BIOF3 组学数据分析

AI 辅助编程与智能体工具

导出日期：2026年6月27日

AI 辅助编程与智能体工具

AI 编程工具已经从"补全几行代码"发展到"阅读项目、修改文件、运行命令、检查结果"的智能体工作流。对组学数据分析来说，它们可以显著提高效率，但**不能替代你对数据、统计方法和生物学问题的判断**。

这一篇不追工具热点，目的是建立一套可靠的使用框架：什么时候用 AI、怎么交付任务、如何检查结果、哪些数据不能交给外部服务。

先记住一句话

AI 帮你打字，不替你思考。

打字层面（写一段代码、改一份脚本、生成测试）AI 越来越快，但思考层面（这份数据该不该用、这个统计方法对不对、这条结论能不能写）必须人来。

下面所有内容都围绕这一句展开。

什么是 vibe coding

Vibe coding 指用自然语言描述目标，让 AI 快速生成原型代码或应用。它适合探索想法，例如：

- 快速画一张表达量分布图
- 把一段 R 代码改写成 Python
- 根据报错信息定位可能原因
- 生成一个小型数据清洗脚本
- 搭一个分析报告模板

它的代价也直白：模型默认很多假设，生成的代码看起来能跑，但不一定统计上正确、可重复或适合真实数据。

所以 vibe coding 在 BioF3 的语境里只是"快速草稿"，后续必须补上：

- 明确输入和输出
- 固定软件版本
- 保存参数和随机种子
- 人工检查统计方法
- 用小数据集验证结果
- 把一次性代码整理成可复现脚本

智能体工具和普通聊天的区别

普通聊天工具只回答问题。智能体工具可以连接代码环境，做更完整的开发动作：

- 读取项目文件
- 修改多个文件
- 运行终端命令
- 执行测试或构建
- 根据错误继续修复

- 生成提交说明或 Pull Request

权限越大，风险越高。第一次接入新工具时，**先让它只读**：让它解释项目结构、提出修改计划，确认无误后再允许它真的改文件。

常见工具类型

1. IDE 内置助手

嵌入编辑器，适合日常写代码、解释局部文件、生成函数和查看 diff。

常见形态：

- VS Code 插件
- Cursor 等 VS Code 衍生编辑器
- JetBrains 系列插件
- 云端 IDE 或浏览器工作区

适合做：解释当前脚本 / 补全函数 / 重构局部代码 / 修复语法错误 / 模仿现有风格写一段相似代码。

不适合做：没有上下文的大型分析决策 / 未确认就批量改整个项目 / 直接处理敏感临床数据。

2. Codex

Codex 是 OpenAI 的编码智能体，可以在终端、本地开发环境或相关产品界面中使用。适合读项目、改文件、运行命令、做代码审查和处理多文件任务。

```
codex
```

可以让它做：

```
梳理这个 Docusaurus 项目的目录结构，指出教程内容、静态资源和部署脚本分别在哪里。
```

```
检查 docs/single-cell/module02.md 里的 R 代码块，找出可能无法直接运行的地方，只给建议，不要修改文件。
```

使用建议：

- 先让 Codex 读项目并给计划
- 修改前确认影响范围
- 每次只给一个明确任务
- 改完后运行 build / 测试 / 示例脚本
- 不要把未脱敏数据、密钥、服务器凭据写进提示词

3. Claude Code

Claude Code 是 Anthropic 的编码智能体，可以在终端、IDE、桌面应用和浏览器中使用。能读代码库、编辑文件、运行命令，并和开发工具集成。

```
claude
```

适合做：解释陌生代码库 / 根据报错追踪问题 / 编写测试 / 批量修复 lint 或格式 / 整理项目文档 / 通过 `CLAUDE.md` 固定项目规则。

在生信项目中，可以这么用：

```
阅读这个 R 脚本，解释每一步在单细胞分析流程中的作用，并指出哪些参数需要根据数据集调整。
```

4. opencode

opencode 是开源的 AI 编码智能体，主打终端工作流，也提供桌面和 IDE 形态。可以连接不同模型供应商，适合希望掌控工具栈和模型来源的开发者。

```
opencode
```

适合做：本地项目问答 / 生成修改计划 / 实施局部功能 / 维护项目级 `AGENTS.md` / 在多个模型之间切换。

特别注意：API key 千万不要写进仓库，不要提交 `.env` 。

5. Kiro

Kiro 是偏"规格驱动开发"的 AI IDE。先把需求、设计、任务拆清楚，再让智能体执行。相比纯 vibe coding，它更适合把原型推进到可维护项目。

核心概念：

- **specs**：把需求、设计、任务写成结构化文档
- **steering**：给项目提供长期规则和上下文
- **hooks**：在保存、创建或删除文件时触发自动化任务
- **agentic chat**：通过自然语言和项目交互

适合做：从想法生成需求说明 / 把功能拆成可检查任务 / 生成实现计划和测试计划 / 维护中大型项目的一致性。

如果只是临时画一张图，Kiro 偏重；如果要长期维护一个网站、分析平台或工作流，它的规格驱动思路有价值。

6. API 中转站和模型聚合服务

很多用户会接触到"中转站"、"转发站"或"模型聚合服务"。它们提供统一 API，把请求转发到不同模型供应商。

优点：一个接口访问多个模型 / 支付和额度管理可能更方便 / 有些服务提供兼容 OpenAI 格式的接口。

风险：

- 数据会经过第三方服务
- 稳定性和响应速度不可控
- 模型版本、价格、上下文长度可能变化
- 数据保留策略可能不透明
- **不适合**处理未公开论文、临床数据、密钥、商业代码

建议：能用官方 API 或企业账号时优先用 / 不把真实患者数据、访问密钥、服务器密码交给不明中转服务 / 如必须使用中转，先做脱敏和小样本测试 / 在项目文档中记录模型、供应商、日期和关键参数。

