OpenDrawio Core 项目说明书

OpenDrawio Team
项目参与人：董佩杰、王强、唐桢桁、苏小彦、林铠涛

2026 年 3 月 16 日

目录

1 项目概述

OpenDrawio Core 是一个面向 `draw.io / diagrams.net` 的对话式智能画图平台。项目由 OpenClaw 后端服务与前端工作台组成，目标是在保留 Drawio 可编辑性的前提下，让用户通过自然语言完成图表创建、修改、审查和持续迭代。

这里所说的 OpenClaw，本质上可以理解为一种面向图任务的 agent。它不是只负责返回一段文本的普通对话接口，而是会围绕当前图文件、会话历史和可用工具持续做判断，再把“用户想表达什么”转成“系统下一步应该回答、澄清还是直接改图”。因此，在产品语义里，OpenClaw 扮演的是“会理解上下文、会调用能力、会真正动手改图”的执行中枢。

项目强调三点。第一，结果不是一次性图片，而是可继续编辑的 `diagram.drawio` 文件。第二，用户、图文件和会话历史彼此绑定，适合多图、多轮交互。第三，工程结构已经具备验证脚本、状态文档和功能验收清单，适合持续演进，而不只是一次性 Demo。

2 产品价值

相较于传统画图工具，这个项目的价值并不只是“生成一张图”，而是缩短从想法到结构化图表的路径。用户不需要先学习大量操作，再通过拖拽整理结构，而是可以先用自然语言表达意图，再在图上继续迭代。这样既降低了第一次出图成本，也保留了后续精修空间。

从组织角度看，该项目还有三项更长期的价值。其一，图表成果以可编辑的 Drawio 文件形式保存，适合在后续需求讨论、技术评审和交付文档中复用。其二，多轮会话和图文件绑定，使每张图的演化路径更清晰。其三，前后端、验证脚本和交接文档已经形成稳定约束，适合多人或多轮 OpenClaw 协作接力维护。

3 项目参与情况

当前说明书记录的 Drawio 项目参与人包括：董佩杰、王强、唐桢桁、苏小彦、林铠涛。

在说明书层面，参与人信息的意义不只是署名，而是明确该项目已经具备多人协作背景。对于这种同时涉及产品、前端、后端、测试和文档的系统，协作方式本身就是交付质量的一部分。只有当需求、实现、验证和文档都能被不同参与者持续接力，项目才能稳定演进。

因此，本项目强调两类协作机制。第一类是代码与接口层面的边界清晰，避免前后端或工具链之间出现职责重叠。第二类是文档与验证层面的边界清晰，确保每次改动都能留下状态记录、验证结果和可接续的上下文。

4 项目目标与范围

4.1 建设目标

1. 让用户通过自然语言快速生成和修改流程图、架构图、关系图等常见图表。
2. 保留 Drawio XML 作为核心资产，使图表可以被继续编辑、导出和复用。
3. 建立清晰的前后端边界、多租户隔离边界和长期维护机制。
4. 为流式对话、生图渲染、审查、版本恢复等高级能力提供可扩展底座。

4.2 当前范围

当前版本覆盖以下范围：

- 浏览器内登录、图集切换、聊天改图、画布同步与保存。
- 以 `user_id + diagram_id` 为核心的工作区和会话隔离。
- 后端接口、前端应用、验证脚本、功能验收清单和协作接力文档。

当前不以自研画布、复杂组织权限、计费系统或企业级协同流程为主要目标。

4.3 交付边界

本说明书面向“当前仓库可支撑的产品和工程能力”，因此只描述已经实现或已被仓库明确验证的主链路，不把未来规划包装成当前能力。对于画布替换、企业权限、计费、集中式存储等方向，本项目目前仍然处在可继续扩展但尚未完全交付的阶段。

5 目标用户与典型场景

5.1 目标用户

角色	主要诉求	典型操作
业务用户	快速表达结构，减少手工拖拽	直接描述流程或关系，让系统生成初稿
研发与架构人员	在已有图上持续细修	增删节点、重排结构、修正文案、补充组件
审查和协作人员	快速发现问题并统一表达	查看图、给出修改意见、触发审查或回退
运维与维护人员	部署、诊断、验证与接力维护	管理配置、查看健康状态、执行验证脚本

