RedNote-Agent 技术报告

0. 基本信息

- 项目名称：RedNote-Agent | 小红书图文生成智能 Agent
- 项目目标：基于结构化商品信息，自动生成小红书风格的图文文案与封面图，支持多语气风格与多商品批量处理。
- 技术栈：
  - 编程语言：Python 3.12
  - Agent 框架：LangGraph（基于 LangChain）
  - 文本模型：Kimi（通过胜算云路由调用，兼容 OpenAI 接口）
  - 图像模型：doubao-seed（通过胜算云异步任务 API 调用）
  - 其他：Pillow（本地封面渲染和文字叠加）、requests、python-dotenv

项目入口：main.py
核心逻辑目录：agent/

1. Agent 架构设计

1.1 整体架构概览

RedNote-Agent 采用「轻量工作流 + 外部工具」的 Agent 架构，而不是典型的"对话式工具调用 Agent"。核心设计思路：

- 用 LangGraph 建模为一个**有向工作流图（StateGraph）**；
- 把每一步抽象为「节点函数」，共享一份结构化状态 AgentState；
- 外部模型（Kimi / Gemini）由节点内部自己调用，Agent 不再在"每一步都让大模型自行规划"，而是让架构层面明确控制顺序和职责。

整体流程：

1. **数据加载 & 编排：**
   - agent/processor.py 从 inputs.json 读取商品列表，构造初始 AgentState，并依次调用工作流。
2. **内容生成节点（generate_content）：**
   - agent/content_generator.py 负责把结构化商品信息 → 小红书文案 JSON（标题、正文、标签）。
3. **封面生成节点（generate_cover）：**
   - agent/cover_generator.py 负责：
     - 用 Kimi 生成英文图像提示词；

        - 用胜算云 doubao-seed 模型异步生成图片；
        - 如果失败，则退回到本地 Pillow 封面生成；
        - 最后叠加商品名与标题，输出封面路径。
4. **结果落盘：**
    - results.json：记录每个产品的 product_id、title、content、tags、封面文件名；
    - covers/：保存 PNG 封面图。

1.2 工作流实现细节

工作流定义位于 agent/agent.py：

- 使用 StateGraph(AgentState)建模，有两个节点：

  Entry: generate_content

  ↓

  Node: generate_cover

  ↓

  END

- AgentState 定义在 agent/state.py：
    - product: dict—— 当前处理的商品信息；
    - title: str—— 生成的小红书标题；
    - content: str—— 图文正文；
    - tags: list[str]—— 小红书标签列表；
    - cover_path: str—— 最终封面图文件路径；
    - error: str | None—— 错误信息（用于中断后续节点）。

1.3 架构选型及调整

采用的架构：

- 选择 LangGraph 的 StateGraph：
    - 原因：
        - 项目流程清晰、步骤有限，适合「有向有序节点」模型；
        - 可视化思路清晰，评审易理解；
        - 未来扩展（加入审核节点、自我修正节点）非常自然，只需新增节点与边。

相较于传统“单 Prompt 多轮对话 Agent”的调整：

1. **明确的阶段划分**
    - 把“生成文案”和“生成封面”拆成两个独立节点，而非在一个大 Prompt 里让模型自己决定是否生成图片提示词。
    - 好处：
        - 每一步的输入输出结构明确，便于调试和日志记录；
        - 出现问题可精准定位在某一节点。
2. **强类型状态管理**
    - 通过 TypedDictAgentState 限制状态字段，不允许随意添加/覆盖未知 key。
    - 避免了“模型返回的字段名不一致、后续节点读取失败”的隐性 bug。
3. **工具调用下沉到节点内部**
    - 调用 Kimi 与 Doubao 的细节隐藏在节点内部，例如 agent/kimi_client.py、agent/image_generator.py。
    - Agent 层只关心“节点完成了什么”，不关心“怎么调用模型”。
4. **本地回退机制内置在工作流里**
    - 封面节点内建了“远程模型失败 → 本地 Pillow 回退”的机制，避免整体任务因为单点模型故障而失败。

2. Prompt 策略揭秘

本项目的 Prompt 设计主要分为两块：

1. **文案生成 Prompt（小红书图文）**
2. **封面图像提示词 Prompt（英文 image prompt）**

2.1 文案生成 Prompt 策略

对应文件：agent/content_generator.py
系统提示（System Prompt）设计要点：

- 定位角色：明确告诉模型“你是一位专业的小红书内容创作者”；
- 明确输出结构：要求“以 JSON 格式返回，包含 title / content / tags”；
- 明确写作规范：
    - 标题：疑问句/感叹句/数字，15–25 字；
    - 正文：短句分段、口语化、像朋友聊天、多 emoji、突出卖点；
    - 标签：3–5 个相关标签；
    - 语气：根据 tone 映射为具体描述（例如“温暖、治愈、像朋友般贴心”）。

