



AI Native 实践大作业说明

一、总体设计

1.1 核心目标

通过一个完整的AI应用开发项目,让团队成员掌握:

- 大模型API调用与参数调优
- Prompt工程设计与迭代优化
- RAG或Agent开发实践
- 量化评测与效果调优
- AI Coding工具的深度使用

1.2 设计原则

原则	说明
角色公平	前端、后端、测试都需完成前后端开发,打破角色边界
技能全覆盖	必须涉及大模型、Prompt、RAG/Agent、评测四大模块
实操优先	做出能跑、能用、能部署的真实应用

AI Coding	鼓励使用AI工具辅助开发,但需记录过程
可评测	标准化提交,便于快速review和防抄袭

1.3 作业形式

- **个人或小组:** 允许1-3人组队
- **时间周期:** 6周(10月13日 → 11月21日)
- **场景选择:** 完全自定义,结合真实业务需求
- **难度定位:** 类似毕业设计,聚焦AI能力而非工程复杂度

二、场景要求

2.1 自定义场景规则

完全开放选题,但必须满足以下条件:

核心技术要求

- 调用至少1个大模型API(Claude/GPT/国产模型均可)
- 必须有Prompt设计与优化(至少3轮迭代)
- 必须实现RAG或Agent之一(或两者都有)
- 必须有量化评测体系(至少30条测试用例)

功能要求

- 有可交互的Web前端界面(不能只是命令行)
- 有后端API服务(至少3个接口)
- 能够部署到服务器并提供访问地址
- 解决一个真实的业务或学习场景问题

三、参考案例:项目文档智能问答系统

这是一个技术参考案例,仅供没有想法的同学参考,鼓励完全自定义其他场景

3.1 案例背景与目标

场景描述:

开发一个智能问答系统,帮助团队成员快速查询内部技术文档(API文档、架构设计、操作手册等)。传统的全文搜索经常找不到答案或结果过多,需要用RAG技术实现智能语义检索和精准回答。

核心功能:

1. 文档管理:上传PDF/Markdown/Word文档,自动解析和向量化
2. 智能问答:基于RAG检索相关内容,用大模型生成答案并标注来源
3. 持续优化:通过Prompt迭代和评测体系不断提升准确率
4. Web界面:提供友好的用户界面,支持文件上传和对话交互

3.2 技术实现思路

1. 文档处理流程

代码块

- ```
1 文档上传 → 格式解析(PyPDF2/python-docx)
2 → 文本分片(按段落/固定长度,overlap=50字)
3 → 向量化(OpenAI embedding/本地模型)
4 → 存储(Chroma/FAISS向量数据库)
```

### 关键点:

- 分片策略影响检索质量:段落级(适合结构化文档)vs固定长度(适合长文本)
- 需要保存原始文档信息,用于答案溯源

### 2. RAG检索实现

#### 代码块

```
1 # 检索示意代码
2 def rag_search(question, top_k=5):
3 # 1. 问题向量化
4 q_vector = embedding_model.encode(question)
5
6 # 2. 向量检索
7 results = vector_db.search(q_vector, top_k=top_k)
8
9 # 3. 构建上下文
10 context = "\n\n".join([
11 f"[文档{i}:{r.doc_name}-段{r.chunk_id}]\n{r.text}"
12 for i, r in enumerate(results)
13])
14
15 return context
```

```
13])
14
15 return context, results
```

## 关键点:

- top-k参数影响召回率和精准度(建议3-7)
- 可以尝试混合检索:向量检索(语义相似)+BM25(关键词匹配)

## 3. Prompt工程示例

### V1版本(基础)

#### 代码块

```
1 你是一个技术文档助手,根据以下文档片段回答问题。
2
3 文档内容:
4 {context}
5
6 用户问题: {query}
7
8 请回答:
```

问题:\*经常编造内容,不标注来源,准确率只有65%

### V2版本(添加约束)

#### 代码块

```
1 你是技术文档问答助手,请严格基于给定文档回答问题。
2
3 规则:
4 1. 仅使用下方文档内容,不要编造
5 2. 如果文档中没有相关信息,回答"文档中未找到相关内容"
6 3. 回答简洁,不超过200字
7
8 文档内容:
9 {context}
10
11 用户问题: {query}
```

