教育领域舆情动态监测和简报自动化系统的开发实践与经验总结

洪煜

2026 年 8 月 13 日

2026

01 · 案例背景

项目做什么：开展舆情监测，帮助工作人员及时研判

网络信息数量大、变化快，同一事件还会被多个平台重复转发。系统定时收集公开舆情信息，先做初步整理和分析，再由工作人员核实。

项目目标

更快发现需要关注的舆情变化，减少人工收集和重复整理的时间。系统保留信息来源和修改记录，最终判断仍由工作人员作出。

公开交流仅描述通用流程；业务对象、委托关系和具体内容均已省略。

02 · 项目产出

项目产出了什么：一个平台，两类工作材料

系统平台

01 动态看板：今天收集了多少信息，有何变化

02 复核工作台：查看原文，确认机器判断是否准确

03 规则设置：调整监测范围和判断标准

04 报告工作台：生成、预览、审定和保存简报

舆情简报

- **定期简报** 按日或按周总结整体情况
- **专题简报** 围绕一个主题梳理事实和变化过程
- **多种格式** 网页阅读、 PDF 保存和纯文本导出
- **审定记录** 保留草稿、修改和审定过程

平台和简报使用同一批数据，因此页面上的数字、报告里的文字和保存下来的文件可以相互核对。

报告内容在交流版中仅展示方法与结构，不展示具体信息。

03 · 问题意识

我们真正关心的问题：怎样让智能体开发过程可靠？

重点不是智能体能写多少代码，而是它能否准确理解需求、使用可信数据，并在出错时留下补救办法。

怎样把需求说清楚？

先连续追问，再写成一份可以逐项检查的说明。说明中还要明确哪些事情暂时不做。

怎样避免编造事实？

模型主要负责归纳和写作。数据由程序计算，个人信息和操作权限由系统控制。

三种方法怎么选？

让三种工具使用同一批数据，再比较它们适合什么任务、容易在哪里出错。

比较重点：工作步骤是否清楚、事实是否可靠、出错后能否补救，以及人工需要花多少时间复核。

方法：回顾性案例研究；证据来自仓库内规格、ADR、测试与提交记录。

04 · 项目样本

案例边界：短周期、高约束、可持续运行

15 天	142	12	25
集中开发时段	代码修改记录	重要设计说明	数据结构调整

项目规模

数据与服务	205 个程序文件
用户界面	22 个页面程序文件
自动检查	86 个测试文件
报告工具	7 套专用工作说明

约束条件

- 单机可维护
- 外部模型只能看到完成筛选的数据
- 发布报告、修改记录等操作都要人工确认
- 生成失败必须回退，不能阻断日常产出

计数口径：当前 main 分支；“测试”按测试文件计，不等同于测试用例数。

05 · GIT 历史

研发演进：沿着一条可运行主链逐步加深

1 起步

先让数据进得来

7/29—7/30

搭好数据库、服务端和页面；用固定样本检查收集与去重。

2 成链

做出基本工作流程

7/31—8/2

加入话题归类、趋势分级、人工复核和初版简报。

3 加深

让报告更可靠

8/2—8/7

加入 LangGraph 、统一报告格式、历史比较和人工审定。

4 扩展

增加问答功能

8/8—8/10

用 jcode 和 MCP 数据查询窗口试做报告、复核与连续问答。

5 运行

准备长期使用

8/11—8/12

加入定时运行、权限限制、备份恢复和只读文件分享。

每一阶段都先做出一个能实际使用的小版本，再继续增加功能。这样更容易及时发现误解和错误。

时间与阶段依据：2026-07-29 至 2026-08-12 的 Git 提交记录。

06 · 功能地图

系统由哪些部分组成：从收集信息到形成简报

信息收集

按时段获取新信息，删除重复内容，补采遗漏

公开信息进入系统

初步分析

判断信息与主题是否相关，把相近内容归在一起

规则与模型配合

人工复核

查看原文证据，确认、修改或撤销机器判断

人作最终决定

简报生成

