驾驶员危险行为检测系统架构设计文档

目录

1 架构样式展示

根据系统需求，本项目采用复合架构风格，以三层架构为核心，确保逻辑清晰、职责分离，同时融入客户端-服务器通信模式，支持实时性和分布式部署。

1.1 宏观风格：三层架构

系统在逻辑上被清晰地划分为三个层次，确保了各层职责的独立性，便于开发和维护。

- **表现层**：负责用户交互和数据展示。这包括运行在 PC 端的 **Vue.js 前端 Web 界面**，以及运行在客户端 Python 程序中，通过 OpenCV 实时显示的视频窗口和报警提示。
- **逻辑层**：负责处理业务逻辑和数据。这主要是指 **Flask 后端服务器**，它提供 RESTful API 接口，处理数据存储、查询、统计分析等核心业务。
- **数据层**：负责数据的持久化存储和管理。由 **PostgreSQL 数据库** 构成。

1.2 通信风格：客户端-服务器架构

- **胖客户端**：执行核心实时监测任务的 **Python 应用程序**。它在本地完成视频流处理、AI 模型推理和危险行为分析等重计算任务，然后仅将结果（报警事件）发送给服务器。
- **瘦客户端**：管理员使用的 **Web 浏览器**。它只负责渲染后端返回的数据和界面，所有业务逻辑均由服务器处理。
- **服务器**：**Flask 应用程序** 作为中心服务器，响应来自胖客户端的数据上报请求和来自瘦客户端的数据查询请求。

2 参考模型说明

为全面描述系统架构，我们选用 **4+1 视图模型** 作为理论框架。该模型通过多视角描绘系统，确保利益相关者（如开发者关注模块化、管理员关注可用性）获得针对性信息，避免单一视图的偏差。

1. **逻辑视图**：描述系统的功能结构，主要关注系统为用户提供的服务。
2. **过程视图**：描述系统的动态行为，即系统运行时各个进程、线程间的交互和并发。

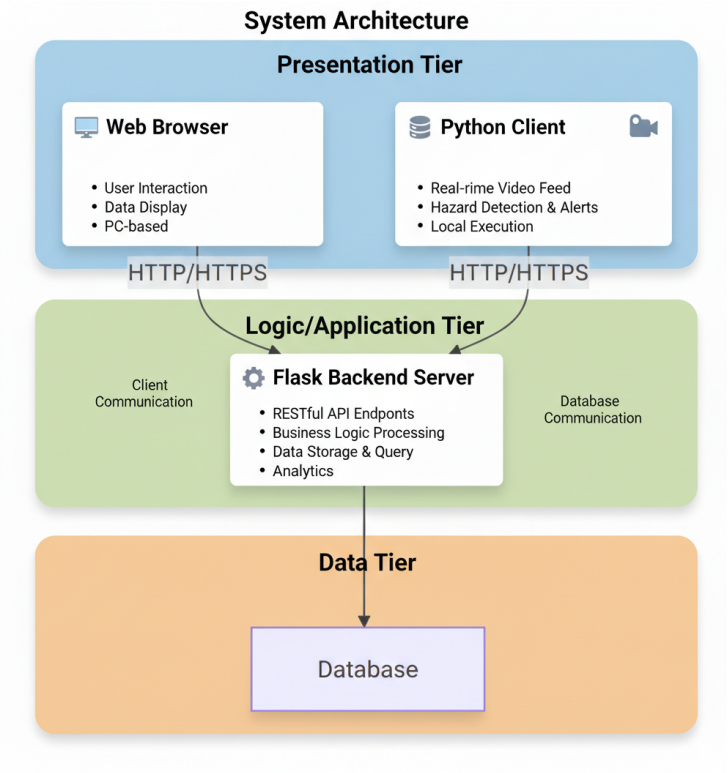

图 1: 三层架构与客户端-服务器架构图

3. **开发视图：**描述系统的模块划分和组织，即软件在开发环境中的静态结构。
4. **物理视图：**描述系统如何部署到硬件节点上，以及各节点间的物理连接。
5. **场景视图：**作为核心，用于连接和验证其他四个视图，通过具体的用例或场景来展示架构的合理性。

