



Python 算法库构建实践

复杂必须存在于某处

Mathew Shen

2025-02-27



Outline

1. What

2. Why

3. How



1.1 算法库是什么

- 一个库(library)/包(package)/SDK
- 一个有着易用接口, 详细文档, 完善测试的库(library)/包(package)/SDK



Outline

1. What


2. Why

3. How



2.1 算法是复杂的

```
1  def fleiss_kappa(table: np.ndarray) -> float:
2      # skip many lines here...
3      assert n_total != 0, "Empty table"
4      assert n_total <= n_sub * n_rat, "Check total rates count"
5      p_cat = table.sum(axis=0) / n_total
6      table2 = table * table
7      p_rat = (table2.sum(1) - n_rates) / (n_rates * (n_rates - 1.0))
8      p_mean = p_rat.mean()
9      p_mean_exp = (p_cat * p_cat).sum()
10     kappa = (p_mean - p_mean_exp) / (1 - p_mean_exp)
11     return kappa
```

 Python



2.2 复杂必须存在于某处

With nowhere to go, it has to roam everywhere in your system, both in your code and in people's heads. And as people shift around and leave, our understanding of it erodes.¹

¹<https://ferd.ca/complexity-has-to-live-somewhere.html>



2.2 复杂必须存在于某处

With nowhere to go, it has to roam everywhere in your system, both in your code and in people's heads. And as people shift around and leave, our understanding of it erodes.¹

Complexity Has to Live Somewhere

Complexity has to live somewhere; but it **does not have to live everywhere**.

Embrace it, give it the place it deserves, design your system and organisation knowing it exists, and focus on adapting.