固定数据后，由三种工具分别组织文字

网页 / PDF / 文本

操作界面

看板、规则设置、问答和报告审定

工作人员使用

运行保障

定时启动、限制权限、备份和故障恢复

长期稳定运行

各部分通过统一数据表和报告格式连接。即使更换模型或生成工具，已经确认的信息和工作记录仍然保留。

模块归纳依据：backend/app 、 frontend/src 、 skills 、 ops 与 ADR 。

07 · MATT POCOCK 工作流

开发方法：先把问题问清楚，再开始写程序

01

grill-with-docs

对照现有材料连续追问

弄清问题

→

02

to-spec

写成可逐项检查的需求说明

记录取舍

→

03

to-tickets

拆成较小任务，标明先后顺序

说明完成标准

→

04

implement

先检查，再实现，审核后合并

逐步交付

智能体不再听到一句需求就直接写代码。讨论记录、需求说明和小任务都可以由人检查。

流程名称沿用项目团队采用的 Matt Pocock 开发范式；具体约束由仓库文档和 PR 工作流落实。

08 · 方法工具箱

Matt Pocock skills： 一套给智能体使用的工作说明

skill 可以理解为写给智能体的操作手册。这组开源手册把“讨论需求、写说明、拆任务、做实现”分成几个连续步骤。

项目的实际顺序：连续追问 → 写需求说明 → 拆成小任务 → 逐个实现。每个小任务都先写检查办法，再写代码。

官方仓库：github.com/mattpocock/skills

工具定位依据：Matt Pocock skills 官方 README 。

09 · 快速开始

Codex 安装：从零开始的三个步骤

01 安装 Node.js 长期支持版

打开 Node.js 官网，下载并按提示安装。安装后重新打开终端。

```
node --version
npm --version
```

Node.js 下载

02 安装 Codex

在终端复制下面的命令并回车。Windows 建议在 WSL 中使用。

```
npm install -g @openai/codex
```

官方 npm 包

03 检查并启动

先查看版本；能显示版本号即为安装成功。进入项目文件夹后运行 codex 。

```
codex --version
codex
```

Codex 官方仓库

安装工具和连接模型是两件事： Codex 安装成功后，仍需登录可用账户，或按下一页说明接入 DeepSeek 。

大陆网络环境优先使用 npm 或 GitHub Releases ； 避免依赖 chatgpt.com 安装脚本。

10 · 快速开始

安装 skills，并配置大陆网络下的备用模型

Codex 与其他智能体

1 安装并选择所需 skills

```
npx skills@latest add mattpocock/skills
```

2 每个仓库运行一次初始化

```
/setup-matt-pocock-skills
```

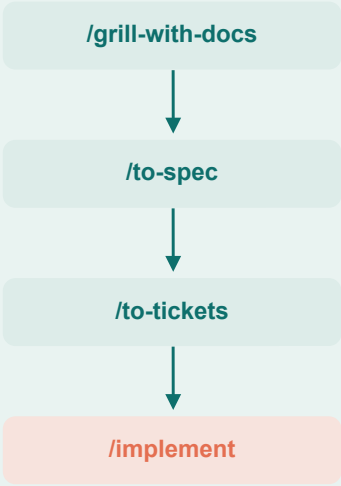
3 网络条件受限时： Codex + DeepSeek

```
export DEEPSEEK_API_KEY=<your-key>
```

接口地址和模型按官方页设置

官方集成说明

本项目的调用顺序



不要同时安装插件版与可编辑文件版，否则同一 skill 会重复出现。

安装说明

命令核验时间： 2026-08-13； DeepSeek 配置以官方页面的最新说明为准。

10 · 需求讨论

grill-with-docs： 在动手之前把问题问明白

四类追问

1 最后由谁作出判断?

2 哪些信息可以交给外部模型?

3 自动生成失败后怎么办?

4 这一阶段明确不做什么?