改进:幻觉减少,准确率提升到72%,但仍不标来源

### V3版本(格式化输出)

## 代码块

```
1 你是技术文档问答助手,请严格基于给定文档回答问题,并标注来源。
2
3 回答格式:
4 答案: [你的回答,不超过150字]
5 来源: [文档X-段Y], [文档Z-段W]
6
7 规则:
8 - 严格使用文档内容,禁止编造
9 - 必须标注每条信息的来源片段
10 - 如无相关内容,回答"未找到"
11
12 ---
13 文档内容:
14 {context}
15
16 用户问题: {query}
```

**效果:** 准确率78%,引用标注率95% ✓

## 迭代总结:

- V1→V2: 加约束减少幻觉(+7%)
- V2→V3: 强制格式化输出提升溯源性(+6%)
- 关键:每次改动都要用测试集量化效果

## 4. 评测体系设计

### 测试数据集构建(30条样本)

## 代码块

```
1 {
2 "test_cases": [
3 {
4 "id": "Q001",
5 "question": "系统支持哪些文件格式?",
6 "expected_keywords": ["PDF", "Markdown", "Word"],
7 "difficulty": "easy"
8 },
9 {
10 "id": "Q015",
11 "question": "如何优化检索性能?",
12 "expected_keywords": ["缓存", "索引", "分片"],
13 "difficulty": "medium"
14 },
15]
16 }
```

```
15 {
16 "id": "Q030",
17 "question": "向量数据库的索引算法对比?",
18 "expected_keywords": ["HNSW", "IVF", "准确率", "速度"],
19 "difficulty": "hard"
20 }
21 // ... 总共至少30条
22]
23 }
```

## 自动化评测脚本(evaluate.py)

### 代码块

```
1 def evaluate_system():
2 results = []
3 for case in test_cases:
4 # 1. 调用问答系统
5 answer = qa_system.ask(case["question"])
6
7 # 2. 检查关键词覆盖
8 accuracy = check_keywords(answer, case["expected_keywords"])
9
10 # 3. 统计延迟和成本
11 latency = answer.metadata["latency_ms"]
12 tokens = answer.metadata["total_tokens"]
13
14 results.append({
15 "id": case["id"],
16 "accuracy": accuracy,
17 "latency": latency,
18 "tokens": tokens
19 })
20
21 # 输出量化指标
22 return {
23 "accuracy": avg([r["accuracy"] for r in results]),
24 "avg_latency_ms": avg([r["latency"] for r in results]),
25 "avg_tokens": avg([r["tokens"] for r in results]),
26 "cost_per_query": calculate_cost(avg_tokens)
27 }
```

## 输出示例(metrics.json)

代码块

```
1 {
2 "accuracy": 0.78,
3 "avg_latency_ms": 1250,
4 "avg_tokens": 450,
5 "cost_per_query": 0.02,
6 "citation_accuracy": 0.95
7 }
```

## 3.3 案例总结

这个案例涵盖了所有技术要求:

- 大模型应用:Claude/GPT API调用,Prompt三轮迭代
- RAG实现:向量检索+引用标注
- 评测体系:30条测试用例,自动化脚本
- 前后端:Web界面+RESTful API
- 可部署:提供访问地址

## ✓ 四、硬性技术要求(100分基础分)

### 4.1 基础架构(20分)

| 要求项   | 具体标准                                                 | 评分 |
|-------|------------------------------------------------------|----|
| 前端界面  | 必须有可交互的Web界面,至少包含:输入区、输出区、状态展示                       | 8分 |
| 后端API | RESTful API,至少包含3个接口(如 /chat , /evaluate , /health ) | 6分 |
| 部署要求  |                                                      | 4分 |

|      |                          |    |
|------|--------------------------|----|
|      | 必须部署到虚拟机/服务器,提供可访问的URL   |    |
| 一键启动 | 提供启动脚本,评审人配置 .env 后可一键启动 | 2分 |

