AI-ASSISTED RESEARCH METHODOLOGY

AI驱动科研编程

从 Vibe Coding 到 规范驱动 的实战指南

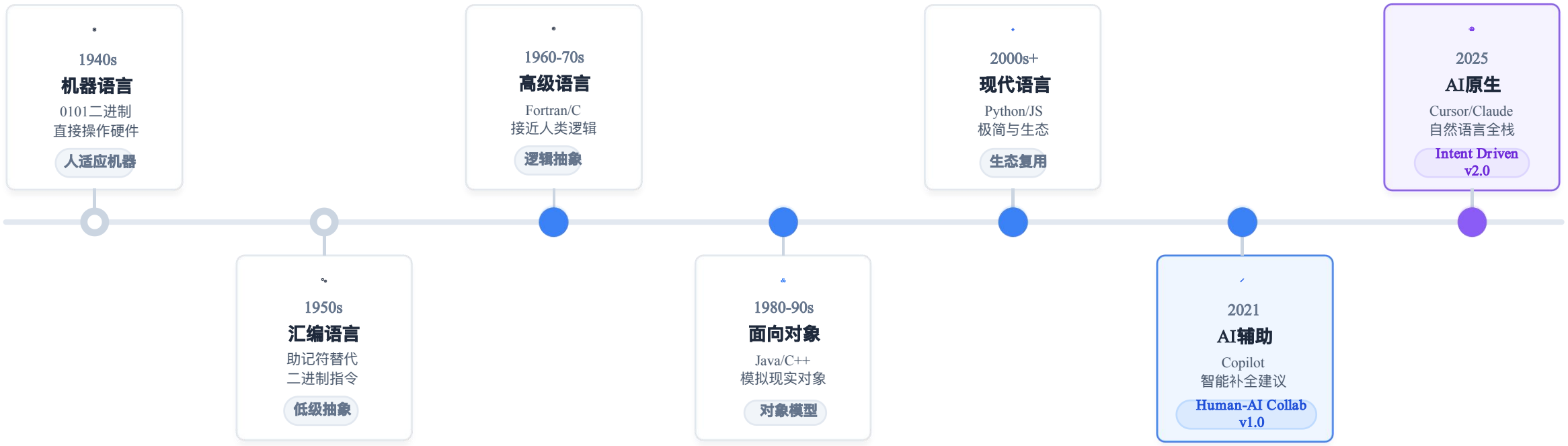
日期

2025-12-28

https://github.com/2025Emma/vibe-coding-cn

1.0 编程进化论

从“人迁就机器”到“机器服务人”的演进史



演进规律

编程语言的抽象层级不断提升，人机交互界面越来越接近人类自然语言。

NEXT STEP

Prompt Engineering Is The New Coding

1.1 编程范式革命

AI编程的本质：从“写代码”到“说需求”

核心观点转变

传统编程 = 人类 → 机器语言

AI编程 = 人类 → AI → 代码协作

从 "Syntax Recall"（语法记忆）
转向 "Prompt Engineering"（需求描述）

关键维度对比

维度	传统编程	AI编程	变革本质
核心技能	语法记忆、API查询	**需求描述、逻辑验证**	记忆 → 表达
学习曲线	陡峭 月-年级	平缓 天-周级	门槛降低
开发速度	慢（手写 + 手动调试）	**快（秒级生成 + 迭代）**	效率质变
适用场景	所有场景（含底层驱动）	80% 常规科研任务	二八定律

AI擅长领域

- 标准算法实现（排序、统计、GIS操作）
- 代码语言转换（Python ↔ R）
- 错误诊断与修复建议（Explain & Fix）
- 文档与注释自动生成