对于算法工程来说，算法复杂性要安放的“Somewhere”是哪里呢？

¹<https://ferd.ca/complexity-has-to-live-somewhere.html>



2.3 算法的“无状态”特征

“无状态”是指算法的输入和输出之间的关系是确定的，不依赖于外部状态

- 天然地和业务流程解耦
- 可以自成体系，独立于业务流程之外



2.4 核心算法的代码量并不大

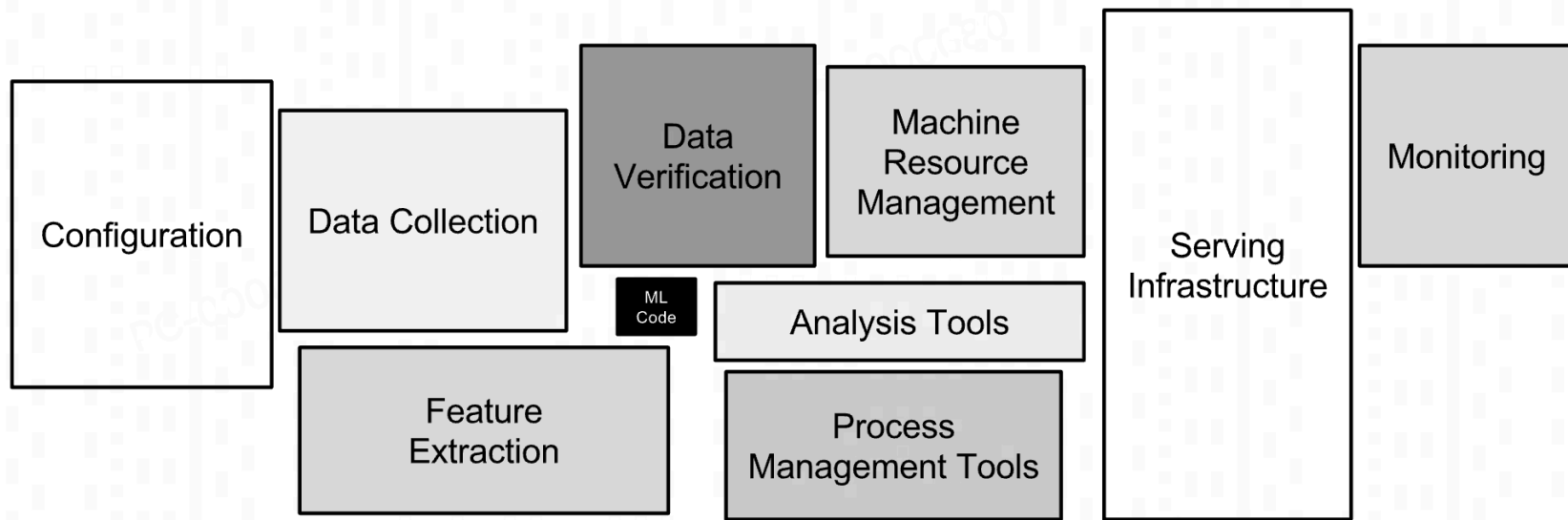


Figure 1: Hidden Technical Debt in Machine Learning Systems¹

¹<https://papers.neurips.cc/paper/2015/file/86df7dcfd896fcdf2674f757a2463eba-Paper.pdf>



2.5 算法代码隔离的好处

1. 更好的可读性：代码的逻辑更清晰
2. 更好的可维护性：更容易进行测试，更容易进行代码重构
3. 更好地复用：更方便地应用到不同的场景中(SDK, HTTP, RPC, ...)



Outline

1. What

2. Why

3. How



3.1 算法库的构建

- 算法库的形态是一个库(Library)/包(Pacakge)/SDK
- 算法实现代码，包括算法的核心代码和相关的工具函数
 - 把代码尽可能写好: 读代码的时间远大于写代码的时间
- 算法文档：用户文档(README/文档网站等)，内部文档(注释 & 说明文档等)
 - README 必不可少
 - 此外最好还有: CHANGELOG, CONTRIBUTING, LICENSE, ...
- 测试代码：单元测试，集成测试，属性测试，性能测试等
 - 注意：测试代码也是需要维护的代码，是负债而不是资产



3.2 构建工具

模板库参考 MPPT¹

- Package and project manager: UV
- Linter and code formatter: Ruff, Isort, Black
- Type checker: MyPy
- Documentation: MkDocs(Material for MkDocs)
- Test: Pytest, Hypothesis

¹<https://github.com/shenxiangzhuang/mppt>



3.3 常用工作流程

- 1.开发流程: 需要新的算法/功能或发现 Bug -> 仓库提 **Issue** 说明需求背景 -> 新建 **PR/MR(关联 Issue)** -> 编写代码 -> 编写测试代码 -> 编写文档(CHANGELOG 等) -> Code Review -> 合入主干
- 2.发版流程: 新建 Release 分支 -> 更新版本号 -> 更新 CHANGELOG -> 合入主干 -> 代码仓库打 **Tag**, 发布 **Release** -> Build SDK -> 发布 SDK
- 3.Debug 流程: 发现 Bug -> 仓库提 **Issue** 说明 **Bug** 现象, 提供复现代码 -> 复现问题 -> 定位问题代码 -> **Blame** 问题代码(找出引入问题的 MR/PR) -> 新建 **PR/MR(关联两个 Issue)** ->(同开发流程)



3.4 当我们在谈论维护时

- Status of Python versions¹
 - 必须支持的 Python 版本: 所有非 EOF(End of Life)版本
 - 截止 2025-02 月, 需要支持的版本: 3.9, 3.10, 3.11, 3.12, 3.13, 3.14
 - feature 阶段的版本为可选支持版本: 如当前的 3.14
 - 当 Python 版本进入 EOF 阶段时, 在新版本移除对应的支持
 - 如目前的 3.8 及之前的版本
- Status of dependencies
 - 按需或定期升级依赖库的版本
- 不断进行优化和改进
 - 使代码更可读, 使文档更完善, 使测试更全面

¹<https://devguide.python.org/versions/>



3.5 算法库的使用

- `uv add xxx`
- `poetry add xxx`
- `pip install xxx`

QA