文档提供上下文，质询暴露冲突。

问题一

简单判断可以直接调用模型；复杂报告才需要分步骤处理。

不同任务用不同方法

问题二

智能体需要查询数据，但不应直接进入数据库。

设置受控查询窗口

问题三

自动生成节省时间，也可能重复放大错误。

保留人工审定和备用方案

经验：需求说明还要写清楚“这次不做什么”，否则范围会不断扩大。

实例依据： ADR 中的适用范围、未采纳方案、失败降级与 PII 边界。

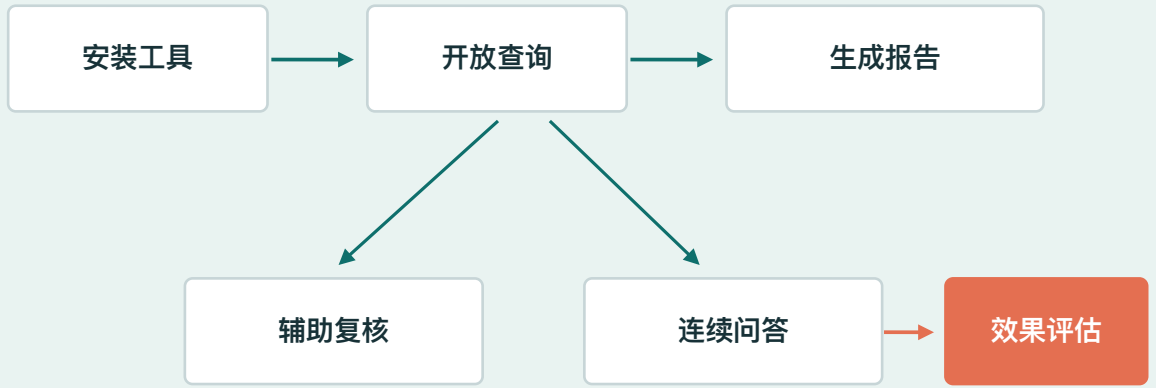
11 · 任务切分

从需求说明到任务清单：把大问题拆小

需求说明至少写清

- **理由** 为什么这样做
- **范围** 哪些事情保持不变
- **规则** 接收什么，产出什么
- **失败** 出错后怎么办
- **完成** 怎样确认已经做好

小任务的先后关系



把任务拆小并标明先后顺序，某一步失败时只需修改局部，也方便换人或换智能体继续工作。

项目采用"一张票、一条短分支、CI 通过后合并"的增量工作流。

12 · 实施过程

逐步实现：每完成一小步，就立即检查

规格 /ADR	测试	实现	审查 /CI	上线反馈
先写决定与杀死判据	fake provider 与契约测试	保持边界可替换	标准与规格双重核对	把真实故障写回契约