AI能力边界

- 复杂业务逻辑（需深度领域知识）
- 创新算法设计（非标准教科书内容）
- 系统级架构决策
- 极致性能优化（硬件级）

1.2 Vibe Coding

定义、双面性与应用场景

Vibe Coding

以自然语言对话为核心的探索式、迭代式编程方法。特点是"感觉驱动"—凭直觉描述需求，快速试错迭代。

三大核心特征

低门槛
无需记忆复杂语法，自然语言即代码

高速度
想法到代码分钟级实现，快速验证

强探索性
通过对话发现未知可能性与新方案

推荐使用

Code < 200 行

- ✓ 一次性数据处理脚本
- ✓ 快速原型验证
- ✓ 新技术栈学习

不适用

Code > 500 行

- × 需要长期维护
- × 团队协作项目
- × 论文核心算法

三大致命缺陷

规模灾难
500 行后代码沦为"补丁堆"，缺乏整体架构，维护成本指数级上升。

理解偏差
领域术语歧义（如"建筑密度"的三种定义），AI 缺乏上下文且不主动澄清。

文档失控
直接修改代码而未更新文档，导致"文不对题"，文档最终完全失效。

1.3 解决方案预告

保留速度优势，补齐可靠性短板

核心思想转变：从 对话驱动 走向 规范驱动

1

Vibe Coding

对话驱动

速度极快 (High Speed)

直觉探索 (Intuitive)

质量不可控 (Unstable)

2

Spec-First

规范优先

先定义后实现 (Def-First)

迭代有锚点 (Anchored)

质量有保障 (Reliable)

3

Spec-as-Source

规范即源码

自动生成代码 (Auto-Gen)

一致性100% (Consistent)

团队协作基石 (Collab)

Vibe Coding = 口头传达盖房

"边聊边改，想到哪说到哪 (Ad-hoc changes)"

结果：房子可能盖歪，细节遗漏

规范驱动 = 先出施工图纸

"图纸明确所有细节和标准 (Blueprints first)"

结果：质量有保障，可复现

"Vibe Coding 是编程的快餐 (Fast Food)—好吃但不健康。 规范驱动是健康快餐 (Healthy Fast Food)—既快又可靠。"

2.1 工具全景图（2025版）

主流AI编程工具四大类别与最新生态

AI原生IDE

完整开发环境，深度集成AI

Cursor | Windsurf | Antigravity | Trae | Kiro | Augment

- 适合中大型项目 (1000+行)
- Agent模式：整项目自主开发
- 深度上下文理解与引用

CLI 命令行工具

极客首选，脚本与自动化

Claude Code | Codex CLI | Gemini CLI | Droid

- 适合复杂算法与深度推理
- 超大上下文窗口 (200K - 1M)
- 自动化任务分解与执行

云平台开发

在线开发，零配置环境

Replit | bolt.new | Lovable | Orchids

- 无需环境配置，浏览器即用
- 实时全栈应用预览
- 适合教学、演示与快速原型

辅助工具

特定任务增强与可视化

Augment Code | Zread | NotebookLM | Mermaid Chart

- 上下文引擎增强
- 文档解读与架构可视化
- 特定场景效率提升

核心建议：组合优于单一选择

科研场景推荐组合：AI原生IDE (用于构建项目) + CLI工具 (用于复杂算法推理)

2025年

2.2 AI原生IDE对比（上）

Cursor / Windsurf / Antigravity 核心能力解析

行业标杆

Cursor

$20 Pro / $40 Business

Claude 4.5 / GPT-5.1、5.2

- **Agent模式：** 项目级自主开发，支持重构
- **Composer：** 多文件联动修改，效率倍增
- **@引用系统：** 精准控制上下文，减少幻觉
- **.cursorrules：** 定义项目规范，统一标准

科研价值

★★★★★

复杂算法 • 大规模数据 • 论文代码

性价比之选

Windsurf

$15/月 (1000 Credits)

Cascade Flow / Claude / GPT

- **Cascade Flow：** 领先的迭代式工作流
- **多模型支持：** Cascade集成Claude与GPT
- **多文件感知：** 出色的上下文理解能力
- **高性价比：** 比Cursor便宜25%

