

活在 Terminal 的团队新成员

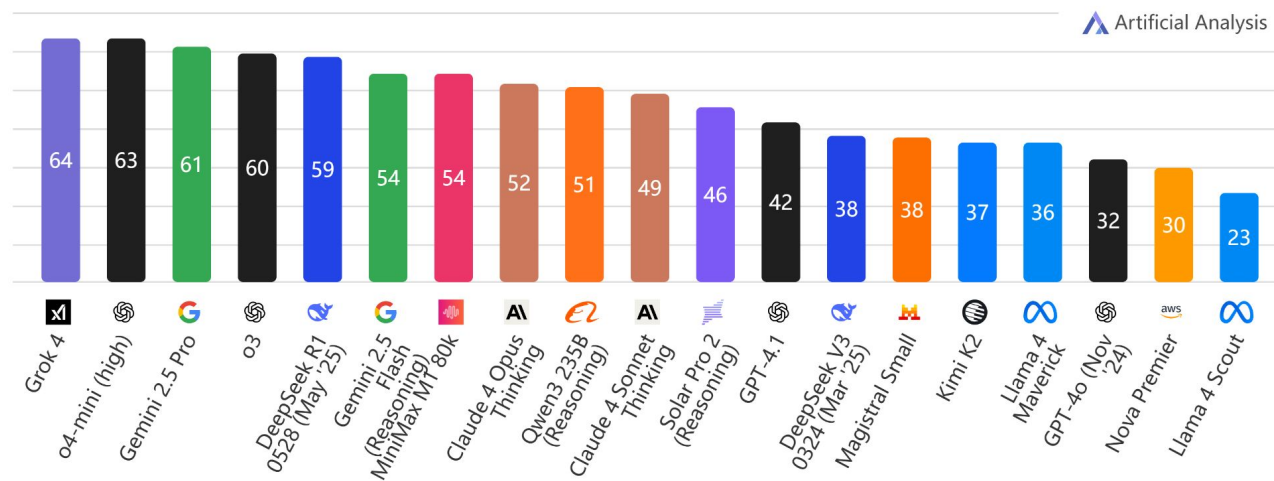
Gemini CLI Codeing 实践



With Gemini 2.5 Pro and its

Artificial Analysis Coding Index

Represents the average of coding benchmarks in the Artificial Analysis Intelligence Index (LiveCodeBench & SciCode)



60

Model requests
per minute

1K

Model requests
per day

Open
Source

Available free-of-charge

Gemini CLI 提供业界最大的使用限额，每分钟 60 个模型请求和每天 1,000 个模型请求免费

认领你的新伙伴

Step 0. install

`npx https://github.com/google-gemini/gemini-cli && gemini`

Step 1. login

- 1. Login with Google
- 2. Use Gemini API Key
- 3. Vertex AI

Step 2. Set Project id

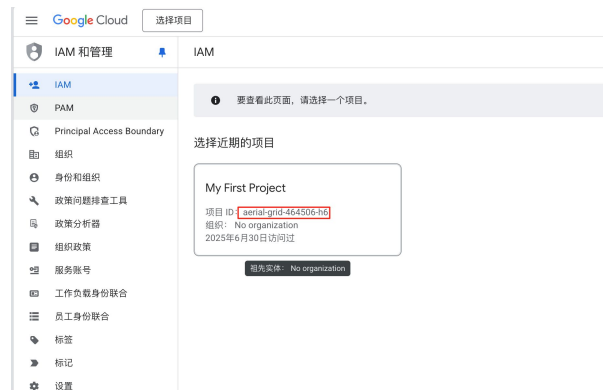
`https://console.cloud.google.com/projectselector/iam-admin/iam?hl=zh-cn`

`bash`

`echo 'export GOOGLE_CLOUD_PROJECT="ID"' >> ~/.bashrc && source ~/.bashrc`

`zsh`

`echo 'export GOOGLE_CLOUD_PROJECT="ID"' >> ~/.zshrc && source ~/.bashrc`





Hello GEMINI

单次 QA

通过管道符对话

```
echo "介绍一下Gemini" | gemini
```

使用命令对话

```
gemini -p 介绍一下Gemini
```

Agent

受限的 Agent

```
gemin
```

Yolo 模式

```
gemin -y # or gemin -yolo
```

入职培训 **Instructional Context**

➤ 培训内容

- 行为准则
- 编码规范
- Git 规范
- 开发工具
- 技术栈

➤ 培训方式 *Context File*

- `~/.gemini/GEMINI.md` (全局)
- `<project_path>/.gemini/GEMINI.md` (Project)
- `<project>/<Submodule>/.gemini/GEMINI.md` (Module)

Project GEMINI.md Example

Project: Ada: Agentic Detection AI

General Instructions

- ada is an artificial intelligence penetration testing agent designed to address real-world cybersecurity issues, including penetration testing, exploit development, exploit execution, exploit debugging, and exploit reporting.
- The idea behind this project is to explore the potential of artificial intelligence in the field of cybersecurity and apply it to practical cybersecurity problems.

