5.1 引言 (Introduction)

本章依据 IEEE 1016 标准及第 3 章确立的概念模型，从设计视角对 FitnessAI 智能健身助手系统的内部结构与技术实现进行详细规约。作为软件设计说明书的核心部分，本章旨在将软件需求规格说明书中的功能与非功能需求，转化为可落地的软件架构与组件设计，明确系统在静态结构、动态行为及数据组织上的具体形态。

5.1.1 设计目标与策略

FitnessAI 的设计紧密围绕 **3.1.2 节** 所定义的关键设计关注点展开。针对"实时性能与低延迟响应"的约束，本设计采用了 **端侧计算为主、云端编排为辅** 的策略，将姿态识别与实时纠错闭环下沉至前端浏览器环境，以确保满足高频交互下的低延迟要求；针对"数据安全与隐私保护"的约束，设计上确立了 **数据最小化流转** 原则，严格界定原始视频流与抽象关键点数据的边界，确保敏感数据不越过前端边界。同时，为响应"模块化与可维护性"要求，系统后端被设计为无状态的业务编排中枢，通过 RESTful API 契约与前端及数据层解耦。

5.1.2 设计视角的组织

为全面、清晰地表达 FitnessAI 的设计方案，本章采用多视角描述方法，涵盖以下维度：

- **上下文视角（Context Viewpoint）：** 承接 **3.1.1 节** 定义的设计边界，进一步明确系统与终端用户、浏览器运行时环境、外部算法库（MediaPipe, TensorFlow.js）及硬件设备（摄像头、GPU）之间的交互界面与数据 I/O 关系。（详见 **5.2 节**）
- **组合视角（Composition Viewpoint）：** 描述系统的物理分解结构，重点阐述前端应用容器（React SPA）、后端服务容器（Flask RESTful API）与数据库组件（PostgreSQL）的装配关系，以及各子系统在"前后端分离"架构下的职责划分与部署拓扑。（详见 **5.3 节**）
- **逻辑视角（Logical Viewpoint）：** 定义系统的静态功能结构，通过类图与包图展示核心领域实体（如 `Session`, `Exercise`, `AnalysisResult`）的属性、方法及关联关系。重点描述姿态分析模块的 **策略模式** 实现（`PoseAnalyzer` 抽象基类及其具体运动子类），确保业务语义在代码层面的一致性。（详见 **5.4 节**）
- **依赖视角（Dependency Viewpoint）：** 分析模块间的调用依赖及对关键第三方组件（如 MediaPipe 0.10.x, Flask 2.3.x, PostgreSQL 14+）的版本约束与兼容性要求，为构建管理（Build Management）与变更影响分析提供依据。（详见 **5.5 节**）
- **信息视角（Information Viewpoint）：** 针对 **3.1.2 节** 中的数据一致性关注点，详细规约基于 PostgreSQL 的持久化存储策略。特别是针对非结构化运动数据（如 `Session.scores`, `User.profile`）采用 **JSONB** 字段存储，与结构化业务数据（如账户、鉴权信息）共存的混合存储模型设计。（详见 **5.6 节**）
- **设计模式使用视角（Patterns Use Viewpoint）：** 说明系统中采用的关键设计模式，包括 **策略模式**（封装不同运动类型的分析算法）、**工厂模式**（动态创建分析器实例）及 **仓储模式**（Repository 层封装数据访问），以验证设计在应对未来运动类型扩展（**3.1.2 可扩展性关注点**）时的演化能力。（详见 **5.7 节**）

5.2 上下文视角 (Context Viewpoint)

本节承接 3.1.1 节定义的设计边界，将 FitnessAI 系统视为一个完整的**黑盒实体**，重点描述系统边界、外部交互实体以及跨越边界的数据流转契约。本视角不涉及系统内部的组件划分（相关内容见 5.3 组合视角），而是聚焦于系统作为一个整体如何嵌入到其运行环境中。

5.2.1 系统边界定义

FitnessAI 系统的边界圈定了所有由本设计负责实现的软件组件。

- **系统内部**：所有待开发的软件资产，包括运行在浏览器端的前端业务逻辑、WASM 推理模块，以及运行在云端的后端服务和数据库。
- **系统外部**:
    - **宿主环境**：标准的 Web 浏览器运行时（提供 WebRTC, Canvas, Network 能力）。
        - **硬件环境**：用户设备的摄像头（Video Source）和 GPU（Compute Resource）。
        - **外部服务**：Zhipu AI (GLM) 认知服务接口。

5.2.2 外部实体与交互

系统与以下四个外部实体进行交互：

5.2.2.1 终端用户

用户是系统的核心服务对象和主动触发者。

- **输入**:
    - **物理层**：连续的肢体运动（系统通过摄像头被动捕获）。
    - **逻辑层**：GUI 控制指令（开始、暂停、调整目标）。
- **输出**:
    - **实时反馈**：<100ms 延迟的骨骼可视化覆盖层与计数反馈。
    - **训练报告**：训练结束后生成的自然语言建议与图表。

5.2.2.2 浏览器运行时

这是前端代码的“宿主容器”，系统完全依赖其提供的标准 Web API 能力，不直接调用操作系统内核。

- **职责**：提供硬件抽象层（HAL）和安全沙箱。
- **关键依赖接口**:
    - `MediaDevices.getUserMedia()`：请求摄像头数据流。
    - `WebGL / WebAssembly`：提供底层算力支持本地 AI 推理。
    - `Fetch API`：提供网络传输能力。

5.2.2.3 物理传感器

- **角色**：原始数据源。
- **约束**：系统要求传感器提供分辨率 $\ge$ 640x480 @ 30fps 的 RGB 视频流。

5.2.2.4 外部智能服务

- **角色**：增强型认知引擎（AI Coach）。
- **交互**：系统后端作为客户端，向 Zhipu AI 发送脱敏的统计数据，接收文本建议。

5.2.3 上下文交互图

下图展示了 FitnessAI System 作为中心节点，与其运行环境及外部实体的交互拓扑。系统作为黑盒，通过标准接口与外部世界交换信息。

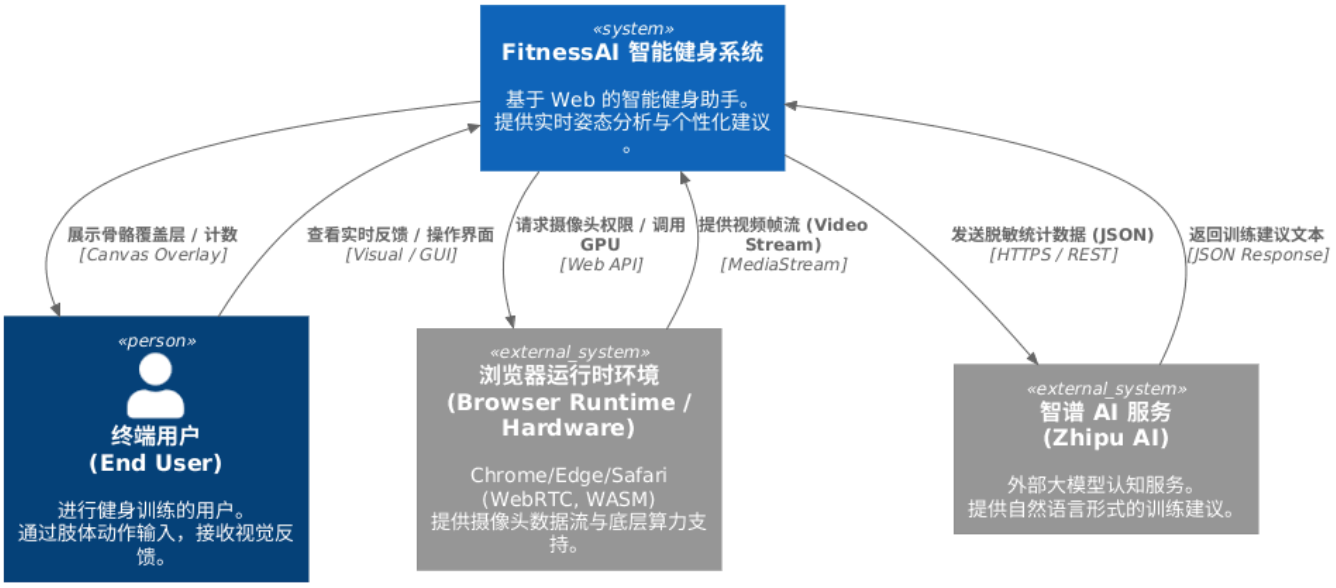


图5-1 上下文交互图

箭头表示数据或控制流的方向，标注文字说明了交互的主要内容。

5.2.4 外部接口契约

系统对外暴露或依赖的接口契约如下，重点关注跨越系统边界的数据交换。

外部实体	交互界面	协议/标准	数据流向	备注
User	GUI / Visual	HCI	双向	视觉反馈闭环
Browser	Web Standard	W3C WebRTC / WASM	双向	获取流/调用算力
Zhipu AI	API Endpoint	HTTPS / OpenAPI	Out/In	请求生成建议

跨边界数据载荷 (Payload) 示例：
当系统与 Zhipu AI 交互时，系统（后端）将发送如下脱敏的 JSON 数据包，此数据包已跨越了系统边界：
JSON

```
{
  "request_type": "training_summary",
  "data": {
    "exercise": "squat",
    "metrics": {
      "total_reps": 15,
      "accuracy_rate": 85.5,
      "major_faults": ["knee_valgus"]
    },
    // 注意：此处不包含任何用户PII（个人身份信息）
    "user_level": "beginner"
  }
}
```