科研价值

★★★★☆

数据脚本 • 快速原型 • 实验迭代

Google免费

Antigravity

完全免费 (2025新政)

Claude Opus 4.5 + Gemini 3.0 Pro、Flash

- **3模型协同：** Claude、Gemini、GPT-OSS120B
- **多智能体异步：** 任务并行处理，提升效率
- **跨平台同步：** VSCode/终端/浏览器无缝切换
- **Google生态：** 深度整合，零成本使用

科研价值

★★★★★

学生首选 • 代码重构 • 算法探索

Cursor推出$40企业版增强隐私；Windsurf积分制更灵活；Antigravity目前完全免费但需Google账号。

2.2 AI原生IDE对比（下）

本土化方案 / 规范驱动 / 高端上下文引擎

Trae

国内免费

字节跳动本土化方案

核心模型	Doubao / Claude 3.7
限制	10快速+50慢速/月

核心功能亮点

- 完全本土化体验，无需代理
- 实时Web预览，可视化开发
- 智能Bug修复与中文文档生成

科研价值 ★★★★☆

最佳适用

中文文档 / 教学演示

Kiro

$20-$200/mo

规范驱动开发 (SDD)

核心模型	Model Agnostic
定价	Pro $20 / Power $200

核心功能亮点

- 强制"先文档后代码"流程
- 自动维护Memory Bank体系
- Spec-Driven保障可复现性

科研价值 ★★★★★

最佳适用

大型协作 / 核心复现

Augment

$100/mo

极致上下文引擎

核心模型	Context Engine + GPT-4
额度	208,000 Credits/m

核心功能亮点

- 超大代码库理解能力(RAG+)
- MCP工具原生集成支持
- SOC 2 Type II企业级安全

科研价值 ★★★★★

最佳适用

超大项目 / 复杂逻辑

2.2 CLI工具深度对比 (2025版)

Claude Code / Codex CLI / Gemini CLI / Droid

Claude Code

PRO/MAX

核心模型	Claude 4.5 Sonnet / Opus
价格	$20 Pro / $200 Max

- ✓ 项目记忆 (Project Memory)
- ✓ 200K 超长上下文
- ✓ 复杂任务多步分解
- ✓ 深度代码审查能力

适用场景：大型项目深度重构、论文算法精确复现

Codex CLI

PLUS

核心模型	GPT-5.2 / GPT-5.1
价格	$20 Plus / $200 Pro

- ✓ 原生多模态支持
- ✓ 实时联网查询
- ✓ 丰富的插件生态
- ✓ API 快速集成

适用场景：快速原型验证、API集成、多模态任务

Gemini CLI

PRO/FREE

核心模型	Gemini 3.0 Pro
价格	Pro、Free (60 req/min)

