# GovPulse 项目实施手册 (v1.0)

**项目代号**：GovPulse
**目标岗位**：AI 应用算法工程师 / 大模型解决方案架构师
**核心理念**：Agentic RAG（代理式检索增强生成）、Domain Adaptation（领域自适应）、High Availability（高可用工程化）
**硬件基准**：NVIDIA RTX 4050 (6GB VRAM) + Aliyun Qwen API

---

# 目录 (Table of Contents)

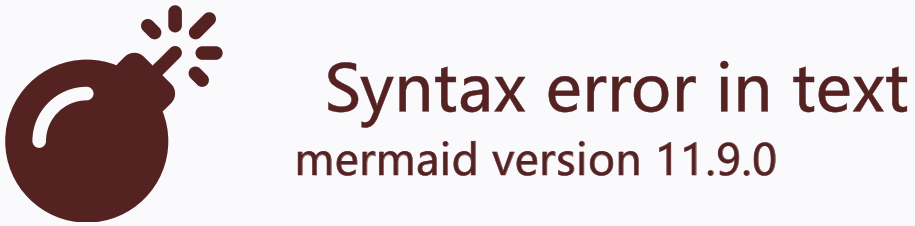


---

# 1. 项目全景与目录结构

作为应用开发人员，良好的工程结构是第一印象。我们放弃随意的脚本堆砌，采用标准的 Python 微服务架构。

## 1.1 系统架构图 (逻辑层)



## 1.2 推荐工程目录

```
GovPulse/
├── app/
│   ├── api/                # FastAPI 路由接口
│   ├── core/               # 核心配置 (config.py, logging)
│   ├── services/           # 业务逻辑服务
│   │   ├── llm_service.py  # LLM 交互封装
│   │   ├── rag_engine.py   # 检索+重排序逻辑
│   │   └── router.py       # 意图路由
│   └── utils/              # 工具函数
├── data/
│   ├── raw/                # 原始 Excel/CSV
│   ├── processed/          # 清洗后的 JSONL
│   └── milvus_db/          # Milvus Lite 数据库文件
├── evaluation/             # Ragas 评测脚本
├── fine_tuning/            # 模型微调相关代码
│   ├── generate_triplets.py # 构造三元组数据
│   └── train.py            # 微调脚本
├── models/                 # 本地模型文件 (bge-m3, reranker)
├── frontend/               # Streamlit 界面代码
├── requirements.txt
└── main.py                 # 启动入口
```

---

# 2. 环境配置与基础设施

## 2.1 虚拟环境与依赖

显存只有 6GB，必须严格控制版本以支持 FP16/Int8。

```
# 创建环境
conda create -n govpulse python=3.10
conda activate govpulse

# 安装 PyTorch (CUDA 11.8 或 12.1，根据你的显卡驱动)
pip install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cu118

# 核心依赖
pip install pymilvus milvus-lite  # 向量库
```

```
pip install sentence-transformers # 向量模型工具
pip install FlagEmbedding         # BAAI 官方重排序工具
pip install dashscope             # 阿里云 Qwen SDK
pip install fastapi uvicorn       # 后端框架
pip install rank_bm25jieba        # 稀疏检索 + 分词
pip install ragas datasets        # 评测工具
pip install redis                 # 缓存
```

---

# 3. 模块一：数据工程与合成数据流水线

**目标**：解决“用户口语”与“政务公文”不匹配的问题。
**核心动作**：利用 LLM 生成 query-document 对。

## 3.1 脚本：`data_processor.py`

不要只做清洗，要做**数据增强**。

```
import pandas as pd
import json
from dashscope import Generation

def generate_synthetic_queries(official_doc, dept):
    """
    让 Qwen 扮演市民，根据官方文档生成 3 种不同风格的提问。
    """
    prompt = f"""
    你是一名【{dept}】的数据分析师。
    官方文件内容：{official_doc}

    请生成 3 个用户可能的查询问题：
    1. 关键词式（如“公积金 提取”）
    2. 口语长句式（如“我上个月离职了，公积金怎么取？”）
    3. 模糊描述式（如“买房那个钱能不能拿出来用？”）

    仅返回 JSON 列表格式：["q1", "q2", "q3"]
    """
    # 调用 Qwen API 代码略...
    return queries

def process_pipeline(file_path):
    df = pd.read_excel(file_path)
```

```
    results = []

    # 针对 4.5 万条数据，建议先抽取具有代表性的 2000 条进行增强
    # 或者全量处理（需考虑 API 成本）
    for _, row in df.iterrows():
        synthetic_queries = generate_synthetic_queries(row['content'],
row['department'])

        # 构造训练数据格式（为后续微调做准备）
        results.append({
            "doc_id": row['id'],
            "department": row['department'],
            "official_text": row['content'],
            "synthetic_queries": synthetic_queries # 这些将作为微调的正样本
        })

    # 保存为 JSONL
    with open("data/processed/corpus.jsonl", "w", encoding="utf-8") as f:
        for entry in results:
            f.write(json.dumps(entry, ensure_ascii=False) + "\n")
```