5.3 组合视角 (Composition Viewpoint)

本节采用**白盒视角**，打开 5.2 节定义的系统边界，详细解构 FitnessAI 系统的内部物理组件及其装配关系。系统遵循 **C4 模型** 的层级分解原则，将系统分解为独立部署的子系统和内部功能模块。

5.3.1 系统分解策略

FitnessAI 采用**分层架构**与**前后端分离**的分解策略，依据**高内聚、低耦合**的设计原则将系统划分为三个主要子系统。这种分解策略旨在应对低延迟交互、复杂业务逻辑与数据灵活性的需求。

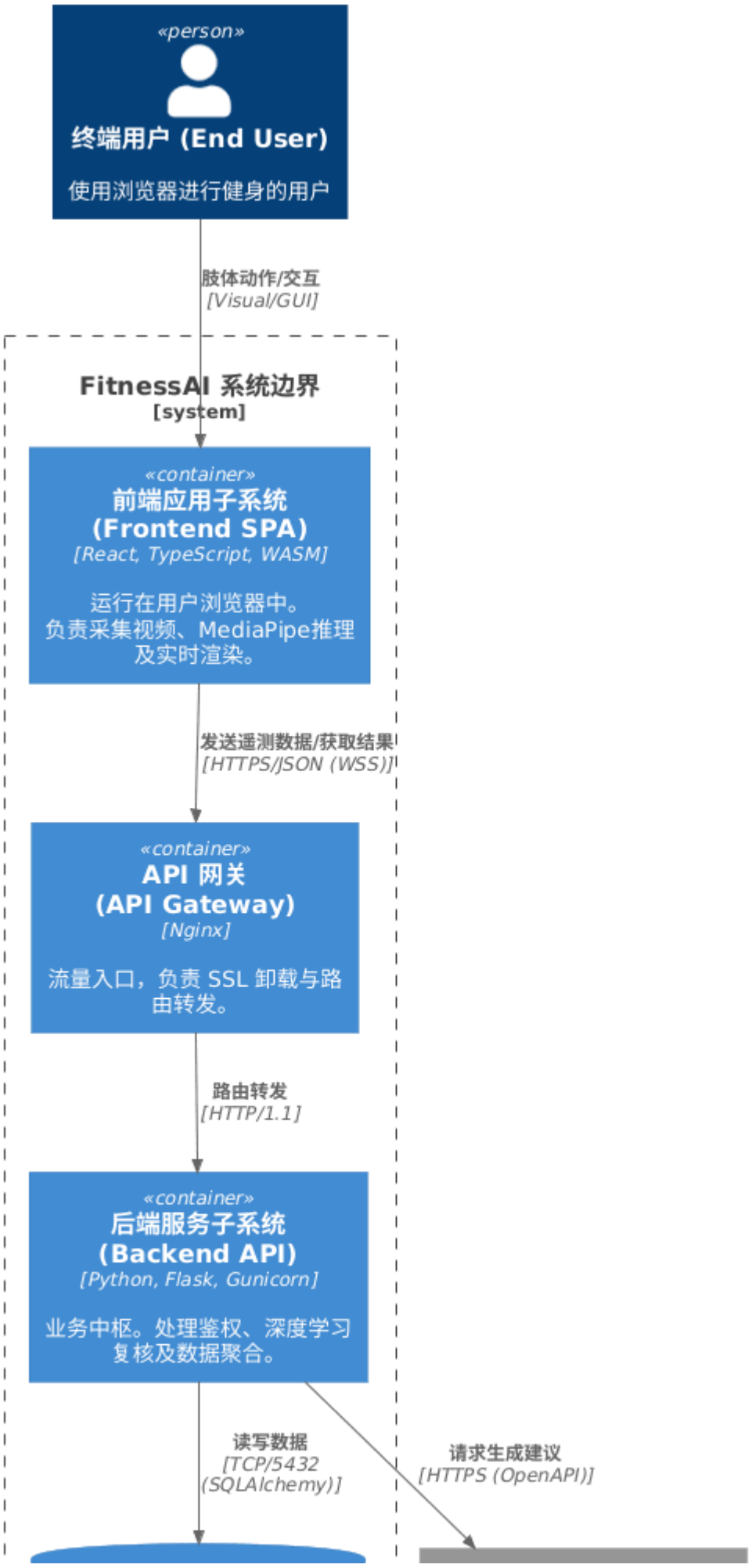

«person»
终端用户 (End User)
使用浏览器进行健身的用户
肢体动作/交互
[Visual/GUI]
FitnessAI 系统边界
[system]
«container»
前端应用子系统
(Frontend SPA)
[React, TypeScript, WASM]
运行在用户浏览器中。
负责采集视频、MediaPipe推理
及实时渲染。
发送遥测数据/获取结果
[HTTPS/JSON (WSS)]
«container»
API 网关
(API Gateway)
[Nginx]
流量入口，负责 SSL 卸载与路
由转发。
路由转发
[HTTP/1.1]
«container»
后端服务子系统
(Backend API)
[Python, Flask, Gunicorn]
业务中枢。处理鉴权、深度学习
复核及数据聚合。
读写数据
[TCP/5432
(SQLAlchemy)]
请求生成建议
[HTTPS (OpenAPI)]

图5-2 系统容器图

5.3.2 子系统分解与职责

5.3.2.1 前端应用子系统

前端应用不仅仅是 UI，更是一个包含感知、推理与决策的**智能富客户端**。它基于 React + TypeScript 构建，内部组件围绕**"实时推理流"**进行组织。为了实现毫秒级的动作反馈，前端设计采用了**流水线模式**。下图展示了从摄像头原始帧采集到最终 UI 呈现的完整数据流向，重点体现了 `MediaPipeAdapter` 如何作为核心引擎驱动整个感知链路。

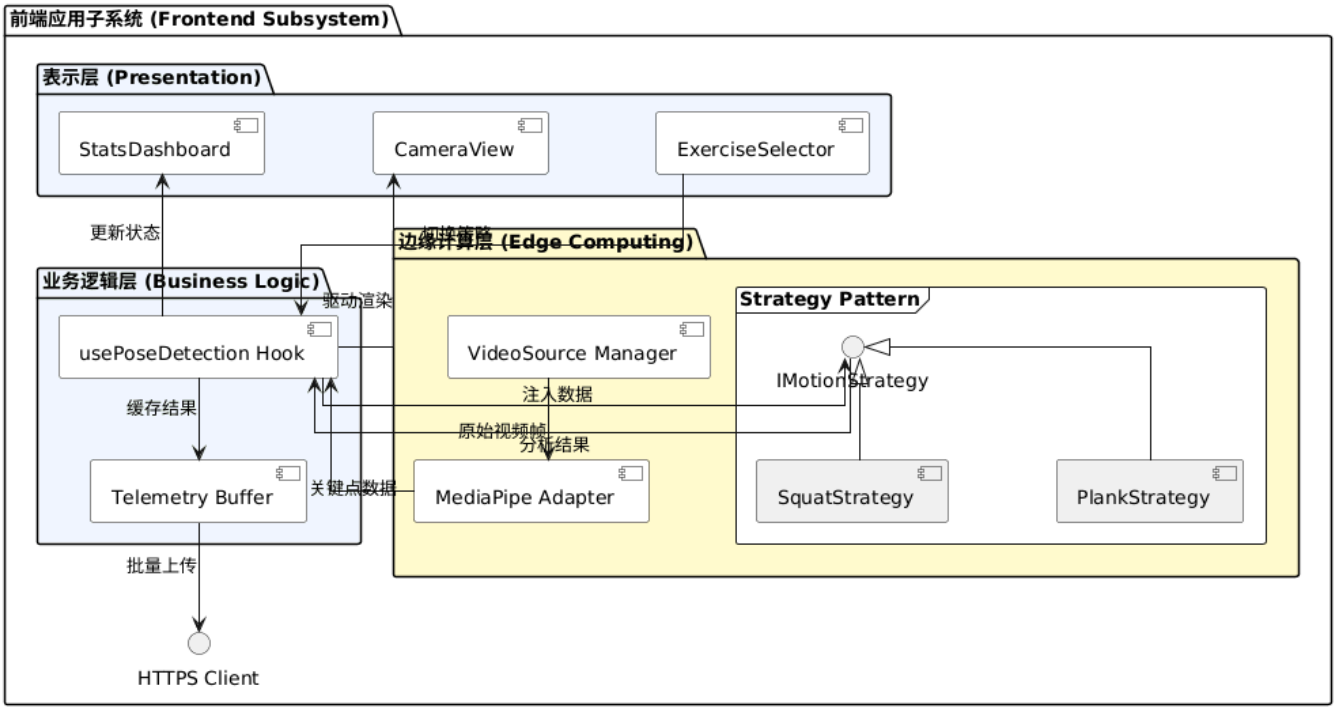


图5-3 前端数据流图

1. 表示层模块

职责：负责页面渲染、Canvas 绘图及用户交互事件捕获。

- **App Container**：应用根容器，集成 React Context 进行全局状态注入（如 AuthToken）。
- **CameraOverlay**：核心视图组件，采用**分层渲染策略**——底层显示原始视频流，上层透明 Canvas 绘制骨骼连线，避免重绘开销。
- **StatsDashboard**：实时仪表盘，通过**观察者模式**订阅分析状态流，以 30fps 频率无闪烁刷新计数与得分。