- ✓ 1M 超大上下文窗口
- ✓ Gemini 3.0 Flash 极速
- ✓ 海量文本无成本处理
- ✓ Google Search 集成

适用场景：零预算项目、大文本分析、学生/教学

Droid

FACTORY

厂商	Factory AI
价格	Subscription / API Key

- ✓ CI/CD 深度集成
- ✓ 代码库安全隔离
- ✓ 自动化任务流水线
- ✓ 企业级合规控制

适用场景：企业DevOps自动化、批量代码重构

2.4 云平台快速对比

Replit / bolt.new / Lovable / Orchids 特点解析

Replit

免费 / 付费

- 浏览器IDE，零环境配置门槛
- 多人实时协作编辑 (类似Google Docs)
- 内置托管，一键部署应用

SCENARIO 教学演示、快速原型验证、Python脚本分享

bolt.new

免费 / 付费

- 现代全栈开发环境，基于WebContainers
- 极速启动，浏览器内直接运行Node.js
- 实时预览与热重载，所见即所得

SCENARIO Web应用开发、前端Demo展示、全栈项目

Lovable

$20/月

- 全栈应用自动生成，关注产品而非代码
- 设计友好，生成的UI美观度高
- 集成数据库与后端逻辑，开箱即用

SCENARIO 快速产品验证 (MVP)、非技术创业、工具搭建

Orchids

免费试用

- UI Bench评分第一，界面生成能力极强
- 专注于复杂交互界面的构建
- 科研工具可视化界面可以尝试选择

SCENARIO 科研数据看板、实验控制面板、复杂UI设计

云平台优势：无需本地环境配置，极其适合跨设备工作与教学

提示：云平台通常依赖稳定的网络连接，处理大规模本地数据 (如GB级影像) 时可能存在上传下载瓶颈，建议搭配本地IDE使用。

2.3 工具选择决策树 (2025版)

Strategies for Tool Selection: Budget × Scenario × Skill

按预算 BUDGET

零预算
Antigravity + Trae
Google与字节提供的免费神器，覆盖全流程需求

低预算 <$20
Windsurf ($15)
Cascade Flow带来极高性价比，适合学生党

标准 $20-40
Cursor Pro ($20)
最成熟IDE + 免费强力CLI，行业标准配置

高端 $20-$100
Augment / Kiro
极致上下文与规范驱动开发，企业级首选

按场景 SCENARIO

数据脚本 <500 Lines
Cursor + Jupyter
交互式分析的最佳拍档，快速验证假设

中型项目
Windsurf / Trae
平衡上下文理解与成本，迭代速度快

大型系统 >2000 Lines
Augment / Kiro
超大上下文引擎或规范驱动，保障架构稳健

算法研究
Claude Code + Antigravity
深度推理与多模型协作，攻克复杂算法难题

按基础 SKILL LEVEL

零基础入门
Trae / Replit
中文友好 / 开箱即用

Python基础
Cursor / Windsurf
利用现有知识最大化效率，无缝迁移习惯

CLI爱好者
Claude Code / Gemini
纯键盘操作的极致快感，黑客式开发体验

Vibe Coding Meta-Method

道法术器：从思维到工具的完整方法论

道 Philosophy

核心思维与第一性原理

- 凡是你会做的，就不要人工做 (AI First)。
  凡是你不会做的，让AI代替搜索引擎
- 上下文是第一性要素 (Context is King)
- 先结构后代码，规划优于实现 (Structure First)
- 奥卡姆剃刀：如无必要，勿增代码

法 Methodology

系统化的实施原则

- 一句话目标 + 非目标 (Explicit Boundaries)
- 正交性设计，功能不重复 (Orthogonality)
- 能抄不写，文档即上下文 (Docs as Context)
- 按职责拆模块，接口先行

Meta-Methodology

术 Tactics

具体的实战技巧

- 明确写清：能改什么，不能改什么 (Constraints)
- Debug只给：预期 vs 实际 + 最小复现
- 测试可交给 AI 生成，断言由人审
- 代码一多就切会话 (Reset Session)

器 Tools

工具选择与组合

- IDE: Cursor / Windsurf / Kiro
- CLI: Claude Code / Gemini CLI
- Model: Claude 3.5 / GPT-4o / Gemini 2.5
- *详细对比见第二部分"工具全景图"*

3.1 Prompt 工程：精准沟通的艺术

5W1H 结构化框架与科研场景实战模板

对比实验：AI 为何"听不懂"？

模糊描述（Vibe Coding典型） Result: Confusion

User: "帮我分析这个数据"

AI Thinking: - Which dimension? (Stats? Regression?) - Which tool? (Pandas? Numpy?) - Output format? (Print? CSV? Plot?)

精准描述（Prompt Engineering） Result: Success

User: "Read data.csv, calculate Mean/Median/Std of 'pop' column. Use Matplotlib to plot a bar chart. Save as output.png with 300dpi."

