= 构建 _ 模型,
    build_instructions="... 开始->A,B,C->D-> 结束。"
    build_cache_path= 图形设计缓存路径,
    pull_keys={"my_works": "我本周完成的工作"},
    push_keys={"final_weekly_report": "每周报告"},
)
root = RootGraph(name="demo2", nodes=[("weekly_report", weekly_report)],
    edges=[("入口", "weekly_report", {}), ("weekly_report", "出口", {})]
root.build()

msg, attr = root.invoke({}, {"my_works": my_works})
print(attr["最终每周报告"])

```

图 5: Vibe 绘图的 MASFactory 编程代码。

```

{
  "edges": [
    {"source": "ENTRY", "target": "绘图员 A"}, {"source": "ENTRY", "target": "绘图员 B"},
    {"source": "条目", "target": "绘图员 C"}, {"source": "绘图员 A", "target": "终稿员"},
    {"source": "绘图员 B", "target": "最终确定者"}, {"source": "绘图员 C", "target": "最终确定者"},
    {"source": "终结器", "target": "EXIT"}
  ],
  "结点": [
    {"id": "DrafterA", "type": "动作", "input_fields": ["my_work"],
      "output_fields": ["draft_report_a"], "instructions": "你是一周报告起草智能体 A ..."}
    {"id": "DrafterB", "type": "动作", "input_fields": ["my_work"],
      "output_fields": ["draft_report_b"], "instructions": "你是一个周报起草智能体 B ..."}
    {"id": "DrafterC", "type": "Action", "input_fields": ["my_work"],
      "output_fields": ["draft_report_c"], "instructions": "你是一个周报起草智能体 C ..."}
    {"id": "终结器", "type": "Action",
      "input_fields": ["我的工作", "草稿报告_a", "草稿报告_b", "草稿报告_c"],
      "output_fields": ["final_weekly_report", "selection_rationale"],
      "instructions": "你是每周报告评估员..."}
  ]
}

```

图 6: Vibe Graphing 生成的 workflow 规范。