联邦学习可视化系统 - 软件工程化说明文档

项目概述

本项目是一个基于Flask的联邦学习可视化系统，主要用于LUNA16肺结节检测的分布式机器学习。系统采用多种软件工程化手段，实现了自动化、协作化的开发和部署流程。

---

1. 自动化工程实践

1.1 依赖管理自动化

实施手段：

- 使用 `requirements.txt` 统一管理项目依赖
- 包含54个明确版本的依赖包
- 支持 `pip install -r requirements.txt` 一键安装

技术栈：

```
Flask==3.0.3                    # Web框架
flask_socketio==5.5.1           # 实时通信
torch==2.6.0                    # 深度学习框架
numpy==1.26.4                   # 数值计算
pandas==2.2.2                   # 数据处理
simpleitk==2.5.0                # 医学图像处理
supabase==2.15.2                # 云数据库
```

1.2 配置管理自动化

实施手段：

- 使用 `.env` 文件管理环境配置
- 自动检测和加载环境变量
- 支持本地存储和云存储的自动切换

```
# 自动化配置加载
load_dotenv()
SUPABASE_URL = os.getenv("SUPABASE_URL")
SUPABASE_KEY = os.getenv("SUPABASE_KEY")

# 自动降级机制
if SUPABASE_URL and SUPABASE_KEY:
    supabase = create_client(SUPABASE_URL, SUPABASE_KEY)
else:
    print("🔄 将使用本地存储作为备用")
    SUPABASE_AVAILABLE = False
```

1.3 训练流程自动化

实施手段：

- 自动化联邦学习训练流程
- 自动模型聚合和分发
- 自动生成训练历史和可视化图表

核心组件：

- `FederatedServer`：自动处理模型聚合
- `FederatedClient`：自动执行本地训练
- `FederatedLearningCoordinator`：协调整个训练流程

1.4 数据处理自动化

实施手段：

- 自动医学图像预处理
- 自动数据分片和分布
- 自动生成训练数据集

```
class FederatedLearningCoordinator:
    def distribute_data_automatically(self, data_path):
        """自动分发数据到各个客户端"""
        # 自动扫描数据文件
        # 自动分片和分布
        # 自动验证数据完整性
```

---

2. 协作化工程实践

2.1 模块化架构设计

实施手段：

- 采用MVC架构模式
- 明确的代码分层和职责分离
- 模块化的组件设计

目录结构：

```
code/
├── app.py                      # 主应用入口
├── src/                        # 核心业务逻辑
│   ├── federated_training.py # 联邦学习训练
│   ├── federated_inference.py # 联邦推理
│   ├── federated_inference_utils.py # 推理工具
│   └── simple_inference.py   # 简单推理
├── templates/                  # 前端模板
├── static/                     # 静态资源
│   ├── css/                    # 样式文件
│   ├── js/                     # JavaScript文件
│   └── training_images/        # 训练图像
└── models/                     # 模型存储
```

2.2 实时协作机制

实施手段：

- 基于WebSocket的实时通信
- 多用户协作界面
- 实时状态同步

```
// 实时协作功能
function initializeSocket() {
    socket = io();

    socket.on('connect', function() {
        console.log('WebSocket 连接成功');
        updateConnectionStatus(true);
        socket.emit('request_status_update');
    });

    socket.on('user_status_update', function(data) {
        updateUserStatus(data);
    });
}
```

2.3 用户管理和权限控制

实施手段：

- 基于角色的访问控制（RBAC）
- 客户端和服务端不同权限
- 安全的用户认证机制

```
# 用户角色管理
@app.route("/server_dashboard")
def server_dashboard():
    if "username" not in session or session.get("role") != "server":
        return redirect(url_for("login"))
    # 服务端专用功能

@app.route("/client_dashboard")
def client_dashboard():
    if "username" not in session or session.get("role") != "client":
        return redirect(url_for("login"))
    # 客户端专用功能
```

2.4 数据持久化和共享

实施手段：

- JSON格式的结构化数据存储
- 训练历史的持久化管理
- 多客户端数据共享机制

```
def save_training_history(history_data):
    """保存训练历史数据到JSON文件"""
    history_file = "./training_history.json"
    with open(history_file, "w", encoding="utf-8") as f:
        json.dump(history_data, f, indent=2, ensure_ascii=False)
```

---

3. 质量保证工程实践

3.1 错误处理和异常管理

实施手段：

- 完善的异常处理机制
- 自动错误恢复
- 详细的错误日志记录

```
try:
    supabase = create_client(SUPABASE_URL, SUPABASE_KEY)
    print("✅ Supabase连接成功")
except Exception as e:
    print(f"⚠️ Supabase连接失败: {e}")
    print("🔄 将使用本地存储作为备用")
    SUPABASE_AVAILABLE = False
```

3.2 代码文档化

实施手段：

- 详细的代码注释和文档字符串
- 模块功能说明
- API接口文档

```
"""
联邦学习训练系统 - LUNA16肺结节检测
使用FedAvg算法进行分布式训练

主要组件:
1. FederatedServer: 联邦服务器，负责模型聚合
2. FederatedClient: 联邦客户端，负责本地训练
3. FedAvg算法实现
4. 数据分片和分布
"""
```

3.3 兼容性设计

实施手段：

- 跨平台兼容性（可在电脑端和移动端使用）
- 多浏览器支持（经测试
- 向后兼容的API设计

---

4. 部署和运维自动化

4.1 云服务集成

实施手段：

- Supabase云数据库集成，所有的账号密码都使用密文存储。

4.2 监控和日志

实施手段：

- WebSocket连接状态监控（在用户端可以查看在线人数以及服务端是否在线，服务端可以看到每一个用户端的在线情况）
- 用户活动跟踪（在服务端可以实时看到每一个用户的文件上传情况）
- 训练过程监控（训练时服务端和客户端都可以实时查看当前的训练情况）

```
// 连接状态监控
socket.on('disconnect', function() {
    console.log('WebSocket 连接断开');
    updateConnectionStatus(false);
    // 自动重连机制
    if (reconnectAttempts < maxReconnectAttempts) {
        setTimeout(() => {
            reconnectAttempts++;
            console.log(`尝试重连... (${reconnectAttempts}/${maxReconnectAttempts})`);
            socket.connect();
        }, 2000 * reconnectAttempts);
    }
});
```

---

5. 采用的软件工程化工具和技术

5.1 开发框架和库

- **Flask**: Web应用框架
- **SocketIO**: 实时通信
- **PyTorch**: 深度学习框架
- **SimpleITK**: 医学图像处理

5.2 数据管理

- **JSON**: 结构化数据存储
- **Supabase**: 云数据库服务
- **pandas**: 数据处理和分析

5.3 前端技术

- **JavaScript ES6+**: 现代前端开发
- **WebSocket**: 实时通信
- **CSS3**: 响应式设计

5.4 部署和运维

- **gunicorn**: WSGI服务器
- **dotenv**: 环境变量管理
- **bcrypt**: 密码加密