### 注意事项:

- ⚠️ **后端禁止使用Java**(鼓励Python/Node.js/Go等,增加挑战)

- ⚠️ 不能只做命令行工具,必须有Web界面
- ⚠️ 部署地址必须在提交时可访问

## 4.2 大模型应用(25分)

| 要求项      | 具体标准                                    | 评分  |
|----------|-----------------------------------------|-----|
| 模型调用     | 至少对接1个大模型 API(Claude/GPT/国产模型均可)        | 5分  |
| Prompt工程 | 展示至少 <b>3轮Prompt迭代</b> ,记录每版提示词与效果对比    | 10分 |
| 参数调优     | 对比不同 temperature/top_p/max_tokens对结果的影响 | 5分  |
| 异常处理     | 超时重试、错误降级(如切换模型或默认回复)                   | 5分  |

### 关键要求:

- Prompt迭代必须有量化数据支撑**,不能只说"感觉更好"
- 每次迭代要说明:改了什么、为什么改、效果如何
- 参数对比至少测试2组不同配置

## 4.3 RAG或Agent实现(30分,二选一或都做)

### 选项A: RAG实现

| 要求项  | 具体标准                               | 评分  |
|------|------------------------------------|-----|
| 文档处理 | 支持多格式文档,实现文档解析与分片                  | 10分 |
| 向量检索 | 实现向量检索<br>(embedding+向量数据库),返回相关片段 | 10分 |
| 引用标注 | 答案必须标注来源,可追溯到原始文档片段                | 5分  |
| 知识管理 | 支持动态添加/删除知识库内容                     | 5分  |

## 选项B: Agent实现

| 要求项  | 具体标准                                        | 评分  |
|------|---------------------------------------------|-----|
| 工具定义 | 至少实现2个可调用工具(如搜索、计算、文件操作)                    | 5分  |
| 决策链路 | 显示Agent思考过程<br>(observation→thought→action) | 10分 |
| 失败重试 | 工具调用失败时的自动恢复机制                              | 5分  |
| 多步协作 | 至少有1个任务需要调用2次以上工具完成                         | 10分 |

## 4.4 评测与优化(20分)

| 要求项   | 具体标准                      | 评分 |
|-------|---------------------------|----|
| 测试数据集 | 准备至少30条测试用例(问题+标准答案/预期行为) | 5分 |

|       |                             |    |
|-------|-----------------------------|----|
| 自动化评测 | 提供可独立运行的评测脚本,输出量化指标         | 5分 |
| 量化指标  | 至少包含3类:准确性、性能(延迟)、成本(token) | 5分 |
| 迭代证明  | 展示至少2轮优化,每轮有"改动→指标变化"对比     | 5分 |

## 评测指标示例:

| 代码块                                                                                                                                                                                                                                                     |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> 1  { 2      "accuracy": 0.78,           // 准确性 3      "avg_latency_ms": 1250,    // 平均延迟 4      "avg_tokens": 450,         // 平均token消耗 5      "cost_per_query": 0.02,    // 单次查询成本 6      "citation_accuracy": 0.85 // 引用准确率(如果做RAG) 7  } </pre> |

## 4.5 工程质量(5分)

| 要求项   | 具体标准                      | 评分 |
|-------|---------------------------|----|
| 代码规范  | 关键函数有注释,变量命名清晰            | 1分 |
| 日志记录  | 记录每次请求的模型、token、耗时、检索片段ID | 2分 |
| 安全防护  | Prompt注入防护、敏感信息脱敏、输入校验    | 1分 |
| Git提交 | 至少10次渐进式提交,不能一次性提交完成      | 1分 |