3 参考架构分析

应用 4+1 视图模型，从多视角剖析系统架构，确保覆盖功能、动态、开发与部署维度。

3.1 逻辑视图

此视图展示了系统的主要功能模块及其关系（见于图2）。

3.2 过程视图

此视图通过一个核心时序图展示“检测到危险行为并上报”的动态过程（见于图3）。

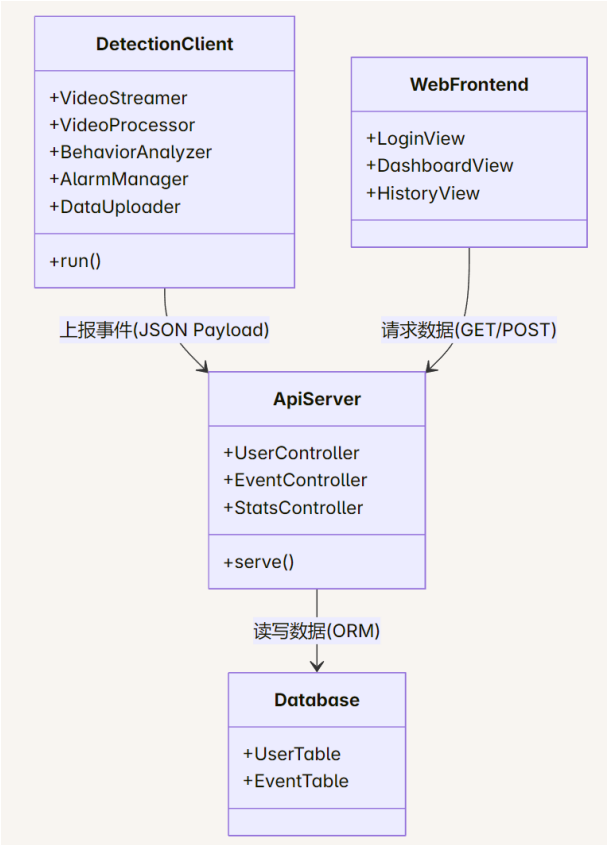

图 2: 逻辑视图类图

3.3 开发视图

此视图展示了项目的代码组织结构（见于图4）。

3.4 物理视图

此视图描述了系统的部署方式（见于图5）。

3.5 场景视图

此视图通过关键的用户场景驱动和验证架构设计，确保系统满足核心需求。以下为三个核心场景：

3.5.1 场景一：实时危险行为识别与报警

- **描述：**驾驶员行驶中长时间闭眼，客户端需在500ms内通过本地扬声器和屏幕发出声光报警。
- **架构验证：**