关键策略 1：语气风格映射表
代码中使用了一个 tone_map：

- 温馨治愈→“温暖、治愈、像朋友般贴心的语气”
- 活泼俏皮→“活泼、可爱、充满活力的语气”
- 专业测评→“专业、客观、详细的测评语气”

- 种草安利→“热情、推荐、真诚分享的语气”
- 简约高级→“简洁、高级、有品质感的语气”

这样做的好处是：

- 前端/输入侧只需要传一个枚举值（tone），
- Prompt 里则给出对模型友好的自然语言解释，减少模型误解语气的概率。

关键策略 2：强约束 + 反向约束

- 正向约束：详细列出内容结构和写作要求（标题长度、语气、emoji 使用等）；
- 反向约束：显式禁止“最、第一、100%”这类绝对化用语，贴合平台规范。

关键策略 3：JSON-only 输出 + Markdown 清洗
模型经常会输出带 ```json 代码块的内容，因此在解析前做了两步清洗：

1. 如果包含 json …，只取里面的内容；
2. 否则如果包含…，也只取代码块内的部分。

再将该字符串 json.loads 为 dict，填回 state 中。
这一步极大降低了“模型多输出解释文字导致 JSON 解析失败”的概率。

2.2 封面图 Prompt 策略

封面 Prompt 由 agent/cover_generator.py 中的 generate_cover_node 生成。
核心思路：

- 角色设定：
  - 模型被设定为“AI 图像提示词工程师”，职责是输出适合 doubao-seed 图像模型的**英文 Prompt**；
- 输入信息：
  - 产品名称、类别、核心卖点；
  - 标题文案、风格调性（tone）；
- 输出要求：
  - 使用英文描述；
  - 主体突出、视觉具体（颜色、材质、光线、构图）；
  - 强调氛围和小红书风格；
  - 指定 3:4 竖版构图；
  - 明确要求不要在图片中包含文字、标签、数字。

结构化提示词模板：

- 模板明确给出了建议结构：

  [主体物品描述], [场景环境], [光线色调], [整体氛围], [艺术风格], professional product

photography, high quality, 3:4 aspect ratio

这样做的好处：

- 降低模型输出过长散文式描述的概率；
- 增强 Prompt 的可预测性，便于和图像模型配合调参。

Markdown 清洗策略：

- 和文案类似，封面提示词也对 ``` 包围的内容做了清洗，避免模型用 Markdown 包起来导致多余符号进入图像模型。

3. Bad Cases 分析

开发与测试过程中遇到的典型坑点及解决方案如下。

3.1 模型返回的 JSON 不干净

现象：

- 文案模型返回时，经常出现如下形式：

  ```json

  {

  "title": "...",

  "content": "...",

  "tags": [...]

  }

- 直接 json.loads(response.content)会报错。

**解决：**

- 在 generate_content_node 中增加了 Markdown 清洗逻辑：
  - 如果包含 ```json，则截取其中内容；
  - 否则如果包含 ```，也只取代码块内部；
  - 再进行 json.loads。
- 同时 try/except 捕获异常，将错误信息写入 state["error"]，后续节点根据 error 进行短路。
  ```

3.2 图像生成 API 不稳定或响应格式异常

对应文件：agent/image_generator.py
现象：

- 图像生成是异步任务：
  - 先提交任务拿到 request_id；
  - 再轮询查询状态；
- 可能出现：
  - HTTP 状态码非 200；
  - code != "success"；
  - data 字段为空；
  - image_urls 为空；
  - 任务状态长时间处于非 COMPLETED。

解决：

1. **完善调试日志**
   - 对提交和查询阶段都打印状态码与部分响应体，方便定位问题；
   - 在关键失败分支上打印明确的错误原因。
2. **严格的字段检查**
   - 检查 code == "success"、data 非空、request_id 存在、image_urls 非空等。
3. **超时与失败处理**
   - 轮询最多 60 次，每次 2 秒，总共最多等待 120 秒；
   - 若超过时间或状态为 FAILED，统一返回 False。
4. **在封面节点实现降级回退**
   - generate_cover_node 中，如果 generate_image_with_api 返回 False：
     - 自动退回本地 generate_cover（Pillow 渲染）；
     - 确保整条 pipeline 不因图像 API 不稳定而完全失败。

3.3 文本在封面图上的排版溢出 / 模糊

现象：

- 标题过长时，如果简单居中绘制，很容易出现：
  - 文本溢出图片边界；
  - 文本与背景颜色对比度不足，导致难以阅读。

解决：

1. **自适应换行算法**
   - 使用 wrap_text_by_width 按像素宽度包行，而不是按字符数；
   - 支持中文/中英文混排，优先在标点处换行；