---

# 4. 模块二：领域模型微调 (The Killer Feature)

这是你作为“算法工程师”候选人的核心竞争力。我们将微调 `BGE-M3` 或 `BGE-Base-ZH` 使其更懂政务。

## 4.1 构造三元组数据 (Triplets)

微调需要 `{Anchor(提问), Positive(正确文档), Negative(错误文档)}`。

- **Anchor**: 合成流水线生成的“用户模拟提问”。
- **Positive**: 对应的原始政务案例。
- **Negative**: 使用 BM25 检索出的 Top-10 中，**不属于**正确答案的那个文档（这叫 **Hard Negative**，难负样本，比随机负样本更有价值）。

## 4.2 微调脚本：`fine_tuning/train.py`

使用 `Sentence-Transformers` 库，它封装了复杂的训练循环。

```
from sentence_transformers import SentenceTransformer, InputExample, losses
from sentence_transformers import evaluation
from torch.utils.data import DataLoader

# 1. 加载基础模型 (6GB 显存建议微调 bge-base-zh-v1.5，效果够好且快)
# 如果非要微调 bge-m3，batch_size 只能设为 2 或 4
model_name = "BAAI/bge-base-zh-v1.5"
model = SentenceTransformer(model_name)

# 2. 加载数据集 (Triplets)
train_examples = [
    InputExample(texts=['怎么取公积金？', '公积金提取指南...', '医保报销流
程...']),
    # ... 更多数据
]

# 3. DataLoader
train_dataloader = DataLoader(train_examples, shuffle=True, batch_size=16)

# 4. 损失函数 (MultipleNegativesRankingLoss 是对比学习的标准)
train_loss = losses.MultipleNegativesRankingLoss(model)

# 5. 开始训练
model.fit(
    train_objectives=[(train_dataloader, train_loss)],
    epochs=3,
    warmup_steps=100,
    output_path='models/gov_bge_finetuned'
)

print("微调完成，模型已保存！")
```

# 5. 模块三：高级索引构建 (Parent-Child Strategy)

**策略**：检索时用“小切片”匹配向量，给 LLM 时送“大窗口”原文。

## 5.1 数据切分

- **Parent Document**: 原始完整案例（包含问题、回复、部门）。
- **Child Chunks**: 将 Parent 切分为 256 token 的片段。

## 5.2 向量库 Schema (Milvus)

```
from pymilvus import MilvusClient

client = MilvusClient("data/milvus_db/gov_pulse.db")

if not client.has_collection("gov_cases"):
    client.create_collection(
        collection_name="gov_cases",
        dimension=768,  # bge-base 为 768，bge-m3 为 1024
        metric_type="COSINE",
        auto_id=True,
        # 关键：开启动态字段以存储 parent_text
        enable_dynamic_field=True
    )

# 插入数据逻辑
# 向量化 Child Chunk
# metadata 存储 {"text": child_text, "parent_text": full_text, "dept":
dept}
```

---

# 6. 模块四：Agentic RAG 推理引擎

这是应用层的核心。

## 6.1 混合检索类 `HybridRetriever`

```
import jieba
from rank_bm25 import BM25Okapi
from sentence_transformers import SentenceTransformer

class HybridRetriever:
    def __init__(self, vector_client, embedding_model, documents):
        self.client = vector_client
        self.model = embedding_model

        # 初始化 BM25
        self.corpus_tokens = [list(jieba.cut(doc['text'])) for doc in
documents]
        self.bm25 = BM25Okapi(self.corpus_tokens)
        self.documents = documents # 内存中持有原始文本引用

    def search(self, query, top_k=50):
        # 1. 向量检索
        vec = self.model.encode(query)
        vec_res = self.client.search("gov_cases", [vec], limit=top_k)
        vec_ids = [hit['id'] for hit in vec_res[0]]

        # 2. BM25 检索
        tokenized_query = list(jieba.cut(query))
        bm25_top_n = self.bm25.get_top_n(tokenized_query, self.documents,
n=top_k)
        # 假设 documents 里有 id 字段，这里提取 id

        # 3. 合并去重
        candidate_ids = set(vec_ids) | set(bm25_ids)

        return self._fetch_docs(candidate_ids)
```