2. 核心业务逻辑模块

职责：前端的大脑，封装姿态检测生命周期与状态管理。

- **usePoseDetection Hook**：作为**控制器**，协调输入源、适配器和视图层之间的协作。维护 `isDetecting`（检测状态）、`currentRepCount`（计数）等瞬时状态。
- **Telemetry Buffer**：数据缓冲组件。为了防止高频 HTTP 请求阻塞主线程，该组件暂存每帧分析结果，按批次打包发送给后端。

3. 边缘计算集成模块

职责：屏蔽底层 AI 算法库的复杂性，提供标准化的推理接口。

- **MediaPipe Adapter**：单例模式组件，管理 WASM 二进制文件的加载与内存释放。
- **Motion Analyzer Strategy**：采用**策略模式**实现。定义统一接口 `analyze()`，具体策略包括 `SquatStrategy`（深蹲几何计算）、`PlankStrategy`（直线度检测）等。

5.3.2.2 后端服务子系统

后端服务子系统是系统的**业务中枢**，基于 Flask 框架构建，采用经典的**三层架构**设计，强调关注点分离。后端架构的核心目标是**高内聚与低耦合**。通过下方的分层结构图可以看到，系统将复杂的 AI 代理逻辑（Advisor Agent）与基础的 CRUD 业务进行了物理隔离，确保了业务逻辑的纯粹性与外部服务接入的灵活性。

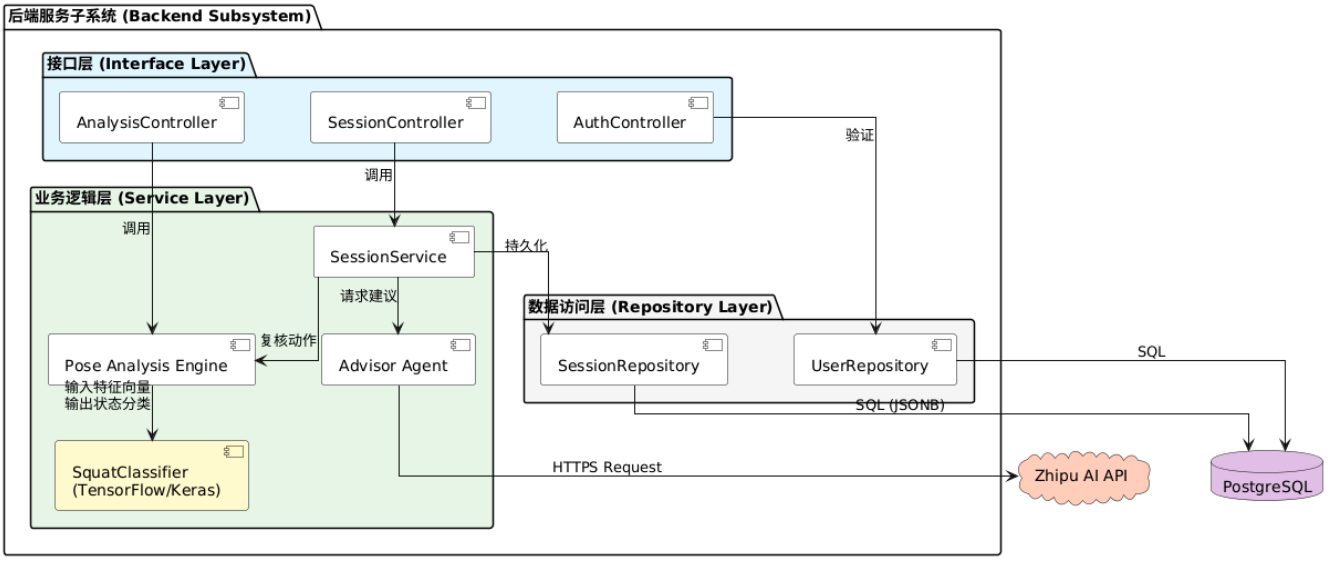


图5-4 后端分层架构图

1. 接口层

职责：流量入口，处理 HTTP 协议解析、参数校验与响应格式化。

- **SessionController**：处理遥测数据上传。
- **AnalysisController**：处理同步分析请求（如请求后端深度学习模型复核）。

2. 业务逻辑层

职责：纯粹的业务规则实现，不依赖 HTTP 或 SQL 细节。

- **Pose Analysis Engine**：执行复杂分析逻辑。
    - **SquatClassifier**：封装 TensorFlow 运行时。加载模型，输入特征向量，输出动作状态分类及置信度，

用于校准前端规则引擎。

- **Advisor Agent**：作为**智谱 AI**的代理客户端。负责将结构化的训练统计数据转换为 Prompt，调用大模型 API，并将返回文本清洗为标准格式。

3. 数据访问层

职责：封装数据库操作，实现数据访问与业务逻辑解耦。

- **SessionRepository**：封装对会话表的操作。实现对 **JSONB** 字段的高效查询逻辑。

5.3.2.3 数据持久化子系统

本系统的数据模型旨在兼顾"用户关系"的严谨性与"运动遥测"的灵活性。下图的 ER 模型（图表 5-5）展示了系统如何利用 PostgreSQL 的 **对象-关系（Object-Relational）** 特性，构建混合存储架构。

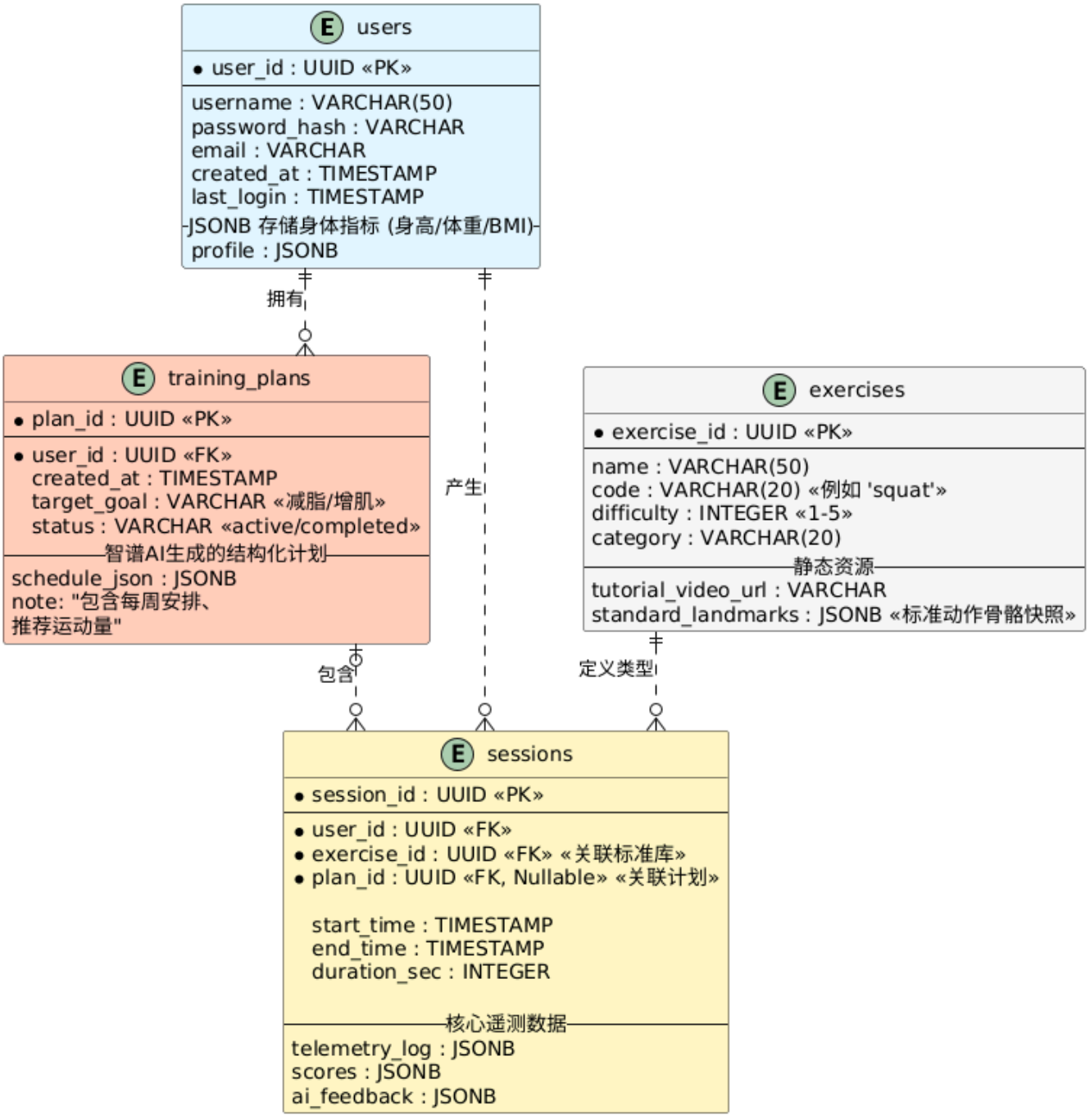


图5-5 实体关系图

职责：提供可靠、可扩展的数据存储，并支撑 AI 业务的数据闭环。

组件：PostgreSQL 实例。

混合数据模型设计：

- **基础数据区域（结构化）**：
  - `users` 表：存储账号与鉴权信息，保证事务的 **ACID** 属性。
  - `exercises` 表：作为系统的**元数据字典**，标准化定义各类运动的参数（如难度、标准视频 URL），支持业务逻辑的动态配置。
- **业务数据区域（半结构化 JSONB）**：
  - `sessions` 表：核心业务表。利用 **JSONB** 字段（`telemetry_log`）存储高维度的动作时序数据与骨骼快照，避免了因运动类型差异导致的频繁表结构变更。
  - `training_plans` 表：存储智谱 AI 生成的长期训练计划（`schedule_json`）。由于 AI 生成内容的结构多变，使用 JSONB 存储能最大程度保留数据的灵活性。