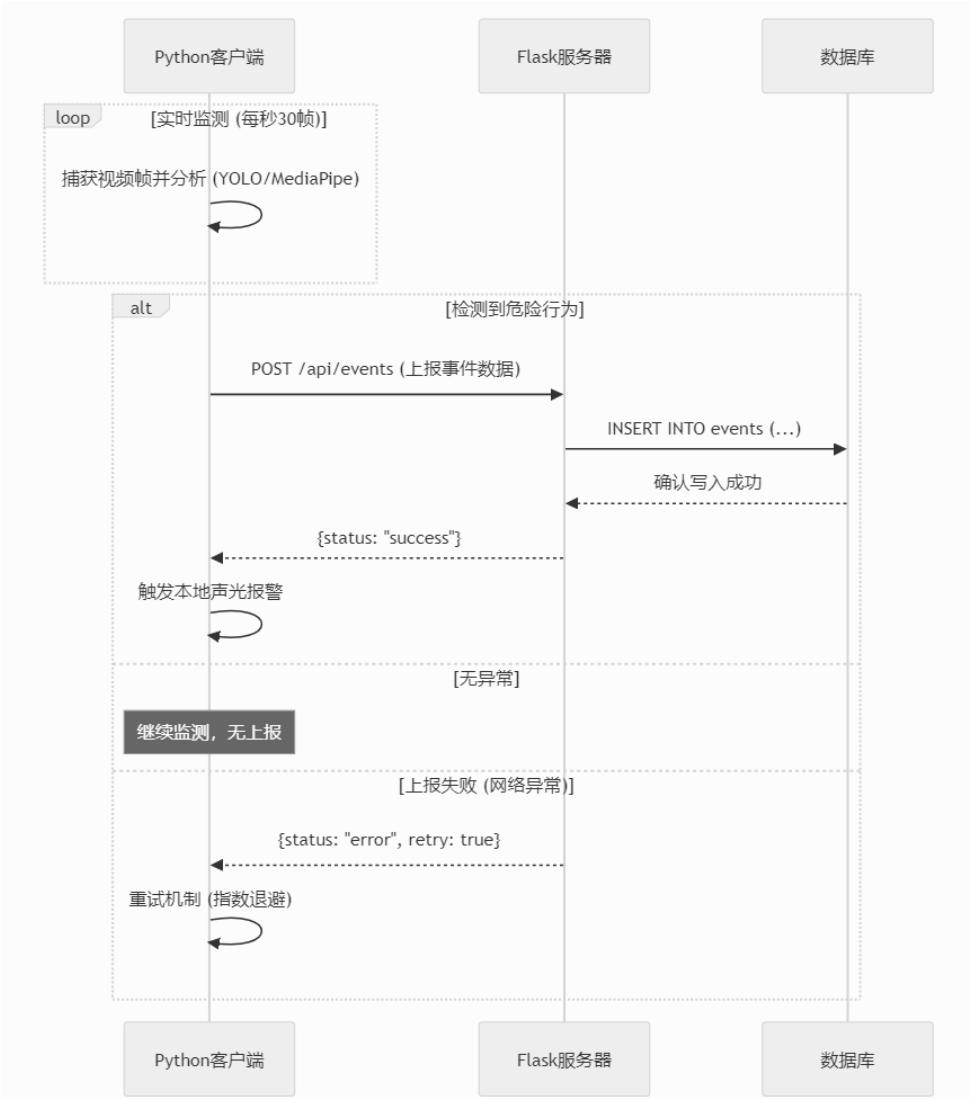

图 3: 过程视图时序图

1. **物理视图：**在客户端PC上，Python客户端通过USB摄像头捕获视频，依托PC硬件执行推理，扬声器和屏幕输出报警，验证胖客户端独立性。
2. **逻辑视图：**`DetectionClient` 模块集成 `VideoStreamer`（捕获视频）、`VideoProcessor`（预处理）、`BehaviorAnalyzer`（YOLO+MediaPipe推理）和 `AlarmManager`（触发报警），模块职责清晰。
3. **过程视图：**对应时序图的“实时监测循环”和“触发本地声光报警”分支，验证动态流程。
4. **开发视图：**`client/detector.py` 封装AI模型，`client/analyzer.py` 实现闭眼检测逻辑，单例模式确保模型高效复用。