5.2 典型场景

场景一：从零创建图。用户输入一句自然语言，例如“画一个三层 Web 架构图，包含前端、API、MySQL 和 Redis”，系统生成可编辑的 Drawio 图。

场景二：在现有图上继续修改。用户在已有图基础上提出增量指令，例如补充缓存、拆分服务、修改颜色或布局，系统在原图上下文中继续调整，而不是重新起草。

场景三：多图并行工作。同一用户可以维护多张图，每张图都有独立标题、XML 文件和对话历史；切换图时，前端同步切换画布和消息记录。

场景四：人工编辑与自动回写。用户在嵌入的 Drawio 画布里直接修改内容，前端接收 `autosave/save` 事件并回写后端，保证浏览器状态和工作区文件一致。

场景五：逐步生成与审查。在较复杂的生成任务中，用户既可以等待流式回复逐步呈现，也可以在生成后继续发起结构审查、视觉审查或版本回退。也就是说，这个平台并不是“一次生成后结束”，而是允许围绕同一张图形成完整的工作流程。

6 系统组成

6.1 整体结构

子系统	技术基线	主要职责
OpenClaw 后端服务	Python + FastAPI	提供聊天、图文件、登录注册、工作区和会话相关接口
`drawio-web`	React + TypeScript + Vite	提供首页、登录页、对话工作台、图集侧栏和画布集成
项目状态文档	Markdown	记录当前状态、下一步任务、工程决策和会话接力信息
验证与交接脚本	Bash	提供启动、验证、交接和功能回归守卫脚本

6.2 核心架构说明

系统从逻辑上可以分为五层：

1. 用户交互层：首页、登录页、对话页和 Drawio 画布。
2. 前端状态层：认证状态、当前图、图集缓存、消息列表和 XML 缓存。
3. 接口层：`/v1/chat`、`/v1/chat/stream`、`/v1/diagram`、`/v1/login` 等统一入口。
4. OpenClaw 执行层：对话循环、上下文构建、技能注入、工具调用、会话管理。
5. 持久化层：工作区 XML、聊天历史、用户数据和项目运维文档。

这种分层使项目能够同时满足产品体验与工程维护要求：前端关注交互和画布联动，后端负责对话、文件和会话逻辑，文档与脚本负责长期可接力维护。

如果换一种更直白的说法，OpenClaw 就是这套系统里的 agent 层。它上接用户输入和前端状态，下接技能、工具、工作区文件以及模型能力，负责把模糊需求组织成可执行动作。也正因为有这一层，系统才不是“模型回答一句话”后就结束，而是能在同一张图上连续推进一个完整任务。

6.3 后端职责

后端的核心角色不是普通问答接口，而是图工作流调度器。它一方面要接收用户消息、组织上下文、调用模型并决定是否修改图；另一方面要负责会话历史、工作区边界、图文件读写和错误保护。因此后端设计更加接近“面向图任务的 OpenClaw 服务”，而不是单纯聊天 API。

进一步说，OpenClaw 实际上就是一种被工程化约束住的 agent。它既保留了 agent 的核心特征，即基于上下文做判断、调用外部能力、分步骤完成任务；又通过工作区边界、接口协议、验证脚本和持久化约束，避免它变成不可控的黑盒。对于项目说明书而言，这一点很重要，因为 OpenClaw 不是一个营销名称，而是系统里真正承担“理解需求并执行图任务”职责的核心角色。

6.4 前端职责

前端承担的也不仅是消息展示。它需要同时管理图集、当前图状态、画布嵌入、保存回写、错误提示、流式渲染和登录态恢复。换句话说，前端实际交付的是一个三栏式工作台，而不是一个普通聊天窗口。

6.5 多租户与持久化模型

系统采用 `user_id + diagram_id` 双键隔离模型。每个用户可以有多张图，每张图对应独立工作区和会话历史。核心持久化对象如下：

- 图文件：保存为当前图对应的 `diagram.drawio`。
- 会话历史：保存每张图的多轮聊天消息和状态。
- 用户信息：保存登录、注册和基础身份数据。

该设计便于本地开发、故障排查和功能回放，也为后续向数据库或对象存储迁移保留空间。

6.6 工作区组织方式

工作区天然按用户和图进行拆分。这样做带来三个好处：一是减少不同用户之间的文件干扰；二是让每张图天然拥有自己的上下文边界；三是使故障排查和回归测试可以直接围绕单个用户、单张图展开，而不必在共享目录里查找状态。