## 五、加分项(最高+30分)

### 5.1 创新与优化类

| 加分项         | 要求                                | 分值  |
|-------------|-----------------------------------|-----|
| 自动化Prompt调优 | 实现A/B测试或参数网格搜索,有量化数据对比            | +5分 |
| 多模型对比融合     | 对比 $\geq 2$ 个模型并智能选择/融合结果         | +5分 |
| Agent多步协作   | 实现复杂任务的多步工具调用( $\geq 3$ 步),展示决策链路 | +5分 |
| RAG高级优化     | 混合检索(向量+BM25)、重排序、查询改写等高级技术       | +5分 |

### 5.2 业务价值类

| 加分项    | 要求                                                             | 分值   |
|--------|----------------------------------------------------------------|------|
| 实际上线使用 | 在团队内真实使用 $\geq 2$ 周,收集 $\geq 20$ 条用户反馈,并根据反馈迭代优化(需提供反馈记录和改进说明) | +10分 |

### 5.3 AI Coding沉淀类

| 加分项       | 要求                                                  | 分值  |
|-----------|-----------------------------------------------------|-----|
| 深度使用记录    | ai_usage.md详实记录AI使用过程(>1500字, $\geq 3$ 个具体案例,含提效数据) | +5分 |
| Prompt模式库 | 总结 $\geq 3$ 个可复用的Prompt设计模式,每个有效果验证数据               | +5分 |

**重要说明:**

- ⚠ 加分项需提供充分证据(代码、数据、截图、视频)
- ⚠ 不能只写文档说明,必须有实际实现
- ⚠ UI美观不作为主要评分依据,避免前端角色优势
- ⚠ AI Coding深度实践的评判标准见第六章ai\_usage.md模板

## 六、提交要求

### 6.1 必须提交的材料

| 序号 | 材料名称        | 说明                    |
|----|-------------|-----------------------|
| 1  | Git仓库地址     | 含渐进式提交历史(至少10次commit) |
| 2  | 部署访问地址      | 虚拟机IP:端口或在线服务URL      |
| 3  | README.md   | 说明启动方式、技术架构、核心功能      |
| 4  | 评测脚本        | 可独立运行,输出量化指标          |
| 5  | Demo视频      | 3-5分钟,演示功能与优化对比       |
| 6  | ai_usage.md | AI使用记录(防抄袭关键文档)       |

### 6.2 目录组织(建议,非强制)

#### 代码块

```

1 你的项目/
2 └── (自定义的代码结构)
3 └── README.md # 必须,说明如何启动和使用
4 └── ai_usage.md # 必须,记录AI使用过程
5 └── evaluate.sh # 必须,评测脚本(可以是其他名字)
6 └── .env.example # 建议,方便评审人配置

```

```
7 └── data/ # 建议,存放测试数据
8 └── test_cases.json
```

## 灵活性说明:

- 代码目录结构完全自定义
- 评测脚本名字自定义(但需在README中说明)
- 适配不同技术栈和项目组织方式

## 6.3 核心文档要求

### README.md模板

#### 代码块

```
1 # 项目名称
2
3 ## 项目简介
4 (1-2段话说明做什么、解决什么问题)
5
6 ## 快速开始
7
8 ### 环境要求
9 - Node.js 18+ / Python 3.9+
10 - Docker (可选)
11
12 ### 启动步骤
13 1. 安装依赖: `npm install` 或 `pip install -r requirements.txt`
14 2. 配置环境: 复制`.env.example`为`.env`并填写API密钥
15 3. 启动服务: `npm start` 或 `python main.py`
16 4. 访问地址: http://localhost:3000
17
18 ### 评测运行
19 bash evaluate.sh
20
21 ## 技术架构
22 (贴上架构图或文字描述)
23
24 ## 核心功能
25 - [x] 功能1: 大模型调用
26 - [x] 功能2: RAG检索
27 - [x] 功能3: 评测系统
28 - [] 功能4: (未完成的)
```

```
29
30 ## 性能指标
31 | 指标 | 数值 |
32 |-----|-----|
33 | 准确率 | 78% |
34 | 平均延迟 | 1.2s |
35 | Token消耗 | 450/次 |
36
37 ## 团队分工(如果是小组)
38 - 张三: 前端开发、UI设计
39 - 李四: 后端架构、RAG实现
40 - 王五: 评测体系、文档编写
```