AI Understanding: Input: data.csv | Task: Stats + Plot Stack: Matplotlib | Output: 300dpi PNG

5W1H 结构化框架

WHAT
具体要做什么任务？Core function definition

WHY
业务背景是什么？Context for intent

WHO
面向什么用户/读者？Style & Depth level

WHEN
批处理还是实时？Performance & Arch

WHERE
数据源与输出位置？I/O Path definition

HOW
技术约束与质量要求？Libs/Specs/Precision

科研场景通用模板

任务描述 (Task)：[Verb] + [Object] + [Purpose]

数据背景 (Context)：Source / Scale / Format

处理步骤 (Steps)：Step 1... Step 2... (Logic Chain)

输出要求 (Output)：Format / Precision / Metrics

技术约束 (Constraints)：Required Libs / Coding Style

测试验证 (Test)：Provide Mini Test Case

规范驱动开发（SDD）总览

核心原则、基本流程与三层实践路径

核心原则

Spec (规范) 是项目的"源头真理"，Code (代码) 只是规范的"实现产物"。

修改需求 → 先改Spec → 再让AI更新Code

Spec-First规范优先

Level 1

一次性任务的理想起点。先写清楚要求，再让 AI 生成，生成后规范可归档。

- ✓适用：<200 行小脚本
- ✓比Vibe Coding质量更高

Spec-Anchored规范锚定

Level 2

工程化标准。规范与代码同步更新，作为长期维护的锚点，需 Git 管理。

- ✓适用：500-2000 行项目
- ✓团队协作与长期维护必备

Spec-as-Source规范即源码

Level 3

极致自动化。只编辑规范，代码完全自动生成且不可手动修改，保证绝对一致。

- ✓适用：结构化/重复逻辑
- ✓100% 一致性保障

核心收益　少走弯路　可复现性　协作无障碍　长期可维护

Level 1: Spec-First (规范优先)

快速、高效、可追溯的入门级实践

Standard Workflow

1. Define Requirements & Goals
2. Write Spec Document (Markdown)
3. AI Generates Code Based on Spec
4. Manual Review & Simple Testing
5. Task Complete, Archive Spec

Applicable Scenarios

- Small Scripts (<200 lines)
- One-off Data Processing
- Quick Prototype Validation
- Learning New Tech Stacks

Pros

Higher quality than pure chat
Documented and traceable

Cons

Spec can become outdated
Not for complex iterations

spec_poi_density.md (Example Spec) — Markdown

```
# Feature: POI Kernel Density Analysis

## 1. Requirements Overview Calculate kernel density of urban POIs, generate heatmap for spatial distribution visualization.

## 2. Input Data - pois.geojson: GeoDataFrame, Point Geometry, EPSG:4326 - boundary.shp: Study Area Boundary Polygon

## 3. Output Deliverables - density.tif: GeoTIFF Density Raster (Float32) - heatmap.png: Visual Heatmap (300dpi, with legend)

## 4. Algorithms & Parameters - Method: Gaussian Kernel Density Estimation - Bandwidth: Use Scott's rule adaptive calculation - Resolution: 100 meters

## 5. Code Requirements - Function Signature: calculate_kde(gdf, bandwidth=None) -> raster - Must include Google-style docstrings - Unit test coverage > 80%
```

Key Tip

The clearer the Spec (especially input/output formats and core algorithm parameters), the higher the usability of AI-generated code, reducing Debug time.

Level 2: Spec-Anchored（规范锚定）

长期维护项目的工程化标准

锚定流程：Spec作为长期锚点

1. **需求变更** — 发现新需求/Bug
2. **更新Spec** — 修改文档
3. **AI更新代码** — 基于新Spec
4. **一致性审查** — 确保符合Spec

适用场景与核心实践

中型项目 (500-2000行)

代码量超出记忆范围，需要文档作为外部记忆。

团队协作项目

多人开发时，Spec是唯一的共识来源，避免"Vibe"不一致。

长期维护项目

3个月后重看代码，Spec比注释更易理解业务逻辑。

三大核心实践

- 双向同步更新
- Spec纳入Git管理
- 评审查Spec一致性

Memory Bank 体系架构

通过结构化文档体系，为AI建立"长期记忆"。

AI Context

```
project/
├── .ai-context/          # AI 上下文核心
│   ├── constitution.md       # 项目宪法
│   ├── architecture.md       # 架构设计
│   └── coding-standards.md       # 编码规范
│
├── specs/      # 功能规范
│   ├── feature-001/
│   │   ├── requirements.md
│   │   └── tests.md
│   └── feature-002/
└── src/      # 源代码 (基于Spec实现)
```