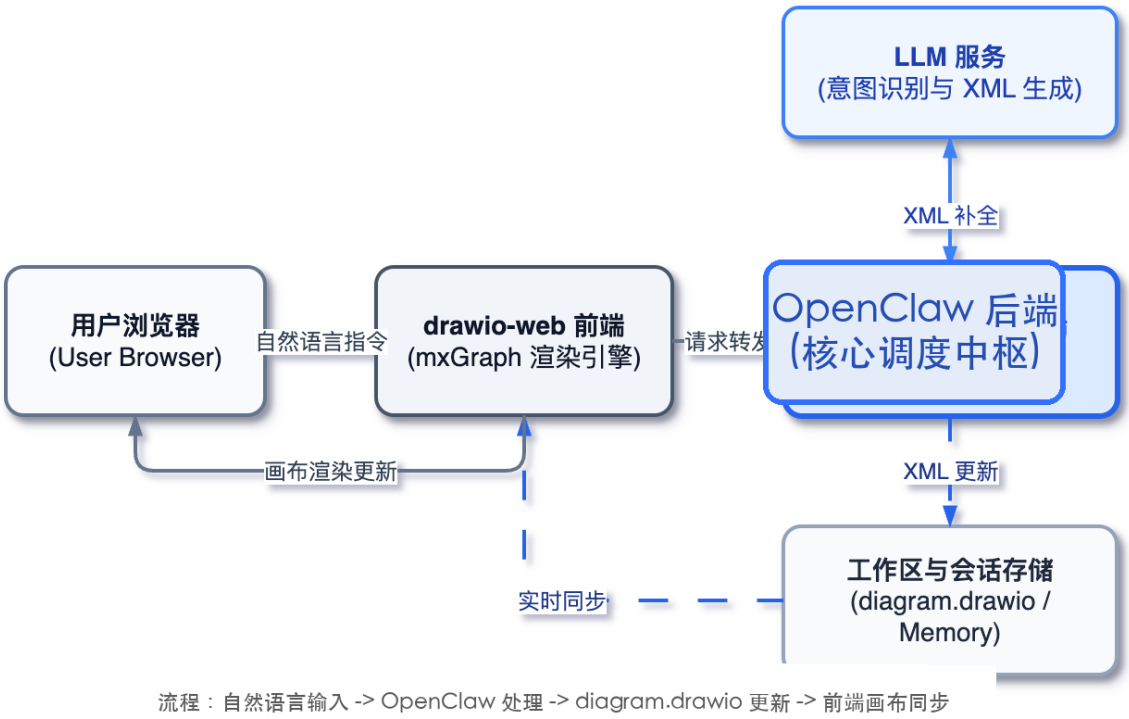

图 1: OpenDrawio Core 系统总体架构示意

7 核心功能

7.1 对话式改图

这是系统的主能力。用户发送文本后，后端根据上下文判断当前轮是否需要改图；如果需要，就在工作区中操作 Drawio XML，并把结果同步回前端画布。该机制让用户可以像与通用聊天助手对话一样处理图表，但最终结果仍然保持可编辑。

这里的关键不是“模型会不会说话”，而是“模型是否能在受控边界内修改图文件”。因此项目始终把工作区限制、文件工具、安全约束和图同步视为主能力的一部分，而不是附属细节。

7.2 多图管理

用户不仅可以处理一张默认图，还可以创建、切换、重命名和维护多张图。前端侧栏负责图集管理，后端负责按照图 ID 隔离工作区和会话，从而支持一个用户围绕多个主题并行工作。

7.3 登录、注册与身份状态

项目提供轻量用户系统。前端通过登录或注册接口获取用户身份和令牌，并持久化到本地存储。这个设计既满足当前多租户场景，又为后续更严格的鉴权机制预留了扩展边界。

7.4 画布同步与保存

前端通过 iframe 嵌入 Drawio 画布。聊天改图后，前端重新拉取 XML 并刷新画布；用户在画布内直接编辑时，前端再把最新 XML 回写后端。这条双向同步链路保证了“AI 改图”和“人工改图”可以在同一套文件上协同。

这也是项目体验能否成立的关键部分。如果只有后端改图而没有稳定的前端同步，用户会感觉结果不可控；如果只有前端画布而没有后端回写，历史记录和工作区状态就会断裂。因此该链路属于核心体验，而不是技术细节。