## ai\_usage.md模板(重要!)

### 代码块

```
1 # AI工具使用记录
2
3 > 这是防抄袭的关键文档, 请详细记录AI使用过程
4
5 ## 使用的AI工具
6 - Claude Code: 60%代码生成
7 - ChatGPT: Prompt设计与调试
8 - Cursor: 代码补全
9
10 ## 关键代码片段说明
11
12 #### 1. [模块名称,如"RAG检索模块"]
13 **AI生成占比:** 约70%
14
15 **使用的Prompt:**
16 ```
17 实现一个语义检索函数, 使用sentence-transformers生成向量,
18 faiss进行检索, 支持top-k参数...
19 ```
20
21 **AI产出效果:**
22 生成了基础框架, 但检索策略需要手动优化
23
24 **手动修改部分:**
25 - 添加了BM25混合检索
26 - 调整了相似度阈值从0.7到0.8
27 - 修复了内存泄漏问题
```

```
28
29 ---
30
31 ### 2. [另一个模块]
32 ...
33
34 ## Prompt迭代过程
35 (粘贴Prompt优化的3个版本及效果对比)
36
37 ## 时间与效率提升
38 - 传统开发预计: 80小时
39 - 使用AI后实际: 35小时
40 - 效率提升: 56%
41
42 ## 遇到的问题与解决
43 1. AI生成的代码有内存泄漏 → 人工review修复
44 2. Prompt效果不稳定 → 通过评测数据反复调优
45 3. ...
```

## 6.4 提交方式

在指定的飞书/腾讯文档表格中提交:

| 字段     | 说明                                                    |
|--------|-------------------------------------------------------|
| 项目名称   | 你的项目名称                                                |
| Git仓库  | GitHub/GitLab地址                                       |
| 部署地址   | <a href="http://x.x.x.x:xxxx">http://x.x.x.x:xxxx</a> |
| Demo视频 | 视频链接(腾讯视频/B站)                                         |
| 团队成员   | 姓名+工号(如果是小组)                                          |
| 提交时间   | 自动记录                                                  |

提交截止: 11月21日 23:59

⚠ 提交后不可修改,请确保所有链接可访问

## 🛡️ 七、防抄袭与作弊机制

### 7.1 过程留痕检查

| 检查项     | 检查方式          | 判定标准                 |
|---------|---------------|----------------------|
| Git提交历史 | 查看commit记录    | 至少10次渐进式提交,一次性提交扣20分 |
| AI使用记录  | 审查ai_usage.md | 必须详细具体,空洞或过于简单扣10分   |
| 代码注释    | 抽查关键函数        | 缺少中文注释说明扣5分          |
| 提交时间分布  | 查看commit时间    | 集中在最后1-2天扣10分        |

### 7.2 个性化要求(防完全抄袭)

每人提交作业时需在评测输出中包含以下信息:

#### 代码块

```
1 {
2 "submitter_id": "zhangsan_1234", // 提交人ID(姓名_工号后4位)
3 "test_data_theme": "贝壳找房FAQ", // 测试数据主题
4 "prompt_signature": "1234" // 工号后4位(嵌入Prompt中)
5 }
```

#### 个性化设置说明:

- 在系统Prompt中加入工号后4位(如"你是编号1234的助手...")
- 测试数据集选择个性化主题(不能完全相同)
- 评测脚本输出必须包含提交人ID

## 7.3 人工抽查机制

### 重点审查项

评审时重点检查以下内容(不需要专门工具):