    - 限制最大行数（3 行）以防过多文字挤满画面。
2. **自动寻找“清晰区域”**
    - 通过 ImageStat 统计不同区域的灰度方差，选择纹理较平滑的区域放标题（噪声越小，文字越清晰）。
3. **描边与对比增强**
    - 标题与商品名在叠加时使用描边（stroke）与合适的配色，保证在复杂背景上也能看清。

3.4 Windows 控制台中文输出乱码

对应位置：agent/processor.py
现象：

- Windows 控制台默认编码不是 UTF-8，中文日志可能出现乱码。

解决：

- 在程序开始处检测 sys.platform == 'win32'，执行：
    - chcp 65001 切换控制台编码为 UTF-8；
    - 使用 sys.stdout.reconfigure(encoding='utf-8', errors='replace') 强制输出为 UTF-8。

4. 模型评价

本项目主要使用两个模型：

1. Kimi（经胜算云路由，兼容 OpenAI Chat API）
2. doubao-seed（通过胜算云异步任务 API 调用）

4.1 Kimi 模型评价（用于文案与封面 Prompt）

优点：

- 中文能力优秀：
    - 对小红书风格、口语化、emoji 使用等理解到位；
    - 支持长上下文，能很好利用商品的结构化信息（特点、卖点、目标人群）。
- 可控性较好：
    - 在明确给定 JSON 输出要求和字段名的前提下，绝大部分返回都符合结构化要求；
    - 通过语气描述 +反向约束，能显著减少“广告法禁用词”的出现频率。
- 与胜算云路由集成简单：
    - 使用 langchain_openai.ChatOpenAI 即可接入，配置 API Key 和 Base URL 即可切换不同模型。

不足与限制：

- 对“绝对化用语”并非 100% 避免：
  - 即使在系统 Prompt 中禁止 “最、第一、100%”，在少数样本中仍可能出现类似词汇，需要额外审核或正则过滤。
- 偶尔会输出解释文字/Markdown 包裹：
  - 需要在代码侧做 Markdown 清洗与异常处理。

4.2doubao-seed 模型评价（用于封面生成）

优点：

- 画面质量高：
  - 对“竖版 3:4 构图、社交媒体封面”类场景表现良好；
  - 对氛围、光线、艺术风格等描述有较强理解力。
- 与提示词结构配合度较好：
  - 在使用结构化 image prompt 的情况下，能够明显体现主体、背景和光线的差异。
- 通过胜算云的异步 API 调用：
  - 支持任务 id、轮询查询、多张图片等高级能力；
  - 适合批量生成与队列控制。

不足与限制：

- 生成时延相对较高：
  - 单张图从提交到完成可能需要数十秒，需要通过异步或加速播放方式提升用户体验。
- API 调用复杂度高于简单的同步接口：
  - 需要处理提交任务、查询任务、解析异步数据结构等；
  - 出错场景比简单 HTTP 请求多，需要比较多的防御性编程。
- 在文字/Logo 等精细排版上，可控性不如手工设计：
  - 本项目采取的策略是：**先用模型生成背景图，再用 Pillow 在本地叠加文字与主题元素**，以获得更稳定可控的 UI 效果。

5. 总结与展望

5.1 项目特点总结

- **架构清晰**：
  - 使用 LangGraph 的 StateGraph 将流程拆分为明确的节点，状态结构化，可视化程度高，便于评审快速理解和后续扩展。
- **Prompt 策略可复用**：

- 文案 Prompt 通过语气映射、正反向约束和 JSON-only 输出要求，实现了较高的可控性；
- 封面 Prompt 将“图像提示词工程师”的角色与结构化 image prompt 模板结合，便于对接各种图像模型。

- **鲁棒性好：**
  - 针对 JSON 解析失败、图像 API 不稳定、Windows 编码等 Bad Cases 都有防御性处理与回退机制；
  - 封面生成有远程模型 + 本地 Pillow 双路径保障。
- **易于工程落地：**
  - 使用 main.py 作为标准入口，agent/作为核心逻辑包，requirements.txt 明确依赖，便于打包部署与团队协作。

5.2 后续可扩展方向

1. **加入自我修正节点**
   - 在内容生成和封面生成之间增加“自审核节点”：
     - 校验是否存在敏感词、绝对化用语；
     - 对标题长度、标签数量等进行自动修正。
2. **多模型 A/B 测试能力**
   - 在相同 Prompt 下，对比不同模型（如不同 Kimi 版本、不同图像模型）的表现，自动选优结果。
3. **成本统计与配额控制**
   - 在调用胜算云接口时增加调用计数与 Token 估算，自动生成成本报表；
   - 在超出预设配额时自动降级到更便宜模型或本地方案。
4. **小红书平台规则适配增强**
   - 针对违规词、营销用语、sensitive 话题做更系统的规则库与模型审查，进一步提升“可直接上线”的程度。