7.5 流式回复与扩展能力

系统已提供流式对话接口，用于逐步推送文本增量、工具执行状态和图更新事件。除此之外，当前架构也已经为生图、图审查和版本回退等能力预留了扩展入口，因此项目不是单一聊天页面，而是一个可继续扩展的对话式画图工作台。

7.6 模式与多阶段能力

系统支持普通对话、文本澄清、图像参与执行等多种模式，并已经向生图和审查延伸。这说明系统在设计上区分了“澄清问题”“真正改图”“补充图像输入”“生成最终视觉结果”等不同阶段，有利于后续把复杂任务拆成更稳定的流程。

图 2: 用户从登录到改图再到存档的工作流程示意

8 关键业务流程

8.1 登录与进入工作台

用户通过注册或登录接口进入系统，前端获取并保存用户身份信息，然后跳转到对话页。后续所有图文件和聊天请求都绑定该用户。

8.2 聊天改图流程

1. 用户在右侧输入自然语言。
2. 前端将消息和当前图标识发送给后端。
3. 后端结合技能、历史上下文和当前图文件决定是否改图。
4. 如发生改图，后端写入新的 Drawio XML。
5. 前端根据响应结果刷新画布并显示回复。

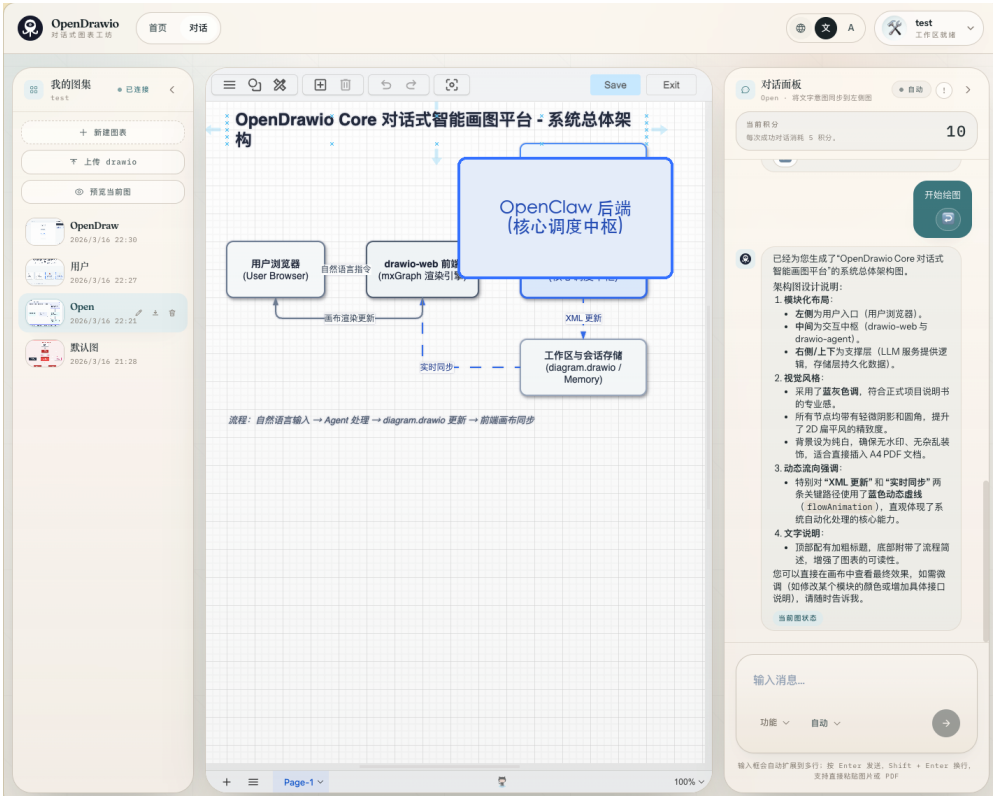

图 3: OpenDrawio Core 三栏式工作台界面示意

在这条流程中，`diagram_updated` 和画布同步结果是一致性关键点。只有当后端真实写入文件且前端成功刷新画布时，用户才会稳定感知到“这条指令已经作用在图上”。