Coding Style

1. Target python 3.13 or higher
2. Use python with type annotations. Use `list` instead of `List`.
3. Use `pathlib` instead of `os.path`. Use `Path.read\_text()` over `with ...open()` constructs.
4. Use `typer` to add user interfaces.
5. Keep code comments to a minimum and only highlight particularly logically challenging things
6. Do not append to the README unless specifically requested
7. Do not catch exceptions unless explicitly told to.
8. Write concise, short, minimal, pythonic code.
9. In most cases, avoid initializing variables just to pass them to a function. Instead just pass the expression to the function directly.

Here's an example for rule 9:

```
```python
bad
a = func()
Class(a)

good
Class(func())
```
```

Test Style

1. Use `pytest`, not `unittest`.
2. **<IMPORTANT>**Do not mock/patch anything that you're not explicitly asked to do**</IMPORTANT>**
3. Avoid writing trivial tests. Every test should test for at least one, preferably multiple points of failure
4. Avoid splitting up code in multiple lines like this: `a=func()\n assert a=b`. Instead, just do `assert func() == b`

Here's an example for rule 4:

```
```python
bad
result = func()
assert result == b

good
assert func() == b
```
```

Development Toolchain

- Python environment: uv and venv (uv venv)
- Test: pytest (pytest -v)
- Lint: ruff (uvx ruff check .)
- Format: black (uvx black .)

Project Structure

```
```text
ada/
├── agents/ # ai agents
├── environments/ # Environment classes
├── tools/ # ai agent tools
├── tests/ # Test files
├── README.md # project description
├── .gitignore # gitignore file
├── .env
└── .env.example # example of .env file
```
```


Commands 技能

```
→ ~/.gemini/commands/  
→ <project_path>/gemini/commands/
```

In: ~/.gemini/commands/git/ctnb.toml
Invoked via: /git:ctnb

description = "提交暂存更改至新建分支"
prompt = "分析Git暂存区修改, 生成符合Conventional Commits规范的提交信息, 并提交到新分支"

In:
~/.gemini/commands/git/commit.toml
Invoked via: /git:commit

description = "为暂存更改编写Commit"
prompt = "分析Git暂存区修改, 生成符合Conventional Commits规范的提交信息"

In: ~/.gemini/commands/git/fix.toml
Invoked via: /git:fix "Button is misaligned on mobile"

description = "Generates a fix for a given GitHub issue."
prompt = "Please analyze the staged git changes and provide a code fix for the issue described here: {{args}}."

```
Using: 1 GEMINI.md file  
  
> /git  
  
git:ctnb      提交暂存更改至新建分支  
git:commit    为暂存更改编写Commit
```

```
✦ 好的, 我已经分析了暂存区的修改。这些修改主要围绕对邮件MDM (Message Data Model) 解析器的重构, 使其更加模块化, 并增加了新的功能和依赖。  
  
这是一个建议的提交信息:  
  
1 feat(mdm): 重构邮件解析为模块化MDM结构  
2  
3 本次提交对邮件MDM解析器进行了重大重构, 旨在提高代码的模块化、可维护性和可扩展性。  
4  
5 主要变更包括:  
6 - **模块化解析**:  
   将邮件解析逻辑分解为针对附件、正文、头部、收件人等不同部分的独立模块。每个模块都有自己的数据模型 (Pydantic Model) 和解析函数。  
7 - **增强功能**:  
   - 增加了对附件内容、文件类型和哈希值的解析。  
   - 增强了对邮件头 (Headers) 的解析, 包括SPF、DMARC等认证信息。  
   - 引入了`tlxextract`, `lxml`, `beautifulsoup4`等库来支持更精准的域名、HTML和MIME内容解析。  
11 - **新增测试**:  
   增加了新的单元测试, 以确保重构后的解析器能够正确处理各种邮件样本, 并与预期的MDM输出匹配。  
12 - **依赖更新**:  
   在`pyproject.toml`和`uv.lock`中添加了新功能所需的新依赖项。  
13 - **辅助脚本**:  
   新增一个脚本, 用于从Sublime Security的API获取MDM数据, 方便本地测试和开发。  
  
你觉得这个提交信息怎么样? 如果同意, 我将使用这个信息进行提交。
```