实际使用中发现的问题

跨越零点时漏收信息 → 明确统计截止时间
生成期间数据变化 → 固定本次报告所用数据

问答结束后连接未关闭 → 补充资源清理
定时任务失败 → 加入重试和恢复办法

这些修正说明，需求说明需要根据实际运行不断补充，不能只在开发开始时写一次。

证据：相关修复提交、测试文件与运行手册。

13 · 工作原理

报告怎样生成：先整理事实，再让模型写作

收集并检查 持续接收新信息 · 去重 · 检查是否漏收

↓

整理报告材料 分类 · 汇总数字 · 固定本次数据 · 标注来源

↓

组织文字 LangGraph / jcode / Pi + skill

↓

审定与保存 统一报告格式 · PDF/ 文本 · 人工审定 · 留下记录

模型不能修改统计数字

查询工具默认不能改数据

发布和修改必须由人确认

结构依据：报告生成、统一文档、会话运行时与自动化运行手册。

14 · 简报自动化

三种写报告的方法，共用同一批已核实数据

报告材料

- 明确统计时段
- 已经核对的数字
- 信息来源
- 固定版本编号

LangGraph

显式图与回退

jcode

外部运行时 + MCP

coding agent + skill

步骤化操作协议

统一报告格式

网页预览	鉴权
PDF	归档
TXT	机器与人可读

先固定材料，再比较三种写作方法。

统一边界使三种方法比较的是编排方式，而不是数据口径或排版差异。

15 · 工具概览

三类工具分别做什么

LangGraph

固定步骤的生成工具

可以把写报告过程规定为几个步骤，并设置检查和备用方案。适合格式稳定的日常简报。

安装

```
pip install -U langgraph
```

官方文档 / 仓库

jcode

占用资源较少的智能体

可以通过 MCP 这个受控查询窗口读取信息。除写报告外，也适合做连续问答机器人。

安装

```
curl -fsSL https://jcode.sh/install
| bash
```

官方文档 / 仓库

Pi Coding Agent

可加载专用手册的智能体

可以读取高度定制的 skill，按规定步骤取数、判断、写作和检查。适合要求较高的专题报告。

安装

```
npm install -g --ignore-scripts
@earendil-works/pi-coding-agent
```

官方文档 / 仓库

安装命令与链接核验时间：2026-08-13；jcode 的内存优势按项目实测与官方定位陈述。

16 · LANGGRAPH

路径一： LangGraph 适合生成结构化简报

优势

节点、条件边和失败路径明确；可注入 fake 测试“调用了什么、走了哪条路”。

代价

多轮调用增加延迟；状态结构和工具协议需要持续维护。

适用条件

任务边界稳定、工具少而可信、输出契约明确，并且需要确定性回退。

项目实现：小图编排；图失败后依次降级到单步生成与确定性模板。

17 · JCODE

路径二： jcode 同时承担报告生成与信息问答

外部运行时边界

显式 skill · 最小工具白名单 · 退出码契约 · 防重入锁

优势

内存占用较小，适合常驻问答机器人；可通过 MCP 组合多轮查询。

代价

部署、生命周期和工具权限更复杂；自主循环带来非确定性。

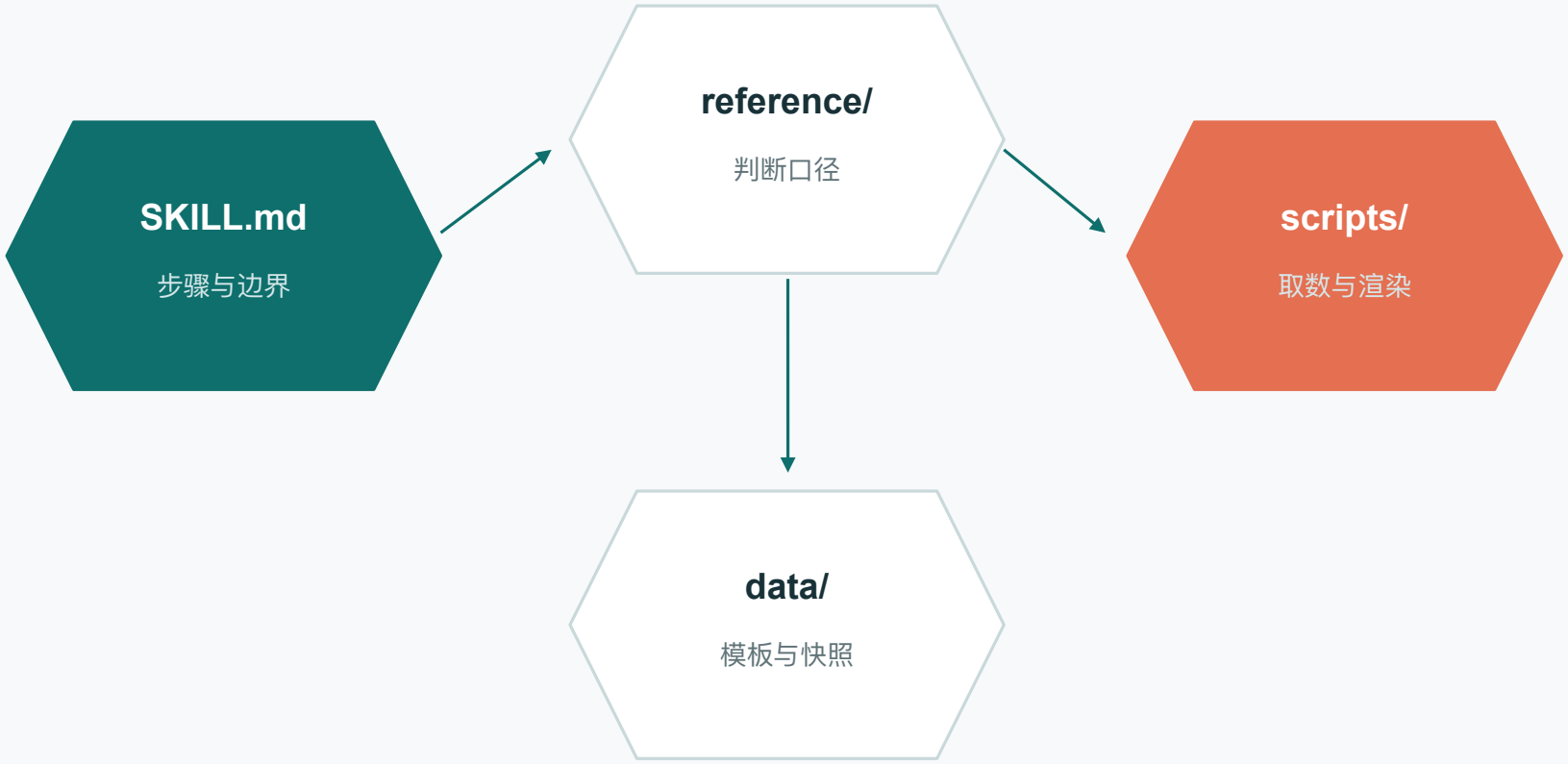
项目分工

报告生成用于验证通路；持续价值来自问答机器人、多轮复核与会话分析。

项目以功能开关保持可撤回，并将数据库访问收口到 MCP 工具层。

18 · SKILL 驱动

路径三： Pi + 专用 skill 生成高质量报告



为什么能工作

- 指令与代码同版本
- 可按材料厚度调整
- 可移交给不同 agent
- 适合生成 HTML/PDF

弱点：执行过程依赖 agent 判断，复现性低于显式图。

项目通过定时服务调用低权限 coding agent，并以 skill 规定取数、组稿、渲染和自检步骤。