8.3 人工编辑回写流程

1. 用户在 Drawio 画布中直接拖拽或修改。
2. 画布通过 `autosave/save` 事件把 XML 返回给前端。
3. 前端调用图文件保存接口，将最新 XML 写回工作区。

8.4 流式反馈流程

在流式模式下，系统不会等到整段回复结束后再一次性返回结果，而是逐步把文本增量、工具状态和图更新事件传给前端。这样做的价值在于提升响应感知速度，并降低用户在长时任务中的等待不确定性。

9 核心接口

接口	方法	说明
`/health`	GET	健康检查
`/v1/register`	POST	用户注册
`/v1/login`	POST	用户登录
`/v1/chat`	POST	单次对话与改图请求
`/v1/chat/stream`	POST	流式对话请求
`/v1/diagram`	GET / POST	读取或保存当前图 XML
`/v1/dashboard`	GET	返回用户相关统计与项目数据

9.1 接口设计原则

接口设计遵循三项原则：

- 面向图工作流，而不是面向单次文本问答。

- 尽量复用同一张图的上下文、文件和会话信息。
- 保持请求语义简单，使前端、测试和回归脚本都能稳定验证。

10 数据与状态管理

10.1 状态对象

系统运行依赖四类核心状态对象：用户身份、当前图文件、当前图的消息历史，以及前端本地保存的 UI 状态。这些状态对象分布在浏览器、本地文件系统和服务端内存之间，因此设计重点是保持它们的边界清晰、同步明确。

10.2 状态一致性要求

对于用户来说，最重要的不是底层存储方式，而是“刷新后还能不能看到同一张图、同一段历史、同一个登录状态”。因此项目在持久化设计上优先保证跨刷新、跨页面和多图切换的一致性体验。

11 安全与并发策略

11.1 安全边界

当前系统通过工作区隔离、登录注册限流、令牌签名和受控文件访问边界建立基础安全能力。虽然它还不是完整企业安全体系，但已经具备面向多用户使用所需的最基本防护。

11.2 并发与保护

考虑到对话请求常伴随模型调用和文件写入，后端对重操作设置了并发上限和排队超时。这样做的目的不是追求极端吞吐，而是避免在压力增大时把整条链路拖入不可预测状态。

11.3 可观测性

系统为每个请求分配追踪标识，使接口、OpenClaw、工具调用和日志之间可以被串联排查。对于对话式系统而言，这一点尤其重要，因为问题往往不是出现在单一层，而是出现在“输入、模型、工具、文件、前端渲染”之间的链路上。

12 部署与运行

12.1 运行条件

项目本地开发需要 Python 3.11 以上、Node.js 18 以上，以及可用的模型服务密钥。默认情况下，后端运行在 `8181` 端口，前端运行在 `6173` 端口。

12.2 启动方式

推荐使用仓库根目录的初始化脚本统一安装和启动。也可以分别启动前后端，但实际交付时通常仍建议保留统一脚本，便于验证环境一致性和交接。

12.3 部署建议

如果项目继续向更稳定的团队环境推进，建议保持三项原则：统一配置入口、统一验证入口、统一产物目录。这样可以让本地开发、CI 和后续接手机器尽量共享同一套运行假设。

12.4 关键配置

配置项	作用
主模型 API 密钥	用于配置 OpenClaw 主对话能力
主对话模型配置	用于指定主模型与服务入口
工作区根目录	用于保存图文件与项目工作区
数据目录	用于保存会话和用户数据
并发上限	用于限制后端重操作请求数量
排队等待超时	用于控制请求拥塞时的超时边界
签名密钥	用于维护登录令牌和身份安全
开发默认账号	用于本地演示与调试环境快速进入系统

12.5 当前推荐模型组合

结合当前仓库验证记录，主对话和改图推荐使用轻量快速模型，生图和审查使用更适合图像处理的模型。项目架构本身并不强绑定某个供应商，但建议在真实交付时保持主对话、生图、审查三类模型职责清晰，以减少行为漂移。

12.6 运行维护关注点

日常运行需要优先关注四类问题：模型可用性、工作区写入是否成功、前后端同步是否稳定、验证脚本是否保持绿色。这四项直接决定项目是否还能被视为“可用产品”，而不只是“还能启动的代码仓库”。