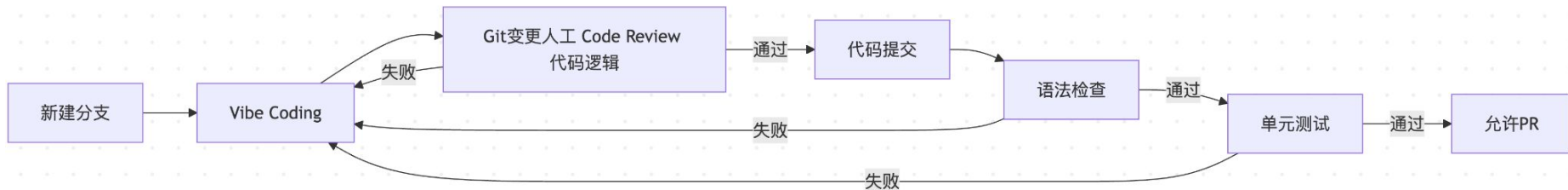
编码 **Codeing**

如何避免AI搞乱你的项目，使用软件工程的方式。任何会用 Git的人都不应该遇到这个问题。

- **CI Pipe line (DevOps先行)**
- **DDD 哲学 架构设计 (文档先行)**
- **TDD 哲学 (测试先行)**
 - **Vibe Codeing**
 - **人机协同**
- **项目分层 (高内聚 低耦合)**

CI Pipe line

- 编码任务交给AI
- 人类审核逻辑
- DevOps流程来保障健壮性



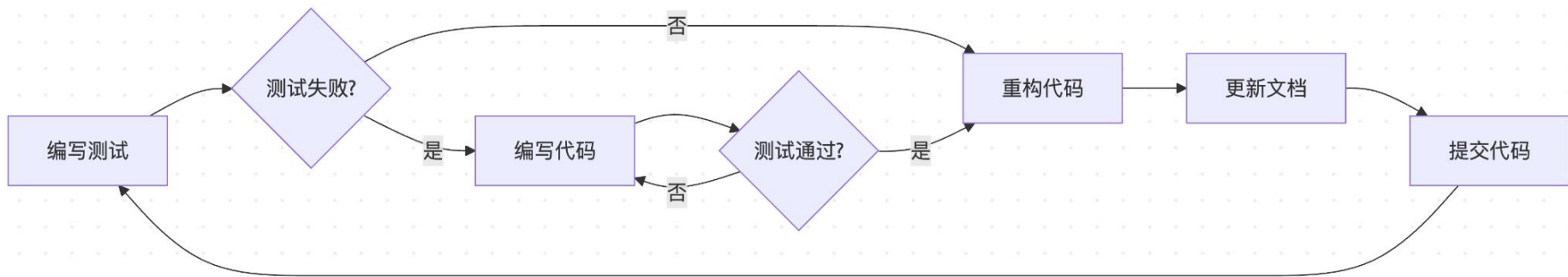
DDD 哲学 架构设计

- 及时更新为 ai 构建长期记忆
- 项目复杂度与规模对抗
- 拒绝上下文失忆



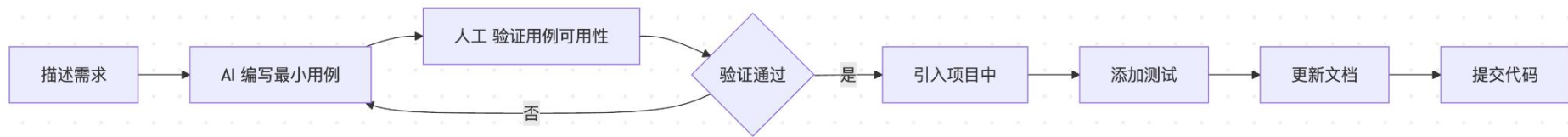
TDD 哲学 Vibe Codeing

- 通过单元测试保障
 - ai code 新实现的正确性 健壮性
 - 新的实现不会影响已有的代 码
- 每次任 务针对单 个功能 对抗Context太长导致的模型降智



TDD 哲学 人“机”协作

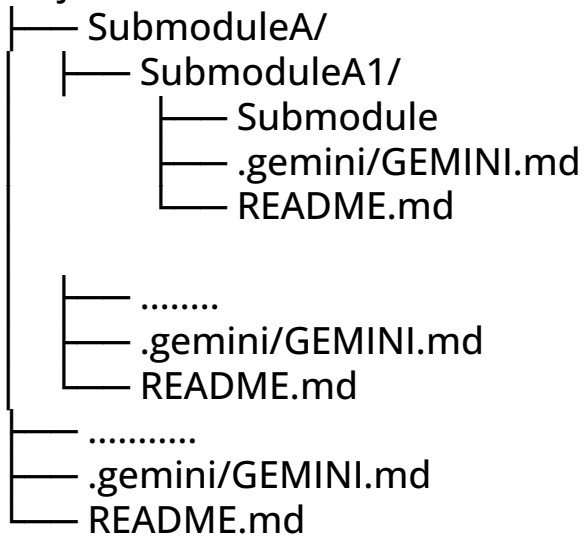
- 简单 > 复杂
- 增量 > 全量



项目分层 (高内聚 低耦合)