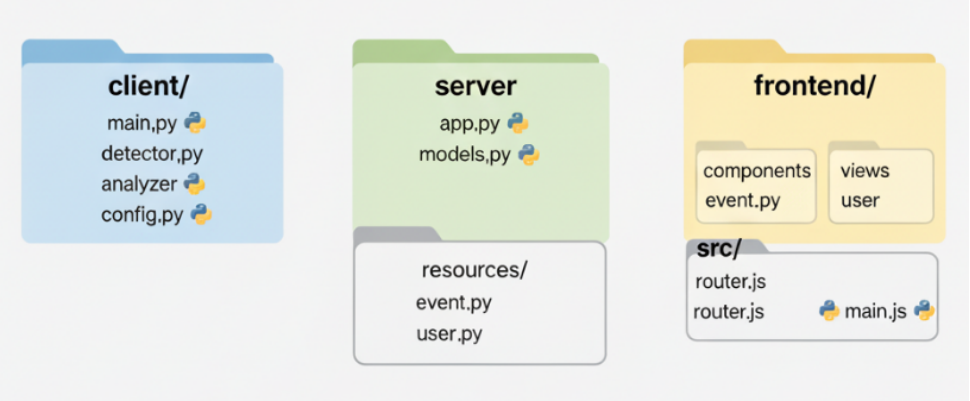

图 4: 代码结构

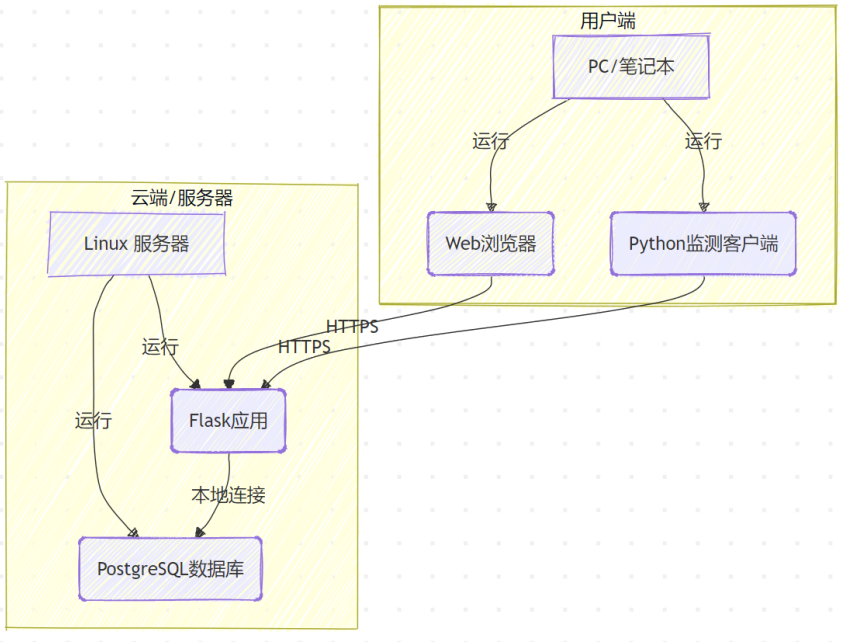

图 5: 物理视图部署图

3.5.2 场景二：危险行为事件上报

- **描述**：场景一触发报警后，客户端将事件（时间、类型、驾驶员ID）通过HTTPS上报至后端，持久化存储，支持网络中断重试。
- **架构验证**：
  1. **物理视图**：客户端PC与云端Linux服务器通信，Flask应用接收JSON数据，存入PostgreSQL，验证C/S架构网络部署。
  2. **逻辑视图**：`DetectionClient.DataUploader` 与 `ApiServer.EventController` 交互，`ApiServer` 通过ORM写入 `Database.EventTable`，验证三层数据流。
  3. **过程视图**：对应时序图的“alt [检测到危险行为]”分支，包括重试逻辑（指数退避）。

4. **开发视图：**`client/main.py` 调用 `requests.post`，与 `server/resources/event.py` 的API端点交互，`server/models.py` 封装ORM操作。

3.5.3 场景三：管理员查询历史数据

- **描述：**管理员登录Web后台，筛选并查看过去一周的“疲劳驾驶”报警记录，响应时间¡5秒。
- **架构验证：**

1. **物理视图：**管理员通过客户端PC的Web浏览器（Chrome）访问云端Flask服务器，服务器从PostgreSQL检索数据，验证B/S架构部署。
2. **逻辑视图：**`WebFrontend` 的 `LoginView` 和 `HistoryView` 与 `ApiServer` 的 `UserController`（认证）和 `EventController`（查询）交互。
3. **过程视图：**如下时序图展示交互序列，验证查询流程。
4. **开发视图：**`frontend/src/views/HistoryView.vue` 通过Axios发送GET请求至 `server/resources/event.py` 的查询端点，`server/models.py` 执行SQLAlchemy条件查询，前端组件复用性和后端查询优化降低开发成本。