Level 3: Spec-as-Source（规范即源码）

极致自动化：只改规范，不碰代码

核心流程

将Spec视为源代码，生成的代码视为编译产物（Build Artifact），禁止人工修改生成的代码。

只编辑 Spec → AI 自动生成 → 代码标记为"只读"

适用场景

- 高度结构化代码：数据模型定义、API接口定义、CRUD操作
- 重复性逻辑：配置管理、样板代码（Boilerplate）
- 一致性要求极高：多端数据同步、协议解析
- 不适用：复杂的自定义业务逻辑、探索性算法

工具与评估

工具支持：Kiro（内置模式）、自定义脚本（监听文件变化）

100% 一致性　修改极简　需工具链支持

data_models.yaml (Spec)　Editable

User: fields: – id: UUID, primary_key – email: String, unique, required methods: – validate_email(): bool

models.py (Generated)　DO NOT EDIT

```python
# AUTO-GENERATED FROM data_models.yaml
# DO NOT EDIT THIS FILE MANUALLY

from dataclasses import dataclass
from uuid import UUID

@dataclass
class User :
id: UUID
email: str

def validate_email (self) -> bool:
"""验证邮箱格式"""
# AI生成的验证逻辑...
```

Memory Bank体系与术语表

消除歧义、保障一致性的项目知识库结构

项目目录结构

```
project/
  .ai-context/        # AI上下文核心
    constitution.md   # 项目宪法
    architecture.md   # 架构设计
    glossary.md       # 术语表
    coding-standards.md
  specs/              # 功能规范
    feature-001/
      requirements.md
      design.md
      tests.md
```

提示：将此目录添加到 .gitignore 但包含在AI上下文中

constitution.md

最高优先级

```
# 项目宪法：城市POI分析系统 ## 核心原则（不可违背） 1.  数据优先：基于真实数据，严禁捏造虚假坐标 2.  可复现性
：每个处理步骤必须提供复现脚本 3. 代码质量
：正确性 > 速度，宁可报错也不要静默失败 ## 技术栈（不可随意更改） - Python 3.9+ - 核心库：geopandas, rasterio,
matplotlib ## 禁止事项 -  禁止使用全局变量-
禁止 from module import *
```

glossary.md

消除歧义

```
# 术语表 ## 建筑密度（Building Density） **定义**：单位面积内的建筑物数量 **公式**：
density = count(buildings) / area(km²)
**单位**：个/km² **注意**：≠ 建筑覆盖率（后者是面积比，Area Ratio） ## 核密度估计（KDE） **定义**：基于核函数的空
间密度估计 **实现**：scipy.stats.gaussian_kde **参数**：
bandwidth（带宽）默认使用 Scott's rule 自适应
```

对比实验：同一任务的两种路径

Vibe Coding vs Spec-First 实战数据对比

任务目标 ◎计算城市路网 15分钟可达范围 (等时圈)　输入：路网 SHP + 起点坐标　输出：GeoDataFrame (Polygon)

方式 A：Vibe Coding

"边聊边改，直觉驱动"

1. 生成代码v1 (AI用了直线距离　)
2. Debug: 改用路网距离 (未用NetworkX　)
3. Debug: 引入NetworkX (忽略限速　)
4. Debug: 加入限速属性 (运行极慢　)
5. *经过 7 轮对话迭代，代码结构混乱...*

60 min 总耗时	**7** 次 DEBUG次数
60 分 文档完整性	 代码质量

⚠ 勉强可用，难以维护

方式 B：Spec-First

"先写规范，契约驱动"