- 每个模块都是一个单独的项目
 - 在最小的项目中工作
 - 每个项目有自己上下文
 - 程序复杂性 与 Agent 上下文缺陷 对抗

Project/



工具扩展 *Tools Extensions*

```
<home>/ .gemini/extensions/mcpname/mcp.json
<workspace>/ .gemini/extensions/mcpname/mcp.json
```

🔗 **gemini-extension.json**

该文件包含扩展的配置。该文件具有以下结构： `gemini-extension.json`

```
{
  "name": "my-extension",
  "version": "1.0.0",
  "mcpServers": {
    "my-server": {
      "command": "node my-server.js"
    }
  },
  "contextFileName": "GEMINI.md",
  "excludeTools": ["run_shell_command"]
}
```

- `name`：扩展名的名称。这用于唯一标识扩展，并在扩展命令与用户或项目命令具有相同的名称时解决冲突。
- `version`：扩展的版本。
- `mcpServers`：要配置的 MCP 服务器的映射。键是服务器的名称，值是服务器配置。这些服务器将在启动时加载，就像在 [settings.json](#) 文件中配置的 MCP 服务器一样。如果扩展名和文件都配置了具有相同名称的 MCP 服务器，则文件中定义的服务器优先。
`settings.json` `settings.json`
- `contextFileName`：包含扩展上下文的文件的名称。这将用于从工作区加载上下文。如果未使用此属性，但扩展目录中存在文件，则将加载该文件。 `GEMINI.md`
- `excludeTools`：要从模型中排除的工具名称数组。您还可以为支持命令的工具（如工具）指定特定于命令的限制。例如，将阻止命令。
`run_shell_command` `"excludeTools": ["run_shell_command(rm -rf)"]` `rm -rf`

Gemini CLI 启动时，它会加载所有扩展并合并其配置。如果存在任何冲突，则工作区配置优先。

推荐：扩展终端命令行工具并在 *GEMINI.md* 文件添加工具使用描述

CLI Tools > MCP

小技巧 *Tips*

- **Shell模式**: 输入!进入 ESC退出
- **对话使用文件**: @filename
- **摘要整个对话**以在下次对话使用: /compress
- **对话**
 - 保存对话: /chat save <tag>
 - 加载对话: /chat resume <tag>
 - 列出对话: /chat list
- **上下文**
 - 添加记忆: /memory add <text to remember> 默认到全局
 - /memory show 查看所有的上下文
 - /memory refresh 刷新上下文
- **ESC**: 打断Agent 操作
- **↑ 键**: 回到上一个指令

高阶玩法

- **开多个Terminal** 在不同分支工作 实现不同的 feature 最终提交PR
- **Gemini 模式**是多模态的可以@图像文件 辅助

```
➤ awesome-offensive-security-ai git:(main) gemini

X GEMINI

Tips for getting started:
1. Ask questions, edit files, or run commands.
2. Be specific for the best results.
3. Create GEMINI.md files to customize your interactions with Gemini.
4. /help for more information.
$ pwd

✓ Shell Command pwd

/Users/waffle/project/awesome-offensive-security-ai

shell mode enabled (esc to disable)

! | Type your message or @path/to/file

~/project/awesome-offensive-security-ai (main*) no sandbox (see /docs) gemini-2.5-pro (100% context left)
```

```
➤ awesome-offensive-security-ai git:(main) gemini

X GEMINI

Tips for getting started:
1. Ask questions, edit files, or run commands.
2. Be specific for the best results.
3. Create GEMINI.md files to customize your interactions with Gemini.
4. /help for more information.

> @

CONTRIBUTING.md
README.md
README_ZH.md
~/project/awesome-offensive-security-ai (main*) no sandbox (see /docs) gemini-2.5-pro (100% context left)
```

不只是 *Code Coding*

不要让思维限制了新伙伴的能力, 任何工作都 值得一试

- 对比 <https://arxiv.org/html/2502.10931v2> 与 <https://arxiv.org/html/2502.10931v1> 之间的差异



- 今日Github趋势关于AI,安全,Agent的有哪些有意思的项目



相关

Cherny, B. (2025, July 4). *Claude Code & the evolution of agentic coding* [Video]. YouTube.

<https://www.youtube.com/watch?v=gv0WHhKeISE>

Google. (2025). *Gemini CLI: An open-source AI agent that brings the power of Gemini directly into your terminal* [Computer software].

GitHub. <https://github.com/google-gemini/gemini-cli>

Cursor. (n.d.). *Agent review documentation*. <https://docs.cursor.com/en/agent/review>

Anthropic. (2025, April 17). *Claude Code best practices*. <https://www.anthropic.com/engineering/claude-code-best-practices>

Yage AI. (n.d.). *Agentic memory*. <https://yage.ai/agentic-memory.html>

QA

感谢聆听