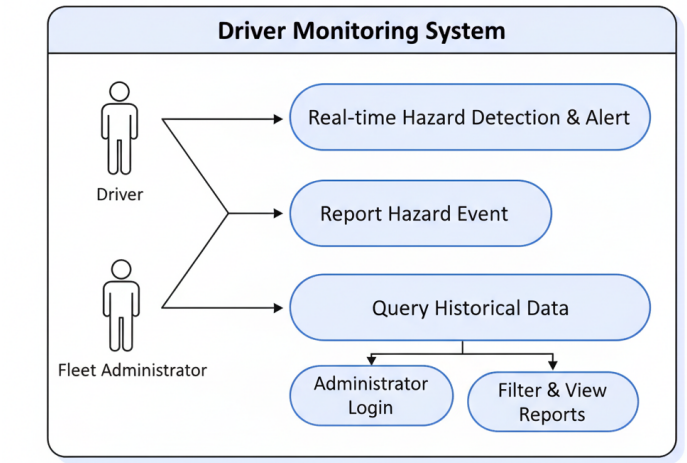

图 6: 场景视图

4 质量场景描述

为确保架构满足关键非功能性需求，我们定义以下质量属性场景。

表 1: 质量属性场景

场景	质量属性	来源	刺激	制品	环境	响应	响应度量
1	性能	驾驶员	发生长时间闭眼等危险驾驶行为	监测客户端	正常运行中	系统发出首次警报	平均延迟低于 500 毫秒
2	可修改性	开发者	需要增加一种新的危险行为（如“打电话检测”）	客户端代码	开发阶段	开发者只需修改 `analyzer.py` 模块，无需改动其他模块	修改和测试工作量小于 4 个工时
3	易用性	管理员	希望查看某驾驶员过去一周的危险行为记录	前端 Web 界面	正常登录后	管理员进入历史查询页面，选择条件后看到结果	操作步骤不超过 3 次点击，数据加载时间小于 5 秒

5 框架与设计模式应用

5.1 框架与核心库选择

- **Python + OpenCV**：视频处理的事实标准，生态成熟，性能可靠。
- **Ultralytics YOLO**：提供高性能的实时目标检测能力，是本项目的技术基石。
- **MediaPipe**：用于精确的人脸关键点检测，判断眼部、口部状态，性能优于传统方法。
- **Flask**：轻量级、灵活的 Python Web 框架，适合快速开发 API 服务，学习曲线平缓。
- **Vue.js**：现代化、渐进式 JavaScript 框架，组件化开发提升前端开发效率和可维护性。

5.2 设计模式应用

- **单例模式 (Singleton Pattern)**：在 Python 客户端中，`YOLO` 和 `MediaPipe` 模型加载耗时且占用大量内存，封装为单例，确保程序生命周期内只实例化一次。
- **策略模式 (Strategy Pattern)**：在 `analyzer.py` 中，将每种危险行为的判断逻辑（如 *疲劳检测策略*、*分心检测策略*）封装成独立策略类，符合开闭原则。
- **数据访问对象/仓库模式 (DAO/Repository Pattern)**：在 Flask 后端，创建数据访问层封装数据库操作，逻辑层仅与该层交互，便于数据库切换。
- **观察者模式 (Observer Pattern)**：`BehaviorAnalyzer` 作为主题，检测到危险行为时通知 `AlarmManager`（本地报警）和 `DataUploader`（数据上报），实现功能解耦。

6 优先级划分策略

采用 **MoSCoW** 方法划分功能优先级，确保核心价值优先交付。

- **Must-have (必须完成)**：构成最小可行产品（MVP）的核心功能。
  - 客户端通过摄像头实时检测至少两种核心危险行为（如 **闭眼疲劳** 和 **低头分心**）。
  - 检测到危险行为后，客户端进行 **本地声光报警**。
  - 客户端将报警事件上报给后端并存入数据库。
  - 后端提供基础的管理员 **用户登录** 功能。
- **Should-have (应该完成)**：重要功能，若时间允许优先实现。
  - 前端 Web 界面以 **列表形式查询和展示** 历史报警记录。
  - 支持更多危险行为检测（如 **手部异常**、**打电话**）。
  - 前后端用户认证机制完善。
- **Could-have (可以完成)**：锦上添花的功能，优先级较低。
  - 前端实现 **数据可视化图表**（如危险行为类型分布饼图）。
  - 管理员可对报警的 **灵敏度进行配置**。
  - 支持查询结果导出为 PDF 或 Excel。
- **Won't-have (本次不做)**：明确在此版本范围之外的功能。

– 移动端 App 的开发。
– 远程设备管理与固件升级。
– 与物理硬件（GPS、4G 模块）的深度集成。