5.3.3 动态组合场景

本小节描述在核心业务场景下，上述组件如何通过协作完成任务。为了验证"边缘推理"的低延迟特性，我们需要观察组件间的动态交互。下方的时序图刻画了在一次典型的深蹲动作中，前端各模块如何实现完全无需服务器参与的自闭环反馈。

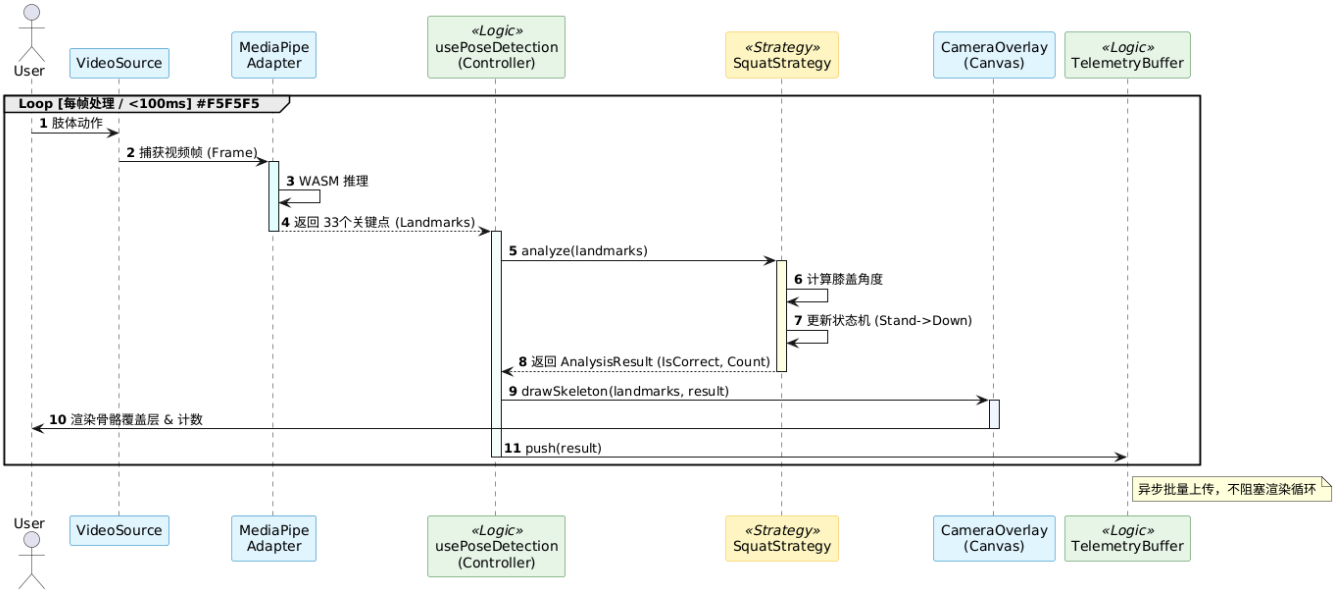


图5-6 实时分析闭环时序图

5.3.3.1 场景一：实时动作分析闭环

此场景展示了前端组件如何形成毫秒级响应闭环，无需后端介入。

1. **VideoSource** 捕获图像帧，传递给 **MediaPipeAdapter**。
2. **MediaPipeAdapter** 在 WASM 中执行推理，输出关键点。
3. **usePoseDetection** 接收关键点，注入当前选定的 **SquatStrategy**。
4. **SquatStrategy** 计算角度并更新计数，返回分析结果。
5. **CameraOverlay** 读取结果，立即绘制骨骼与计数覆盖层（<100ms）。

6. **Telemetry Buffer** 异步缓存该结果。

5.3.3.2 场景二：智能训练总结生成

此场景展示了前后端及外部服务如何跨边界协作。

1. **App** 触发“结束训练”，调用 **SessionController**。
2. **SessionService** 调用 **Advisor Agent**。
3. **Advisor Agent** 构建 Prompt，HTTPS 请求 **智谱 AI**。
4. **智谱 AI** 返回建议文本，**Advisor Agent** 解析并存入对象。
5. **SessionService** 调用 **SessionRepository**，将统计数据与 AI 建议一并写入数据库。

5.3.4 系统装配与连接

各子系统通过标准的**连接器**进行物理装配。

5.3.4.1 客户端-服务器装配

- **连接器**：HTTPS 连接器。
- **通信契约**：JSON over HTTP/1.1。
- **机制**：前端 Axios 客户端配置拦截器，自动注入 JWT Token，与后端中间件对接实现无状态鉴权。

5.3.4.2 服务器-数据装配

- **连接器**：数据库连接池 (SQLAlchemy)。
- **机制**：后端维护 TCP 长连接池连接 PostgreSQL，复用连接以降低开销。

5.3.4.3 服务器-外部服务装配

- **连接器**：API 网关客户端。
- **机制**：后端作为 HTTP 客户端，通过 TLS 加密通道调用智谱 AI 开放接口。

5.3.5 组合结构图

在理解了各个孤立的子系统后，我们需要从全局视角观察它们的**装配关系**。本图通过黑盒视角展示了前端、后端与外部 SaaS 服务之间的物理连接点与数据吞吐边界。

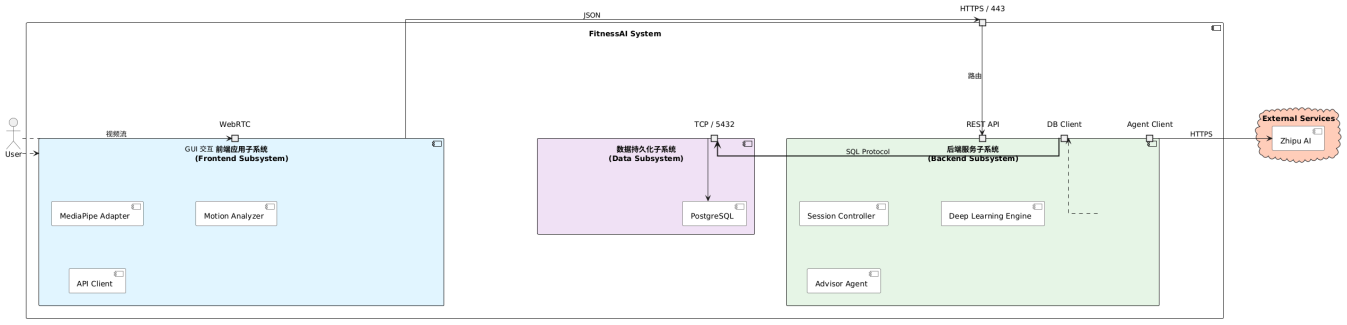


图5-7 系统组合结构图

5.3.6 设计基本原理

本组合视图的设计决策基于以下权衡：

1. **为什么将姿态推理放在前端？** 为了满足 **<100ms 实时反馈** 的非功能性需求。视频流网络传输延迟不可控，只有在边缘端（浏览器）处理才能消除网络瓶颈，同时规避上传用户原始视频的**隐私合规风险。**
2. **为什么后端还需要深度学习引擎？** 前端受限于浏览器算力，只能运行轻量级模型。后端引入 TensorFlow 引擎用于**异步复核**，处理前端难以判断的复杂边界情况，实现“速度”与“精度”的平衡。
3. **为什么使用 PostgreSQL JSONB 存储？** 运动数据具有高度**多态性。**深蹲的数据结构与平板支撑完全不同。JSONB 允许在同一张表中存储异构的遥测数据，避免了频繁的表结构变更。