1. **编写 Spec 文档 (5分钟)**
2. AI一次性生成完整代码 (2分钟)
3. 运行单元测试：通过　(1分钟)
4. AI补充 README 文档 (2分钟)
5. 最终验收，无重大逻辑错误

10 min 总耗时	**0** 次 DEBUG次数
90 分 文档完整性	 代码质量

★ 高质量，可复现

关键洞察：投入 5分钟 写Spec，能在开发中节省 6倍 时间。

EFFICIENCY BOOST: 600%

4.1 上下文管理

分层引用让AI“看全局”

核心挑战：上下文溢出

场景：1000行以上的项目，AI无法一次性读取所有代码。

错误做法： 把所有代码一股脑塞给AI。会导致窗口溢出，AI产生“幻觉”或“失忆”。

核心原则：最小充分上下文

> *"Minimal Sufficient Context"*

正确策略： 像做手术一样精准。只提供当前任务必须要用到的文档、接口定义和代码片段。

分层上下文策略 (Layered Context)

项目宪法 (Project Constitution)

始终包含：技术栈、编码规范、禁止事项（如 constitution.md）

模块架构 (Architecture)

按需加载：模块职责、接口定义、数据流向（如 architecture.md）

文件级代码 (File Level)

精确定位：当前任务直接相关的具体源码文件

函数/类定义 (Details)

细节操作：具体需要修改或参考的函数块、类方法

实战技巧：在 Cursor 等工具中，通过组合不同层级的引用（如 @constitution + @file），构建精准的上下文环境。

4.1 Cursor @引用系统：高效实操示例

让 AI 拥有精准的上下文感知能力

高效提示词构建：上下文 + 任务 + 约束

构建上下文 (Context Building) — The "Memory"

```
# Level 0: Project Constitution
@.ai-context/constitution.md
@specs/kde/requirements.md
@code:calculate_density
# Level 1: Feature Requirements
# Level 3: Target Function
```

+

执行指令 (Execution Prompt) — The "Action"

```
依据上述规范，请优化calculate_densi函数性能： Objectives:
1. Optimize processing time for 100k points to < 5s
scipy.spatial.cKDTree 3. Maintain 100%
2. Replace loops with
Type Hints & Google Style Docstring
# AI combines: Project Rules + Feature Specs + Code Logic
```

@ 常用引用指令速查

@file	引用完整文件e.g., @data_loader.py
@folder	引用文件夹结构（不含全内容）e.g., @src/analysis
@code	引用特定函数或类（精准上下文）e.g., @calculate_kde
@docs	引用已索引的文档库e.g., @Pandas (No switch)
@web	让 AI 联网搜索特定 URLe.g., @https://stack...
@git	引用 Git 提交记录或差异e.g., @diff (Changes)

最佳实践 (Best Practices)

- Avoid @Folder Abuse: Large folders consume Context Window.
- The "Combo": Always include @constitution.md.

4.2 结构化Debug工作流

告别"盲目修复"，让AI成为诊断专家

传统Debug：直接甩报错给AI → 瞎猜原因 → 越修越乱

结构化四步法

STEP 1
问题定位

编写Debug Report：收集症状

记录复现步骤 (1-2-3)

确认环境信息 & 已排除项

Time: 2min

STEP 2
根因分析

让AI列出 3个最可能原因

请求"诊断步骤"而非直接修复

执行诊断，锁定真凶

Time: 1min

STEP 3
修复设计

生成 多个修复方案 (A/B)

对比 Pros & Cons （影响范围）

选择副作用最小的方案

Time: 1min

STEP 4
实施验证

逐个文件修改（原子化）

运行 回归测试 防止倒退

更新文档（如果需要）

Time: 1min

专家级提示词

"不要直接问AI怎么办，而是让AI帮你分析原因 + 设计方案。"

