一、CMMI 的层次成熟度模型

1. Level 1 —— 初始

过程无序，全靠个人英雄主义，成功难以复现，风险极高。

2. Level 2 —— 已管理

为了“能交付”，我需要最基本的管理：给需求建基线、排计划、跟踪进度、做质量保证和配置管理——每一环都有可见的产出。

3. Level 3 —— 已定义

不再只关注单个项目，而是把可复用的模板、规范和最佳实践沉淀为“组织过程资产库”。日后我或他人启动新项目时，可裁剪同一套标准化流程。

4. Level 4 —— 量化管理

关键过程受统计控制。我为质量和性能设立可量化目标，收数据、画控制图、算过程能力指数，用客观数字驱动决策。

5. Level 5 —— 优化进入持续改进模式：遇到波动先做根因分析（RCA），再用技术或过程创新（自动化、AI、DevOps、Chaos Engineering 等）闭环优化，让系统自我进化。

二、项目背景与现实过程

项目：用 TimeMixer 改进 FlightBERT++ 的飞行轨迹预测

开发者：仅有我一个人，兼顾需求、算法、后端、测试与论文写作

周期：2024 年 12 月 → 2025 年 5 月，分三个迭代（Baseline → 改进 → 部署）

技术栈：PyTorch 2.2、HF Transformers、Weights & Biases、FastAPI、Docker、GitHub Actions

我的主要实践

CMMI 过程域	我实际怎么做	工具	收获/痛点
需求管理	读文献 + 头脑风暴列出“长序列、异构数据”两大痛点，写进 Notion 需求清单并标	Notion	找准了问题核心，但是想法容易膨胀

	MUST/SHOULD		
项目计划	6 个月拆 3 迭代；用番茄钟记录每日 6 个番茄，周末看燃尽图	Obsidian + Dataview	目标清晰；期中周计划漂移
监控与控制	W&B 仪表盘实时看 ADE / FDE，周复盘改进幅度	Weights & Biases	数据驱动决策
体系结构设计	画架构图、写 ADR 说明 trade-off	Mermaid + Markdown	决策可追溯
实现 & 自评审	Feature 分支 + pre-commit；提交前自查 Diff	Git + Pre-commit	缺陷密度 < 1/KLOC
测试	pytest 覆盖率 88 %；脚本自动跑 14 条航线 A/B	PyTest + Bash	回归稳定，外部气象 API 偶发 504
度量与分析	记录 ADE、FDE、延迟、GPU 占用；迭代末对比	Excel + W&B	易写论文图表
风险管理	列出三大风险：数据缺口 / 算力 / 截稿；设阈值与备选方案	Markdown Risk Log	未因算力卡死
配置管理	Git 管理代码，DVC 管理数据，MLflow 存储权重，Docker 镜像自动构建	GitHub Actions	一条脚本即可复现
过程与产品质量保证	每月跑一次手工 Checklist，核对文档与实验可复现性	Markdown	审稿人复现无障碍
量化管理尝试	用控制图监控 ADE / FDE 趋势	Python + Matplotlib	初探 Level 4 思维

三、我的成熟度对标评估

CMMI 级别	**我已经达到的关键实践**	**待加强**
L1	依靠个人能力成功交付	
L2	需求基线、计划跟踪、版本与配置管理、基础 QA	缺少了独立的 QA 审查
L3	ADR、统一模版、固定个人流程	模版还没沉淀为“组织资产”
L4	ADE/FDE 控制图	缺少 SPC、缺少统计门限、预测模型存在一定缺

		陷
L5		根本原因分析自动化、持续改进闭环

综合判断，我的整体成熟度在 Level 2 – 3 之间

四、差距与风险

1. 缺少独立 QA：可能导致个人的盲区导致缺陷漏检
2. 量化基线单薄：只监控了 ADE/FDE，缺少了链路延迟、数据漂移和缺陷密度等指标
3. 自动化发布不完整：镜像仍然由我手工推送，没有建立回滚脚本
4. 手工完成根本原因分析：风险日志完备，但是没有自动报警

五、改进计划

在我的预期中，需要 12 个月进行改进：

0-3 月：落实 Level 3 的要求

1. 提炼模版：把需求规格说明书、测试用例以及发布检查表整理到 Assets 仓库
2. 引入轻量 QA：每次迭代结束后请导师做走查，并使用工具进行静态扫描
3. 自动化发布 V1: GitHub Actions 合并以得到打包后的镜像，然后 pull 到 Docker Hub，接下来自动部署到本地 K8s

4-8 月：尝试 Level 4

1. 度量体系全景化：
   过程：工时偏差、缺陷修复周期
   产品：ADE/FDE、延迟 P99、代码覆盖率
   为关键指标设置 UCL/LCL，在 Grafana+Prometheus 上设置报警
2. 统计过程控制（SPC）：自动产出控制图与 Cp/Cpk，每周复盘
3. 缺陷预测模型：用 commit 复杂度等特征训练随机森林，标出高风险文件

9-12 月：向 Level 5 过渡：

1. 自动 RCA：把 5 Whys、鱼骨图封装成脚本，缺陷触发即生成报告
2. 持续改进闭环：异常 → 自动开 Issue → Pull Request 更新流程资产。
3. 技术创新：试水 AutoML 调参、Chaos Engineering 注入 GPU 限速验证韧性。

六、总结

沿着上述路径，我将把“能交付”升级为“能量化、可预测、可优化”。当 SPC、缺陷预测与自动 RCA 三大支柱落地后，我基本具备 Level 4 的量化控制力；再辅以持续改进闭环，就能逼近 Level 5 的优化型结果。

CMMI 不只是一些空洞的标准，更是一套帮助我不断逼近“系统化、数据化、可持续交付”目标的方法论。期待在 12 个月后，我不仅把 TimeMixer+FlightBERT++的模型推向更高舞台，也把自己的过程成熟度推进到新台阶——即使只有一个人，也能像一支成熟团队那样，持续、高质量地交付科研与工程成果。