#### ai\_usage.md真实性检查

- 是否详实具体,有代码片段说明
- 是否有实际遇到的问题与解决过程
- 语言是否真实(不是AI生成的空话)

#### Git提交历史检查

- 是否至少10次提交
- 提交是否渐进式(不是一次性)
- 提交时间是否分散(不是最后集中)

#### 代码与文档一致性

- ai\_usage.md描述是否与实际代码匹配
- Prompt迭代是否有真实的版本记录
- 评测数据是否真实可信

#### 现场答辩(抽查部分作业)

如果怀疑抄袭或造假,安排10分钟答辩:

##### 1. 代码讲解(5分钟)

- "请解释你的XX模块实现思路"
- "为什么这样设计?"

##### 2. 现场小改动(5分钟)

- "把top-k从5改到10,重新跑评测"
- "在Prompt中加一条新约束"
- 看是否能快速定位并修改

#### 答辩判断:

- 完全答不上来 → 怀疑抄袭,扣30-50分
- 磕磕绊绊但能说清 → 可信,不扣分

- 流畅清晰 → 真实完成

## 7.4 合作规范

### ✓ 允许的合作

- 讨论技术方案与实现思路
- 分享公开的技术文档和教程
- 互相code review提建议
- 讨论Prompt设计方法(但不能复制)

### ✗ 严禁的行为

- 直接复制他人代码(超过20行连续相同)
- 共享核心Prompt模板(可讨论但不能照抄)
- 提交相同的评测数据集
- 一人完成多人署名

## 组队说明(2-3人)

- 必须在README.md中明确分工
- 每人必须独立完成至少1个核心模块
- 评审时会单独询问每个成员的负责部分
- 组队作品仍需满足所有技术要求

## 7.5 组队分差机制

针对组队项目,将项目整体得分按贡献度分配给各成员,避免"大锅饭"

### 评分公式

个人最终得分 = 项目得分 × 个人贡献系数

#### 说明:

- **项目得分:** 按第四、五章节标准评定(100基础分+最高30加分)
- **贡献系数:** 根据组内排名确定,用于将项目分分配给个人

### 贡献系数设定

| 组内排名 | 贡献系数 | 适用情况 |
|------|------|------|
|------|------|------|

|     |           |                                    |
|-----|-----------|------------------------------------|
| 第1名 | 0.95-1.00 | 主要贡献者,负责核心模块(如RAG引擎、Prompt优化、评测体系) |
| 第2名 | 0.85-0.95 | 均衡贡献者,负责重要模块(如前后端开发、API设计)         |
| 第3名 | 0.70-0.85 | 辅助贡献者,负责支撑工作(如部署、文档、UI美化)          |

### 系数约束:

- 组内第1名最高系数为1.0(不能超过单人独立完成)
- 组内所有成员系数总和 $\approx$ 组内人数 $\times$ 0.85-0.90(体现协作分工的成本)

### 示例场景

#### 场景1: 3人组队,项目得分85分

- 张三(第1名,系数1.00): $85 \times 1.00 = 85.0$ 分
- 李四(第2名,系数0.90): $85 \times 0.90 = 76.5$ 分
- 王五(第3名,系数0.75): $85 \times 0.75 = 63.8$ 分
- 系数总和: $1.00 + 0.90 + 0.75 = 2.65$ (3人 $\times$ 0.88)

#### 场景2: 2人组队,项目得分100分(满分)

- 张三(第1名,系数1.00): $100 \times 1.00 = 100$ 分
- 李四(第2名,系数0.80): $100 \times 0.80 = 80$ 分
- 系数总和: $1.00 + 0.80 = 1.80$ (2人 $\times$ 0.90)

#### 场景3: 3人组队,项目得分120分(含加分项)

- 张三(第1名,系数1.00): $120 \times 1.00 = 120$ 分
- 李四(第2名,系数0.88): $120 \times 0.88 = 105.6$ 分
- 王五(第3名,系数0.75): $120 \times 0.75 = 90$ 分