生信分析中的安全边界

AI 工具擅长写代码，但**不理解真实实验约束**。下面这些事情必须由人来确认：

- 样本分组是否正确
- 统计检验是否匹配实验设计
- 批次效应是否需要处理
- marker gene 是否符合生物学背景
- 过滤阈值是否合理
- 可视化是否夸大结论

- 结果是否可重复

对涉及人类样本、临床数据、未公开项目的数据，**先默认不能上传到外部 AI 服务**。至少要做：

- 去除姓名、编号、地址等直接标识符
- 去除可回溯到个体的元数据
- 不上传原始 FASTQ、BAM、全量表达矩阵
- 只提供最小可复现示例
- 改用本地模型、企业合规服务或脱敏后的样例数据

推荐工作流

学习阶段：把 AI 当解释器，不是代写器

用初学者能理解的方式解释这段 `Seurat` 代码。请逐行说明输入、输出和关键参数。

我不理解 `NormalizeData`、`FindVariableFeatures`、`ScaleData` 的区别。请结合单细胞表达矩阵解释。

分析阶段：AI 出草稿，保留人工审查

根据这个数据框结构，写一段 `ggplot2` 代码画不同细胞类型的基因表达小提琴图。请不要假设不存在的列名。

下面是报错信息和 `sessionInfo`。请判断最可能的原因，先给排查步骤，不要直接改代码。

项目维护阶段：让 AI 做重复劳动

检查 `docs/basics` 目录下的教程，找出标题层级不一致、图片路径可能错误、代码块语言未标注的问题。

为这个分析脚本生成 `README`，说明输入文件、输出文件、依赖包和运行命令。

提示词模板

解释代码

请解释下面这段代码。要求：
1. 说明每一步的目的
2. 标出输入和输出
3. 指出可能需要根据数据修改的参数
4. 不要重写代码，除非发现明确错误

生成分析脚本

请写一个可复现的 R 脚本完成以下任务：
- 输入：counts.csv 和 metadata.csv
- 输出：QC 图、标准化后的对象、marker 基因表
- 要求：固定随机种子，记录 sessionInfo，所有输出写入 results/
- 不要使用不存在的列名；如果需要列名，请先向我确认

审查结果

请作为代码审查者检查这段分析流程：
1. 是否有统计学问题
2. 是否有不可复现的步骤
3. 是否有硬编码路径
4. 是否遗漏中间结果保存
5. 是否需要补充图注或方法说明

常见坑

坑 1：AI 编出"看起来对"的列名

让 AI 写一段处理 metadata.csv 的代码，它经常猜列名（"sample"、"group"、"condition"），生成看似可运行的代码，跑起来 Error in $: object 'group' not found 。

避免：把真实列名（或 head() 输出）一起喂给 AI，明确告诉它"不要假设不存在的列名"。

坑 2：AI "幻觉"出不存在的函数 / 包

特别在跨语言（R ↔ Python）翻译时，AI 会编出根本不存在的函数。代码看起来很对，跑起来 could not find function 。

避免：每次拿到 AI 写的代码，**先把所有 library() / import 列出来核对一遍**，再跑。

坑 3：把临床数据原文贴进对话

学生最常见的事故。"老师让我分析这份病人数据"，复制粘贴整张表到 ChatGPT，姓名、住院号、确诊日期一并送出。

避免：在做任何分析前，先脱敏一份"开发用样例"，所有 AI 交互只针对样例。真实数据只在本地分析。

坑 4：把 API key 写进代码，提交到 git

```
# 错误示范
client <- httr2::request("https://api.openai.com/v1/...") |>
  httr2::req_headers(Authorization = "Bearer sk-xxxxxxxxxxxx")
```

提交 git → 推到 GitHub → 公开 repo → key 被爬虫扫到 → 余额清零。

避免：所有 key 放 .env 或环境变量， .env 加进 .gitignore 。

坑 5：让 AI "一次性写完整套分析"

提示词："写一个完整的单细胞 RNA-seq 分析脚本，包括 QC、标准化、聚类、注释、差异分析、富集"。AI 真的会给你 200 行代码，但每一步都用了它"觉得最常用"的参数 / 阈值 — **不一定适合你的数据**。

避免：拆成"按一步一步"。每一步：先让 AI 解释这一步在做什么，再让它写代码，再人工跑一下，看输出。然后才进下一步。

工具选择建议

场景	更适合的工具
解释一段代码	ChatGPT、Claude、IDE 助手
修改本地项目多个文件	Codex、Claude Code、opencode
快速原型	Codex、Claude Code、Cursor、Kiro
规范化长期项目	Kiro、Codex、Claude Code
多模型切换	opencode、模型聚合服务
敏感数据分析	本地部署的开源模型（Ollama 等）/ 企业合规服务 / 完全离线工具

下一步

接着深入（按推荐顺序）：

1. 编程基础：R / Python / Bash — 把 AI 帮你写的代码看懂、改对，前提是你自己会一点
2. 数据与环境准备 — 先把环境装好，AI 写的代码才能在你机器上跑

横向延伸：

- Jupyter 与交互式分析环境 — 想用 Notebook 跟 AI 一起探索数据时
- 公共数据库与数据检索 — AI 推荐数据集前你先要懂这些库

参考资料

- OpenAI Codex CLI：https://developers.openai.com/codex/cli
- OpenAI Codex 使用说明：https://help.openai.com/en/articles/11369540/
- Claude Code 文档：https://code.claude.com/docs
- opencode 文档：https://dev.opencode.ai/docs
- Kiro 文档：https://kiro.dev/docs/
- Kiro 规格驱动开发介绍：https://kiro.dev/blog/introducing-kiro/