5.3.7 部署单元

系统物理上映射为以下三个独立的可执行单元：

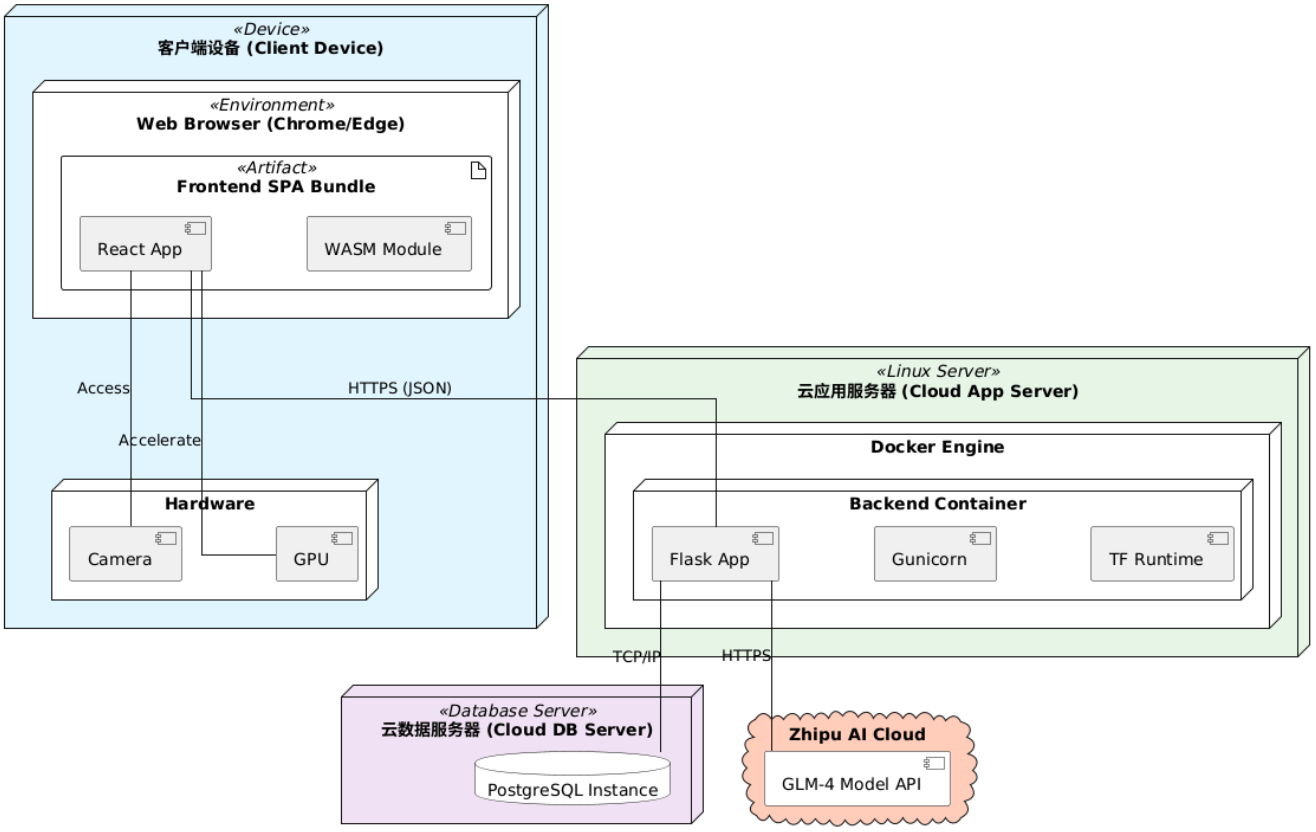


图5-8 部署单元架构图

1. **前端部署单元**：React 编译后的静态资源包 (HTML/CSS/JS/WASM)，部署至 Nginx 或 CDN。
2. **后端部署单元**：Docker 容器镜像（包含 Python 环境、Flask 应用、TensorFlow 运行时及模型文件），部署至云服务器。
3. **数据部署单元**：PostgreSQL 数据库实例。

5.4 逻辑视角

逻辑视角关注系统的静态结构与功能抽象。本节通过类图和包图，定义了 FitnessAI 系统的核心领域实体、接口契约及类之间的静态关系，旨在揭示系统如何通过面向对象设计原则（如多态、封装）来实现 SRS 中的业务需求。

5.4.1 领域对象模型

领域模型定义了系统内流转的核心业务实体及其关系。这些实体在数据库中持久化，并在前后端通过 JSON 序列化进行传输。

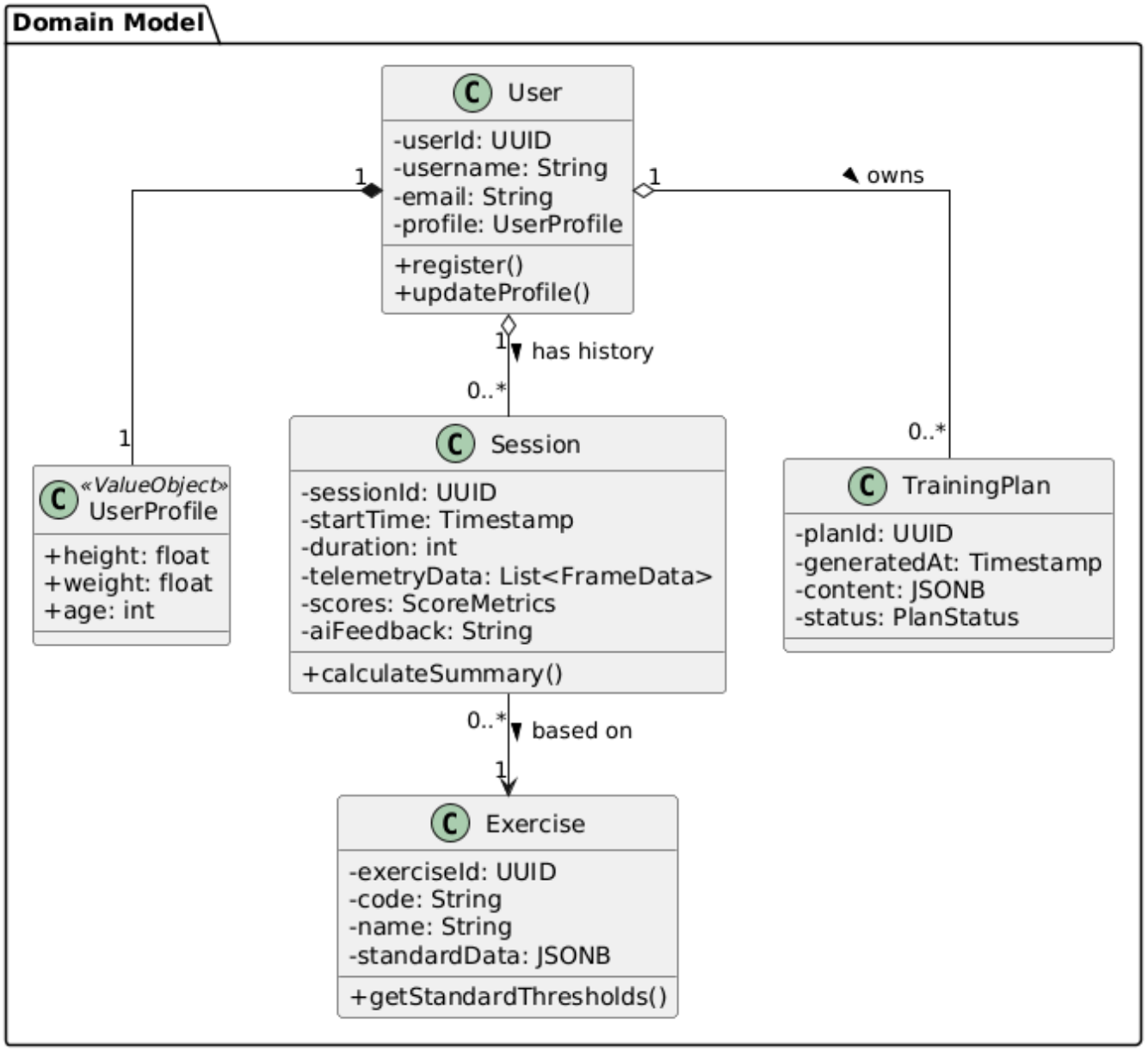


图5-9 核心领域类图

核心实体定义

1. **User (用户聚合根)**：系统的核心主体。不仅包含基础属性，还通过聚合关系管理其拥有的训练计划与历史会话。
2. **Session (训练会话)**：一次完整的运动记录。它是数据密度最高的实体，属性 `telemetry_data` 封装了时序性的骨骼坐标流。
3. **Exercise (运动元数据)**：定义了“深蹲”、“开合跳”等运动的标准参数（如 ID、名称、难度系数），属于系统的静态配置数据。
4. **TrainingPlan (训练计划)**：由 AI 生成的指导性实体，包含周期性的训练目标。

5.4.2 前端分析逻辑设计

前端不仅是 UI，还承载了核心的**实时分析逻辑。**为了支持多种运动类型（深蹲、平板支撑等）并遵循**开闭原则 (OCP)**，系统采用了**策略模式 (Strategy Pattern)。**