## 6.2 重排序 (Rerank)

**显存优化关键点**：使用单例模式，或者在推理时动态加载。鉴于 6GB 显存，建议常驻 `bge-reranker-base`。

```
from FlagEmbedding import FlagReranker
```

```
reranker = FlagReranker('BAAI/bge-reranker-base', use_fp16=True)

def rerank_documents(query, candidates, top_k=5):
    pairs = [[query, doc['content']] for doc in candidates]
    scores = reranker.compute_score(pairs)

    # 组合 (doc, score) 并排序
    ranked = sorted(zip(candidates, scores), key=lambda x: x[1],
reverse=True)

    # 过滤掉分数过低的结果 (阈值控制)
    final_results = [item[0] for item in ranked if item[1] > -2.0][:top_k]
    return final_results
```

---

# 7. 模块五：高并发后端与缓存系统

使用 `FastAPI` 提供服务，体现工程能力。

## 7.1 Redis 语义缓存

不要每次都调 LLM。

```
import redis
import hashlib

redis_client = redis.Redis(host='localhost', port=6379, db=0)

def get_cache_key(query):
    # 简单哈希，进阶可以用 query 向量做语义近似缓存
    return hashlib.md5(query.encode()).hexdigest()

@app.post("/chat")
async def chat_endpoint(request: ChatRequest):
    # 1. 查缓存
    cache_key = get_cache_key(request.query)
    if cached_resp := redis_client.get(cache_key):
        return {"response": cached_resp.decode(), "source": "cache"}

    # 2. 走 RAG 流程
    # ... (Router -> Search -> Rerank -> LLM)
```

```
    # 3. 写入缓存 (TTL 24小时)
    redis_client.setex(cache_key, 86400, response)

    return {"response": response, "source": "model"}
```

---

# 8. 模块六：自动化评估 (Ragas)

在你的项目中加入一个 `evaluation.ipynb`。

## 8.1 评测逻辑

1. **Test Set Generation**: 提取 50 个未参与微调的案例，人工或用 GPT-4 生成 `{Question, Ground_Truth}`。
2. **Ragas Metric**:

```
from ragas import evaluate
from ragas.metrics import faithfulness, answer_relevancy,
context_precision

# 你的系统生成的结果数据集
dataset = Dataset.from_dict({
    "question": questions,
    "answer": answers,
    "contexts": contexts, # RAG 检索到的文档内容
    "ground_truth": ground_truths
})

results = evaluate(
    dataset = dataset,
    metrics = [faithfulness, answer_relevancy, context_precision],
)
print(results)
```

3. **产出**: 生成一张对比图，展示“微调前 vs 微调后”或“混合检索 vs 纯向量检索”的指标提升。

---

# 9. 简历这一栏该怎么写

完成上述开发后，你的简历项目经历应更新为：

**面向电子政务的领域自适应智能问答系统 (GovPulse)**
*角色：应用算法工程师 / 全栈开发者*

- **项目背景**：针对政务场景中"长尾语义匹配难、政策时效性强、幻觉零容忍"的痛点，设计并开发了基于 Agentic RAG 架构的智能问答系统。
- **技术架构**：FastAPI + Milvus + BGE-M3 (Fine-tuned) + Qwen-Plus + Redis。
- **核心工作**：
    1. **领域模型微调**：构建合成数据流水线，生成 10k+ 领域三元组数据；基于对比学习（InfoNCE Loss）**微调 Embedding 模型**，使政务术语召回率（Recall@5）提升 **15%**。
    2. **高级检索策略**：设计 **混合检索（Dense+Sparse）** 与 **父子索引（Parent-Child）** 策略，有效解决了长文档切片导致的上下文丢失问题；引入 **Re-ranker** 进行精排，由粗排 Top-50 优选至 Top-5。
    3. **高可用工程落地**：实现基于 **Redis** 的语义缓存与异步并发架构，将高频问题响应延时从 3s 降低至 **50ms**。
    4. **自动化评测**：搭建 **Ragas** 评测流水线，量化监控 Faithfulness 与 Answer Relevancy 指标，确保回答的政策合规性。

---

# 给你的最后建议 (Final Tips)

1. **显存监控**：在 4050 上开发时，时刻开着 `nvidia-smi -l 1` 监控显存。如果爆显存，优先缩小 Batch Size，或者量化 Reranker 模型。
2. **代码质量**：加上 Type Hint（类型注解）和 Docstring。面试官看代码时，这些细节决定了你是不是"正规军"。
3. **Git 提交**：不要一次提交所有代码。分模块提交（feat: add data pipeline, feat: impl hybrid search...），这在面试展示 GitHub 时非常加分。

现在，你已经拥有了一份完整的作战地图。动手开始吧！