19 · 比较结果

三种路径服务不同场景，也有不同的失控方式

比较维度	LangGraph	jcode	coding agent + skill	优先选择
流程可预测性	高	中	中低	LangGraph
多轮 / 会话任务	有限	强	依环境	jcode
开放式组稿	中	中	强	skill
部署复杂度	低—中	高	中	LangGraph
跨 agent 可移植	中	中	高	skill
失败隔离	图内回退	外部适配层	脚本与人工复核	按场景

经验判断：事实契约固定后，路径选择应由任务形状决定，而不是由模型偏好决定。

表中结论为单项目工程观察；尚未进行跨团队、随机化或成本统一的对照实验。

20 · 研发反思

教训一：先确认数据是否完整，再讨论分析结果

走过的弯路

页面能打开、任务显示成功，并不代表数据已经收全。跨越零点、分页截断和查询区间不一致，都曾让报告少算信息。

问题不在“模型写得好不好”，而在写作之前的材料已经不完整。

时间	统一统计时区和截止时点
数量	检查分页是否真的取完
版本	生成前固定本次所用数据
异常	把漏收和回填写入运行记录

后来形成的原则：完整性检查属于报告生成的一部分，不能等到报告写完后再补。

提交依据：跨日采集、历史回填、分页截断、报告时间窗和固定数据快照等连续修复记录。

21 · 研发反思

教训二：自动化不能越过人工确认

我们一度希望系统自动清理过期状态、自动发布结果。实际运行表明，只要判断仍有争议，自动动作就可能把一次误判变成不可逆的业务后果。

自动清理

程序自行删除被认为“无效”的记录

自动发布

达到时间条件后直接形成正式结果

后来怎么改

- 删除改为可撤销的状态变更
- 正式发布必须由工作人员确认
- 保留原始材料、修改记录和操作人
- 智能体只提出建议，不取得最终决定权

自动化适合减少重复劳动，但不能替代责任主体作出需要承担后果的判断。

提交依据：自动清理功能回退、正式简报改为人工签发、可撤销处置和操作记录等修复。

22 · 研发反思

教训三：本地能运行，不等于上线后稳定

01 本地试验

命令能执行，样例能够返回

02 接入系统

需要限制工具、固定输入输出格式

03 长期运行

还要处理账户、路径、超时、并发和恢复

上线后实际暴露的问题

模型路由

必须固定服务商和模型，避免配置被悄悄替换

运行环境

服务账户、虚拟环境和启动路径需要逐项验证

资源管理

流式会话结束后要关闭连接，定时任务要能重试恢复

提交依据：工具白名单、模型固定、服务账户、虚拟环境、备份回滚、流式资源释放和定时恢复等修复。

23 · 方法提炼

可迁移经验：把“智能”放在最窄的必要位置

01	**先冻结事实，再生成叙述**	模型可以决定怎么写，不能决定数字是什么。
02	**工具权限写进边界**	只读、字段裁剪和人工确认应由代码保证。
03	**把失败当作一等路径**	回退、超时、幂等与恢复需要在规格阶段出现。
04	**用评估门控制下注**	POC 的目标是证明增量价值，也允许否决。

这套方法并不消除不确定性。它做的是把不确定性放到可观察、可复核、可撤回的位置。

提炼依据：项目中的事实快照、MCP 白名单、人工签发、降级链与 POC 决策门。

24 · 局限

研究局限与下一步：需要从“能运行”走向“可比较”

证据局限

单项目、短周期；提交数量和测试文件数只能说明工程活动，不能直接说明质量。

比较局限

三条路径尚未在统一数据、统一模型、统一时间预算下重复测量。

外部效度

高约束、低频简报任务的结论，未必适用于高吞吐实时决策系统。

建议的后续评估设计

质量

事实错误率、遗漏率、复核修改量

效率

端到端时延、工具调用数、人工分钟数

可靠性

失败率、回退率、重试一致性

治理

越权尝试、敏感字段暴露、审计完整度

下一步应保留原始运行日志与人工修改记录，建立可重复的配对评估。

谢谢，欢迎交流

洪煜

微信： hongyuatbj
邮箱： hongyuatcufe@icloud.com