### 对比单人:

- 单人拿同样120分项目:直接得**120分**
- 组队第1名:最高120分(与单人持平)
- 组队第2/3名:明显低于单人(105分/90分)

**最大分差:**组内最高与最低可相差约30-35分(视项目得分和贡献差异)

## 排名依据

| 依据      | 权重  | 统计方式                       |
|---------|-----|----------------------------|
| Git提交数据 | 40% | 提交次数、有效代码行数、核心文件修改         |
| 模块复杂度   | 35% | 核心技术模块(高)>前后端框架(中)>文档部署(低) |
| 答辩表现    | 25% | 能否流畅讲解负责模块,现场改动熟练度         |

## 提交要求

在README.md中添加"团队分工与贡献度"章节:

| 代码块                                                                                                                                                                                                                                                                                                                                     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>1  ## 团队分工与贡献度 2 3  ### 成员分工 4    姓名   负责模块   建议系数   5   ----- ----- -----  6    张三   RAG检索引擎、Prompt优化、评测体系   1.0   7    李四   前端界面、后端API   0.95   8    王五   文档编写、部署配置   0.85   9 10 ### Git贡献统计 11 - 张三: 45次提交, 1200行代码(核心逻辑) 12 - 李四: 30次提交, 800行代码(前后端) 13 - 王五: 10次提交, 300行代码(文档+配置) 14 15 ### 组内共识 16 全体成员已讨论并认可此分工,无异议。</pre> |

## 八、评审流程与评分标准

### 8.1 快速评审流程(每份15分钟)

## 第1步: 环境启动(3分钟)

### 代码块

```
1 # 1. 克隆仓库
2 git clone <repo_url>
3 cd <project_dir>
4
5 # 2. 检查Git提交历史
6 git log --oneline | head -20
7
8 # 3. 配置环境变量
9 cp .env.example .env
10 # 填入统一的测试密钥
11
12 # 4. 启动服务
13 bash start.sh # 或按README说明启动
14
15 # 5. 检查服务状态
16 curl http://localhost:3000/health
```

### 检查点:

- 能否一键启动(不能则扣10分)
- Git提交是否渐进(少于10次扣分)

## 第2步: 功能验证(5分钟)

- 打开前端界面, 测试基础功能
- 查看是否有引用标注/工具调用轨迹
- 尝试一个失败案例(如无关问题)
- 检查错误处理是否优雅

### 检查点:

- 功能是否完整
- 交互是否流畅
- 异常处理是否合理

## 第3步: 评测验证(3分钟)

```
1``` # 运行评测脚本
2 bash evaluate.sh
3
4 # 查看输出指标
5 cat metrics.json
```

### 检查点:

- 评测脚本能否正常运行
- 指标是否达到基线(准确率 $\geq 60\%$ , 延迟 $\leq 5s$ )
- 是否有2轮以上优化对比

### 第4步: 文档检查(4分钟)

- 阅读ai\_usage.md是否详实
- 查看Prompt迭代过程
- 检查架构图是否清晰
- 观看Demo视频(快进查看重点)

### 检查点:

- ai\_usage.md是否真实详细
- Prompt迭代是否有数据支撑
- 文档是否规范完整

## 8.2 评分表

| 维度        | 满分 | 评分要点                             |
|-----------|----|----------------------------------|
| 基础架构      | 20 | 前后端完整8分、部署4分、API 6分、启动2分         |
| 大模型应用     | 25 | Prompt迭代10分、模型调用5分、参数调优5分、异常处理5分 |
| RAG/Agent | 25 | 向量检索10分、引用标注5分、文档处理5分、知识管理5分     |
| 评测优化      | 20 | 测试集5分、自动化5分、量化指标5分、迭代证明5分        |

|      |     |                          |
|------|-----|--------------------------|
| 工程质量 | 10  | 代码规范2分、日志3分、安全3分、Git提交2分 |
| 小计   | 100 | 基础分                      |
| 加分项  | +30 | 创新优化、工程深度、安全对抗、业务价值      |
| 扣分项  | -50 | 无法启动-10、抄袭-30、提交不规范-10   |
| 总分   | 130 | 最高130分(100基础+30加分-扣分)    |