13 交付成果与验收标准

13.1 当前交付成果

当前仓库交付的不是单一源代码目录，而是一组完整成果，主要包括：

- 可运行的后端服务，用于处理聊天、图文件、会话和身份相关请求。
- 可运行的前端工作台，用于承载图集、画布和聊天交互。
- 面向长期维护的文档体系，包括状态文档、决策文档、任务队列和说明书。
- 面向验证和交接的脚本体系，用于统一运行检查、回归和收尾动作。

13.2 验收标准

本项目的验收不只看“页面能不能打开”或“接口能不能返回”，而是同时看三方面：一是关键功能是否可实际运行，二是自动化验证是否仍然通过，三是文档和状态记录是否与当前代码一致。

因此，项目把功能清单、验证命令和通过状态统一维护在功能验收清单中，同时要求会话结束时回写状态文档。这样的验收方式虽然比单纯演示更严格，但可以显著降低“功能似乎存在、实际无法稳定维护”的风险。

14 测试与交付机制

14.1 统一验证

项目不是只依赖人工试用，而是通过统一脚本验证后端、前端、文档和端到端链路。常规改动后，维护者需要根据改动范围运行对应验证项，确保代码、文档和构建状态一致。

这也是项目工程质量的重要区分点。对于对话式产品，仅靠主观试用很难稳定发现回归，因此必须把自动化验证作为主流程的一部分。

14.2 功能验收清单

`feature_list.json` 是项目功能验收清单。每个功能点都包含唯一标识、描述、验证命令和通过状态。它的作用不是记录想法，而是记录哪些能力已经被真实验证过，因此是判断完成度的核心依据。

14.3 长期接力维护

仓库已经建立面向长期智能协作的接力体系，核心文件包括：

- 会话协议文档：用于约束每轮工作的开始与结束动作。
- 进度记录文档：用于记录最近会话完成情况。
- 当前状态文档：用于记录当前产品与工程快照。
- 下一任务文档：用于明确默认下一任务。
- 决策文档：用于记录重要工程决策。

这一套机制使项目即使跨会话、跨维护者、跨模型，也能保持上下文连续和交付标准稳定。

14.4 交付原则

项目交付遵循三个原则：一是功能完成必须伴随验证，二是重要决策必须写回文档，三是不要在一次会话中同时推进多个无关事项。这些规则直接决定了该仓库能否长期由不同维护者稳定接手。

15 风险与后续方向

15.1 当前边界

当前系统仍然依赖嵌入式 Drawio 画布，文件系统持久化也更偏向单机或轻量部署；用户系统和权限控制仍属于基础版本。此外，多模态和高阶能力仍受外部模型与网关稳定性影响。

这意味着项目虽然已经具备产品雏形和工程底座，但在真正走向更大规模部署时，还需要在画布协议、存储层、鉴权层和高并发治理层继续补强。

15.2 后续方向

后续可重点关注以下方向：

1. 抽取更结构化的规划数据，增强复杂改图任务的稳定性。
2. 进一步解耦第三方嵌入画布，降低前端对外部协议的依赖。
3. 增强图元数据管理和跨页面持久化能力。
4. 在保持可验证性的前提下扩展生图、审查和回退工作流。

16 实施建议

16.1 近期建议

如果以当前仓库为基础继续推进，近期最合适的策略不是盲目扩功能，而是优先稳住主链路：登录、选图、聊天改图、画布同步、保存回写和跨刷新恢复。这些能力一旦稳定，后续所有高级功能才有可靠承载面。

16.2 中期建议

中期应把重点放在结构化能力和可观测性上。一方面，需要进一步提升复杂图任务的规划与执行稳定性，减少模型在长流程中的漂移；另一方面，需要让错误定位、请求追踪和回归验证更标准化，使项目在多人协作或长期演进时仍然可控。

16.3 交付建议

在正式对外展示或进入更稳定的团队使用前，建议把说明书、状态文档、验证脚本和功能验收清单视为同等重要的交付物，而不是把它们看作可有可无的附属文档。对于这类对话式系统，真正决定长期质量的，往往不是单次效果，而是是否能被持续验证、持续定位和持续接手。

17 结论

OpenDrawio Core 当前已经形成一套较完整的对话式画图工程底座。它既能支持用户在浏览器中通过自然语言创建和修改 Drawio 图，也具备面向长期迭代所需的接口边界、验证机制和接力文档。对于需要建设“对话 + 图编辑”工作流的团队而言，该项目已经具备继续产品化和工程化演进的基础。