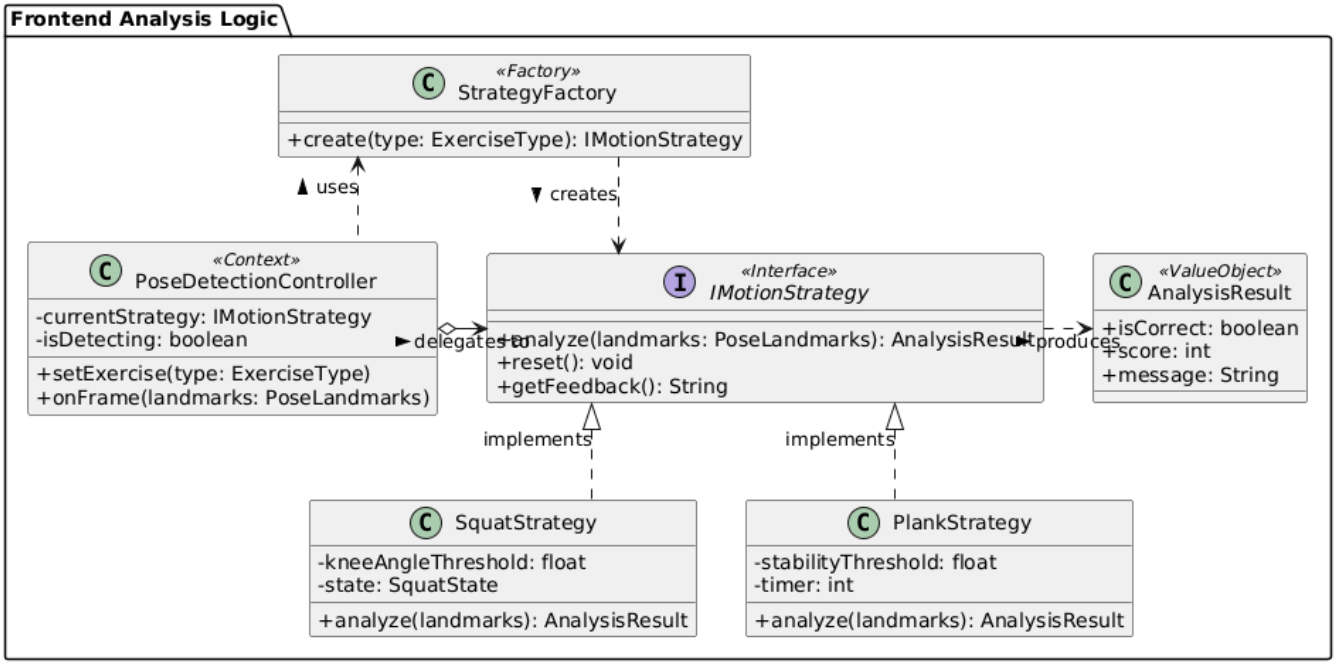


图5-10 前端分析策略模式类图

设计模式应用：策略模式

- **上下文：`PoseDetectionController`。它不关心具体的运动细节，只负责将 MediaPipe 的数据流转发给当前选定的策略。
- **抽象策略：`IMotionStrategy`。定义了统一契约方法 `analyze(frame: Keypoints) -> AnalysisResult`。
- **具体策略:
  - `SquatStrategy`: 实现了基于几何角度（髋膝角 < 90°）的深蹲计数与纠错算法。
  - `PlankStrategy`: 实现了基于身体直线度（头-髋-踝共线）的静态计时算法。
- **工厂 (Factory)**: `StrategyFactory`。根据用户在 UI 选择的运动类型，动态实例化对应的策略类。

5.4.3 后端服务分层设计

后端遵循**领域驱动设计 (DDD)** 的分层思想，将业务逻辑与基础设施解耦。

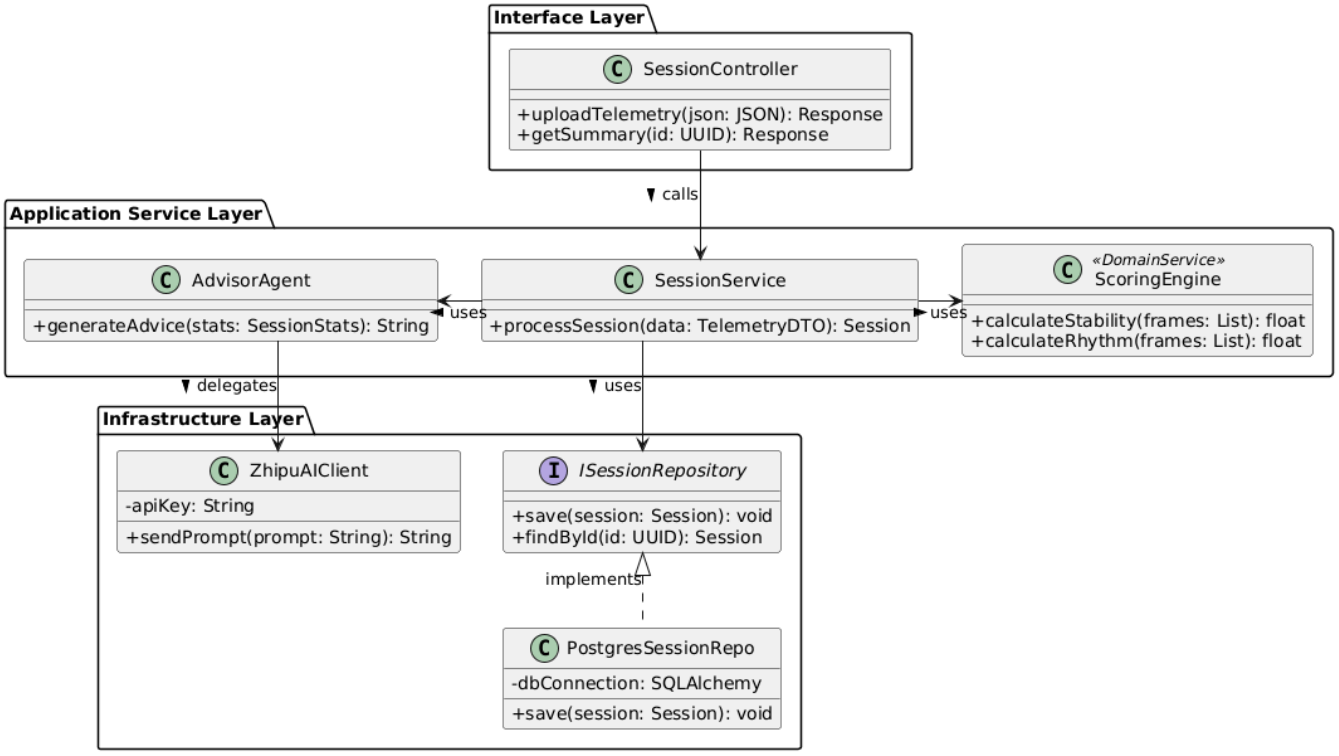


图5-11 后端服务分层类图

分层逻辑详解

1. **控制器层：`SessionController`。负责 HTTP 请求的解析与验证 (Pydantic Models)，不包含业务逻辑。
2. **应用服务层**：
   - `SessionService`：编排层。负责协调 `Repository` 进行存储，并调用 `AdvisorAgent` 获取 AI 建议。
   - `ScoringEngine`：领域服务。包含复杂的评分算法（如基于方差的动作稳定性计算）。
3. **基础设施层:
   - `SessionRepository`：实现数据库访问接口。
   - `ZhipuAIClient`：封装 HTTP 调用细节，处理重试与鉴权。

5.4.4 深度学习推理模块设计

本模块负责后端高精度动作分析的实现。它将非结构化的骨骼关键点转化为数学特征向量，并输入预训练的神经网络模型获取状态概率。

图5-12 深度学习模块类图

核心类定义

1. **SquatClassifier (推理包装器)**
    - **职责**：封装 TensorFlow/Keras 运行时环境。
    - **关键特性**：通常设计为**单例 (Singleton)**，以避免频繁加载大模型文件（`.h5`）导致的内存开销和延迟。它负责将模型的概率输出（如 `[0.1, 0.8, 0.1]`）转化为具体的枚举状态。
2. **FeatureExtractor (特征工程工具)**
    - **职责**：纯函数静态工具类。
    - **逻辑**：它不包含业务状态，仅负责执行向量数学运算。例如，计算 Hip-Knee-Ankle 的三维夹角、计算脊柱的垂直倾角等，最终输出一个 **24 维特征向量** 供模型消费。
3. **SquatState (状态枚举)**
    - 定义了模型的分类结果集：`STANDING` (站立), `DESCENDING` (下蹲过程), `BOTTOM` (底部), `ASCENDING` (起立过程)。

5.5 依赖视角

依赖视角描述了 FitnessAI 系统中各设计实体（子系统、模块、组件）之间的**使用（Uses）**、**包含（Includes）**和**部署（Deploys）**关系。本节旨在明确系统的耦合程度，识别关键路径上的外部依赖，并为后续的维护与变更影响分析提供依据。

5.5.1 模块间依赖原则

FitnessAI 遵循**“单向依赖”**与**“依赖倒置 (DIP)”**的设计原则，严格控制模块间的耦合方向。

5.5.1.1 前端子系统依赖链

前端采用 MVVM 变体架构，依赖流向严格从视图层指向逻辑层，再指向基础设施层。

- **View (UI) ViewModel (Hooks)**: `CameraView` 依赖 `usePoseDetection` 提供的数据流。UI 是易变的，但逻辑是相对稳定的。
- **ViewModel Domain (Strategy)**: `usePoseDetection` 依赖抽象接口 `IMotionStrategy`，而不直接依赖具体的 `SquatStrategy` 类（通过工厂模式解耦）。
- **Infrastructure External Libs**: `MediaPipeAdapter` 直接依赖 `@mediapipe/pose` 库。这是系统的**防腐层 (ACL)**，防止外部库的变更扩散到业务逻辑中。

5.5.1.2 后端子系统依赖链

后端采用分层架构，遵循自上而下的严格调用依赖。

- **Controller Layer Service Layer**: 接口层仅负责路由与参数解析，依赖业务层执行逻辑。
- **Service Layer Domain Layer**: 业务层依赖 `PoseAnalyzer` 和 `SquatClassifier` 等核心领域对象。
- **Service Layer Repository Layer (Interface)**: 业务层依赖数据访问接口（`ISessionRepository`），而非具体的 SQL 实现，确保了数据库技术的可替换性。