## 8.3 分档标准

| 等级  | 分数区间   | 占比  | 说明              |
|-----|--------|-----|-----------------|
| 优秀  | ≥90分   | 20% | 技术全面,有创新点,工程质量高 |
| 良好  | 70-89分 | 60% | 满足所有基础要求,部分有亮点  |
| 及格  | 60-69分 | 15% | 基本满足要求,但有明显不足   |
| 不及格 | <60分   | 5%  | 重要功能缺失或无法运行,需补交 |

### 优秀作品特征:

- ✓ 所有基础要求都完成得很好
- ✓ 至少有2-3个加分项
- ✓ 工程质量高,文档规范
- ✓ 有创新或深度思考

### 不及格情况:

- ✗ 无法启动或功能严重缺失
- ✗ 没有评测体系
- ✗ Prompt迭代缺失或造假

-  明显抄袭

## 九、常见问题FAQ

### Q1: 必须使用特定的技术栈吗?

A: 不必须。后端禁止Java(增加挑战),其他语言自由选择。前端推荐React/Vue,但不强制。

### Q2: 可以使用开源项目改造吗?

A: 可以,但必须:

- 在README中说明基于哪个开源项目
- 在ai\_usage.md中说明你的改动
- 核心功能(Prompt优化、评测等)必须是你自己做的
- 不能只是简单配置,必须有实质性开发

### Q3: 评测数据集怎么准备?

A:

- 至少30条测试用例
- 可以是真实业务问题,也可以是构造的
- 需要包含:问题、预期答案/行为、难度等级
- 建议参考案例中的test\_cases.json格式

### Q4: Prompt迭代一定要3轮吗?

A: 至少3轮,可以更多。每轮必须:

- 记录完整的Prompt文本
- 说明改动原因
- 提供量化的效果对比(不能只说"更好了")

### Q5: 必须做RAG吗?可以只做Agent吗?

A: RAG和Agent二选一即可,也可以两个都做(会有加分)。但必须做到:

- RAG: 实现向量检索+引用标注
- Agent: 至少2个工具+清晰的决策链路

## Q6: 部署到哪里?公司提供虚拟机吗?

A:

- 公司会提供虚拟机资源(需要自己申请)
- 也可以用自己的云服务器
- 或使用免费的云平台(Vercel、 Railway等)
- 关键是提交时必须能访问

## Q7: Demo视频要包含什么内容?

A: 3-5分钟视频,必须包含:

1. 功能演示(正常使用流程)
2. 失败案例(如何处理异常)
3. 优化对比(展示迭代前后效果差异)
4. 可选:架构讲解、亮点介绍

## Q8: 组队怎么分工?怎么评分?

A:

- 2-3人组队,在README中明确分工
- 每人必须独立负责至少1个核心模块
- 评审时可能单独询问每个成员
- 组队作品标准不降低,但可以做得更完善

## Q9: 加分项是必须做的吗?

A: 不是必须。加分项是在100分基础上的额外加分,做得好可以到130分。建议根据时间和能力选择1-2个加分项深入做。

## Q10: 如果作业做不完怎么办?

A:

- 优先保证基础功能完整(80分线)
- 宁可功能少但做精,不要功能多但都不完整
- 如果实在做不完,可以在README中说明未完成部分和原因

- 11月21日是硬性截止,不接受补交

让我们一起通过实践成为真正的AI Native开发者! 🚀

有任何问题,随时在群里提问,我们会持续更新FAQ文档。

预祝大家都能做出优秀的作品! 💪