✓ **预期效果(Expected)**
获得带逻辑推理的分析，而非盲目的代码补丁。

Prompt Template

@debug_report.md # 引用你的症状报告

1. Analyze（分析）：请根据报告，列出导致此错误的 3个最可能原因，并按可能性排序。 2. Diagnose（诊断）：对于每个可能原因，给出 验证步骤（如打印变量、检查特定行）。 3. Fix（修复）：确认原因后，请设计一个 最小改动 的修复方案，并解释为什么选择它。 # 注意：不要直接重写整个文件，先分析。

4.3 版本控制与实验管理

科研场景下的代码管理与实验追踪

Git工作流（科研适配版）

- **main** — 稳定版（论文投稿代码）
- **dev** — 日常开发分支
- **exp/*** — 实验分支（如 exp/attention-v1 ）

AI辅助 Commit Message

"我修改了注意力模块并修复了内存泄漏，请生成符合规范的提交信息。"

Conventional Commits

```
feat(model): add hierarchical attention module
fix(data): resolve data loader memory leak
- Implement HierarchicalAttention class
- Add garbage collection after epoch
```

实验追踪系统（Weights & Biases）

不再依赖Excel记录实验结果，使用专业工具自动追踪超参数与指标，实现实验的可复现与可视化对比。

AI集成指令

"@wandb_init.py 请在训练脚本中添加W&B记录代码，自动记录超参数（lr, batch_size）和训练曲线（loss, accuracy）。"

自动记录维度

HYPERPARAMETERS lr, batch_size, optimizer	**SYSTEM SPECS** GPU usage, RAM, Python ver
METRICS Train/Val Loss, Accuracy, F1	**ARTIFACTS** Best Model (.pt), Log files

```
import wandb
wandb.init(project="urban-flow-pred", config={
"learning_rate": 0.001,
"architecture": "ResNet-50"
})
# ... inside training loop ...
wandb.log({"loss": loss, "accuracy": acc})
```

4.4 关键技巧速查表

提升AI编程质量的6大实战锦囊

1 分步描述

将复杂任务拆解为逻辑链条，避免AI"一口气吃成胖子"。

BAD VS GOOD

"Data Analysis"

"Step 1: Read CSV; Step 2: Stats; Step 3: Plot"

2 提供示例 (Few-shot)

给出输入输出的具体样例，让AI通过模仿来理解格式。

PROMPT EXAMPLE

Input: {'name': 'BJ', 'pop': 2154}

Output: '北京人口：2154万人'

3 角色设定

指定专家身份，激活模型特定领域的知识权重。

PROMPT EXAMPLE

"You are a GIS expert, proficient in GeoPandas spatial indexing..."

4 明确约束

设定边界条件，防止AI自由发挥导致不可控。

PROMPT EXAMPLE

"Use matplotlib only, no seaborn; processing time < 10s"

5 迭代优化

不要追求一次完美，分轮次逐步完善代码。

WORKFLOW

R1: Basic Function → R2: Error Handling → R3: Perf → R4: Docs

6 AI自我审查

利用AI的反思能力，在运行前发现潜在Bug。

PROMPT EXAMPLE

"Please review the code above for potential memory leaks & edge cases."

Follow these 6 tips to reduce 80% of invalid dialogue & debugging time.

总结：认知 × 方法 × 工具

AI驱动科研编程核心回顾

三大核心认知

范式革命：从“写代码”向“说需求”的本质转变

模式选择：Vibe Coding 追求速度，规范驱动保障可靠性

工具观念：没有最强工具，只有最适合的组合

三大实战方法

Prompt工程：5W1H 结构化框架与角色约束

开发层级：Spec-First / Anchored / As-Source 三层境界

上下文：分层管理与精准引用 (@docs, @code)

三大工具策略

零预算：Antigravity + Trae (免费且强大)

标准版：Cursor Pro ($20/mo) + Gemini CLI

企业级：Kiro (规范化) + Cursor Team

> ***"AI是副驾驶，你才是飞行员。用好规范与工具，你将拥有10倍生产力。"***
>
> 永远记住：验证、测试、批判性思维不可缺少。