5.5.2 外部技术栈依赖

系统对第三方组件、库及运行时环境存在强依赖。为确保系统的可构建性（Buildability），以下表格列出了核心依赖及其版本约束。

5.5.2.1 前端运行时依赖

依赖组件	版本约束	关键性	变更风险	说明
MediaPipe Pose	`^0.10.0`	**Critical**	High	核心姿态识别库。API 变更将直接导致 `MediaPipeAdapter` 重构。
TensorFlow.js	`^4.10.0`	High	Medium	用于前端轻量级推理（如需）。
React	`^18.2.0`	High	Low	UI 框架。依赖其并发渲染特性（Concurrent Mode）。
Browser WebRTC	W3C Standard	**Critical**	Low	依赖 `navigator.mediaDevices`。需关注 Safari/Chrome 的实现差异。

5.5.2.2 后端运行时依赖

依赖组件	版本约束	关键性	变更风险	说明
TensorFlow (Py)	`2.14.x`	**Critical**	High	深度学习运行时。版本需与训练出的 `.h5` 模型文件兼容。
Flask	`^2.3.0`	High	Low	Web 框架。
SQLAlchemy	`^2.0.0`	High	Medium	ORM 框架。JSONB 字段的操作依赖其特定扩展。
Zhipu AI SDK	`^4.0.0`	Medium	Medium	外部大模型服务。依赖智谱 OpenAPI 规范。

5.5.3 变更影响分析

基于上述依赖关系，本节分析关键组件变更时的**波及范围 (Ripple Effect)**。

5.5.3.1 场景 A：MediaPipe 库升级

- **触发条件**：Google 发布 MediaPipe 新版本（如 v0.11），更改了关键点的数据结构（如新增了手指节点）。
- **直接影响**：前端 `MediaPipeAdapter` 必须修改以适配新接口。
- **间接影响**：
- `FeatureExtractor` 可能需要更新关键点索引映射。
- 后端 `SquatClassifier` 的输入维度若发生变化，需要重新训练模型。
- **控制策略**：依赖关系终止于 Adapter 层。业务逻辑层（Strategy）通过统一的数据接口与 Adapter 交互，因此

无需修改。

5.5.3.2 场景 B：增加新的运动类型（如“波比跳”）

- **触发条件**：产品需求新增运动支持。
- **影响范围**：
- **前端**：新增 `BurpeeStrategy` 类（实现 `IMotionStrategy` 接口）；`StrategyFactory` 增加分支。UI 层 `ExerciseSelector` 增加选项。
- **后端**：新增 `BurpeeAnalyzer`；数据库 `exercises` 表新增元数据。
- **结论**：由于采用了**策略模式**和**工厂模式**，变更仅涉及新增类，符合**开闭原则 (OCP)**，不会破坏现有代码稳定性。

5.5.3.3 场景 C：从 PostgreSQL 迁移至 MongoDB

- **触发条件**：非结构化数据量激增，需切换数据库。
- **影响范围**：
- 仅影响后端 **Repository Layer**（`PostgresSessionRepo`）的实现。
- Controller 层和 Service 层依赖的是 `ISessionRepository` 接口，因此**完全不受影响。**

5.5.4 依赖关系图

下图展示了 FitnessAI 系统中模块级和组件级的依赖层级。箭头方向表示 即“A 依赖 B”（A import B 或 A call B）。

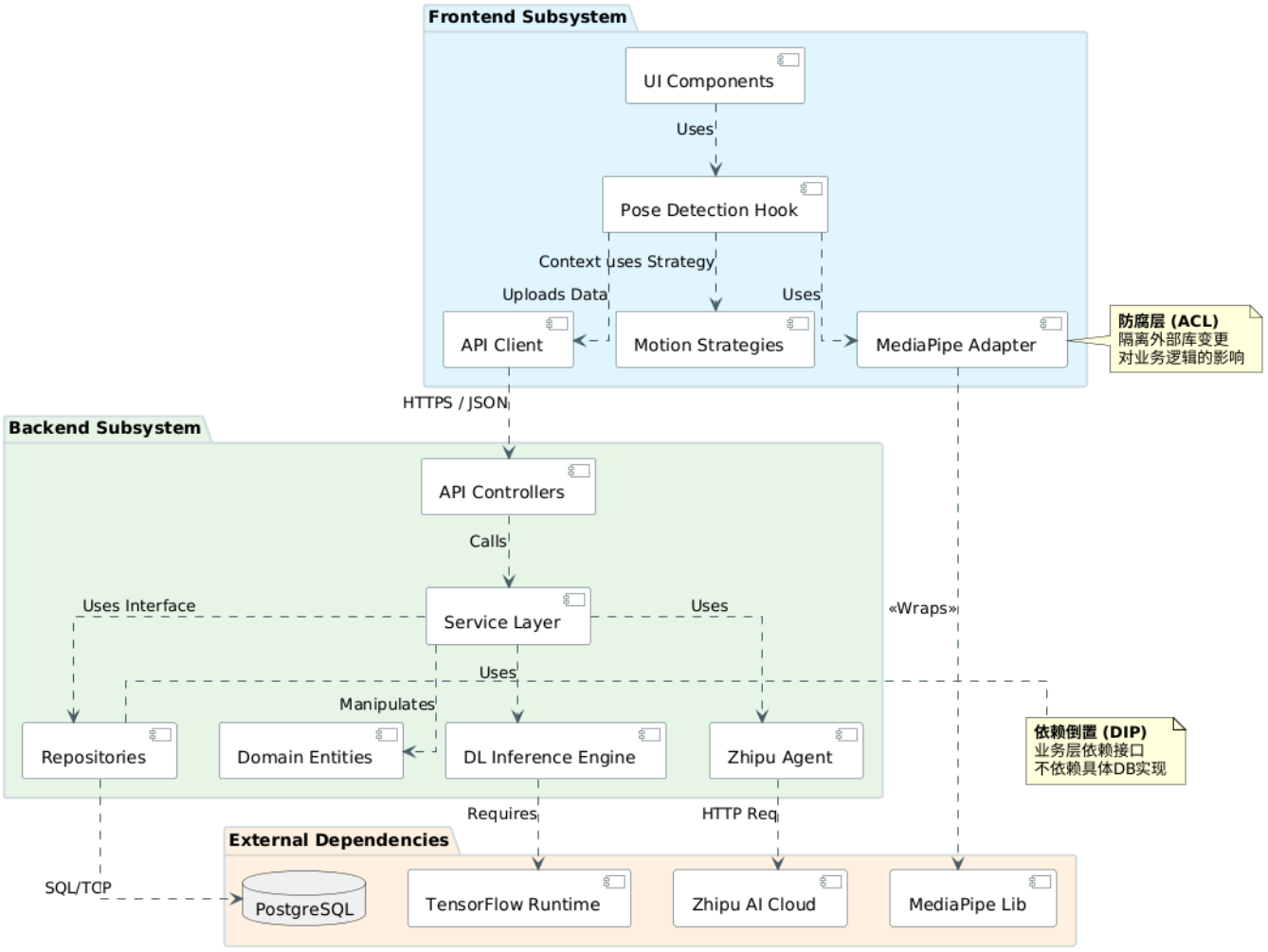


图5-13 依赖关系图

5.6 信息视角

信息视角关注系统持久化数据的组织方式、存储策略及生命周期管理。基于 **5.3 节** 定义的数据子系统架构，FitnessAI 采用 **PostgreSQL 对象-关系型数据库**，通过混合存储模式平衡业务事务的一致性需求与 AI 遥测数据的多态性需求。

5.6.1 数据持久化策略

系统依据数据的性质采用分级存储策略：

1. **强一致性数据**（如用户账号）：采用标准关系型表，严格遵循第三范式 (3NF)。
2. **高吞吐时序数据**（如 `telemetry_log`）：采用 **JSONB** 字段存储，利用 PostgreSQL 的 **GIN 索引** 支持对 JSON 内部字段的高效查询。
3. **非持久化数据**（如原始视频流）：**内存级处理，不落盘。**

5.6.2 数据模式规约

5.6.2.1 物理模型设计

下图是 5.3 ER 图的物理实现版。展示具体的字段类型（UUID, VARCHAR, TIMESTAMPTZ, JSONB）、主外键约束及索引设计。

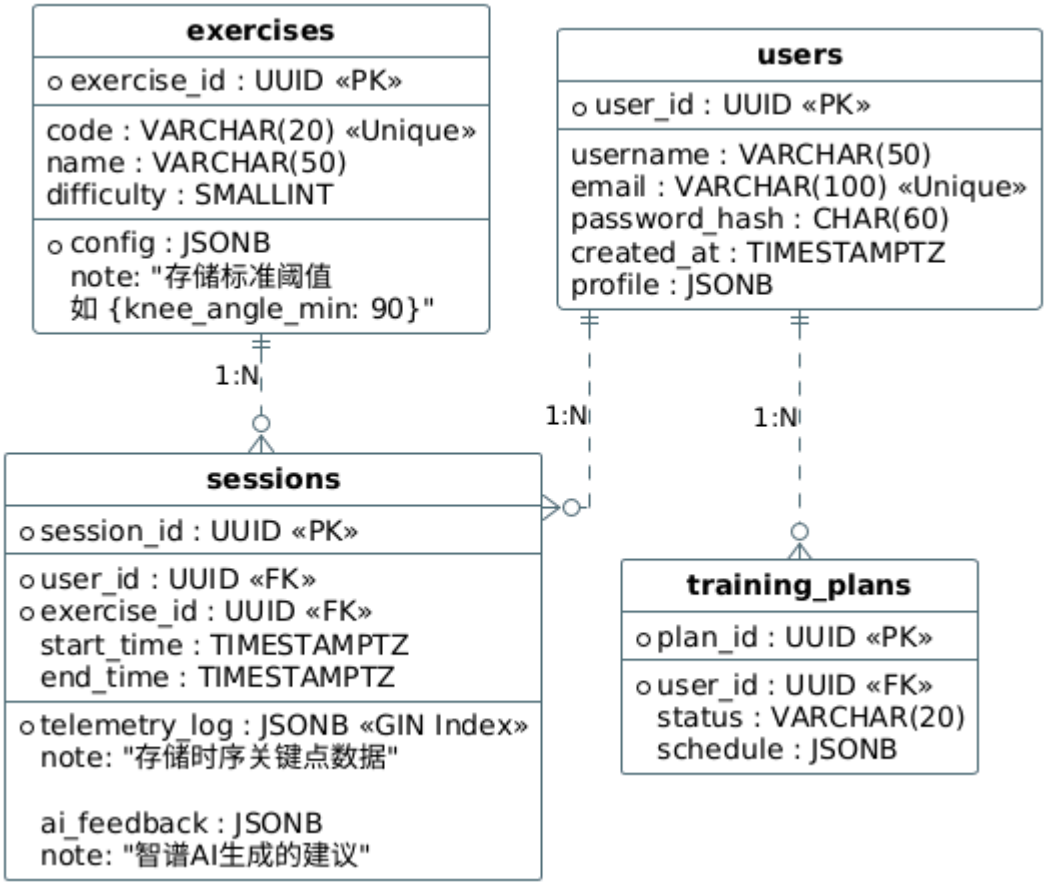


图5-14 数据库物理模型图

关键表定义：

表名 (Table)	核心字段	类型	约束/索引	说明
users	`user_id`	UUID	PK	用户唯一标识
	`email`	VARCHAR	Unique Index	登录账号
	`profile`	JSONB	-	包含身高(cm)、体重(kg)、性别
sessions	`session_id`	UUID	PK	训练会话 ID
	`telemetry_log`	JSONB	**GIN Index**	核心：动作遥测日志（详见 5.6.3）
	`ai_feedback`	JSONB	-	智谱 AI 生成的建议
exercises	`code`	VARCHAR	Unique	如 'squat', 'plank'
	`config`	JSONB	-	运动特定的阈值配置（如标准膝角）

5.6.2.2 JSONB 文档结构

由于 JSONB 缺乏数据库层面的 Schema 强制校验，本设计文档必须明确其**隐式 Schema**。

A. 遥测日志结构 (`sessions.telemetry_log`)

存储一次训练中所有关键帧的快照。系统不存储每一帧（30fps），仅存储**关键动作帧**（如深蹲到底部时）或按固定时间窗口降采样的数据。

JSON

```
{
  "version": "1.0",
  "sampling_rate": "500ms", // 采样频率
  "timeline": [
    {
      "timestamp": 12050,        // 相对开始时间的毫秒数
      "rep_count": 5,            // 当前完成次数
      "is_correct": false,       // 本次动作是否标准
      "faults": ["knee_valgus"],// 错误类型标签
      "metrics": {               // 运动特定的动态指标
        "knee_angle_left": 85.5,
        "hip_depth": 0.45
      },
      "landmarks_snapshot": [   // 33个关键点的归一化坐标 (压缩存储)
        [0.55, 0.65, 0.1],      // [x, y, z]
        ...
      ]
    }
  ]
}
```

B. AI 反馈结构 (`sessions.ai_feedback`)

存储智谱 AI 返回的结构化建议，用于前端展示富文本报告。

JSON

```
{
  "generated_at": "2023-12-20T10:00:00Z",
  "model_version": "glm-4",
  "summary_text": "今天的深蹲深度非常标准，但在后半程出现了轻微的膝盖内扣。",
  "improvement_plan": [
    {"target": "gluteus_medius", "action": "crab_walk", "reps": "3x15"}
  ],
  "score_breakdown": {
    "stability": 8.5,
    "rhythm": 9.0
  }
}
```

5.6.2.3 索引设计与查询优化 (Indexing & Optimization)

为确保在百万级数据量下的查询性能（响应时间 < 200ms），系统采用复合索引策略。

1. GIN 倒排索引

- **目标字段**：`sessions.telemetry_log`
- **策略**：创建 `jsonb_path_ops` 类型的 GIN 索引，支持对 JSON 内部任意键值的快速检索。
- **应用场景**：查询包含特定错误类型（如 "knee_valgus"）的所有会话。

2. 表达式索引

- **目标**：针对高频查询路径（Hot Paths）创建预计算索引，避免运行时解析 JSON。
- **示例 SQL**：

  SQL

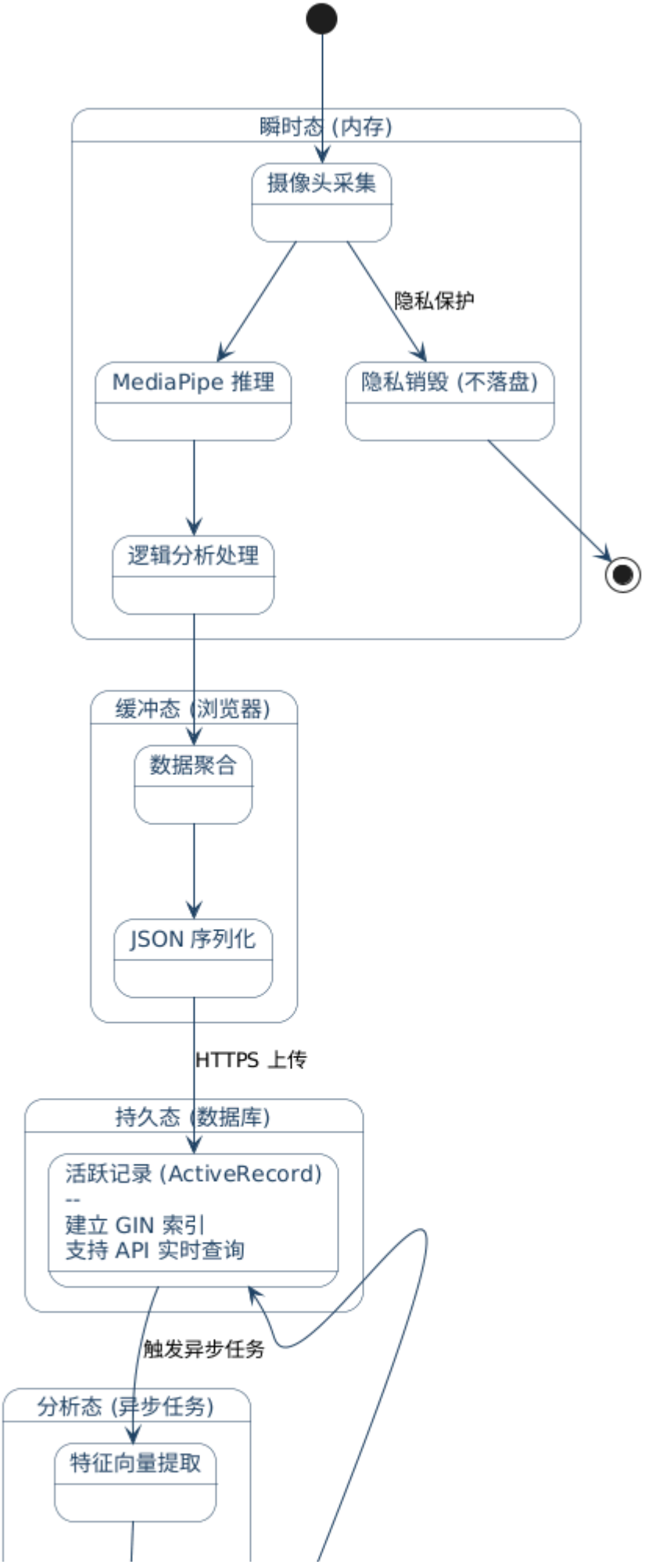
  ```
  -- 加速查询“完成次数”大于 10 的记录
  CREATE INDEX idx_sessions_rep_count ON sessions
  (((telemetry_log->'metrics'->>'total_reps')::int));
  ```

3. 常规 B-Tree 索引

- `users.email`：唯一索引，加速登录鉴权。
- `sessions.user_id` + `sessions.created_at`：复合索引，加速“用户历史记录”的分页查询。

5.6.3 数据生命周期与归档

数据流转遵循“采集-缓冲-活跃-归档”的生命周期。


瞬时态 (内存)
摄像头采集
隐私保护
MediaPipe 推理
隐私销毁 (不落盘)
逻辑分析处理
缓冲态 (浏览器)
数据聚合
JSON 序列化
HTTPS 上传
持久态 (数据库)
活跃记录 (ActiveRecord)
--
建立 GIN 索引
支持 API 实时查询
触发异步任务
分析态 (异步任务)
特征向量提取

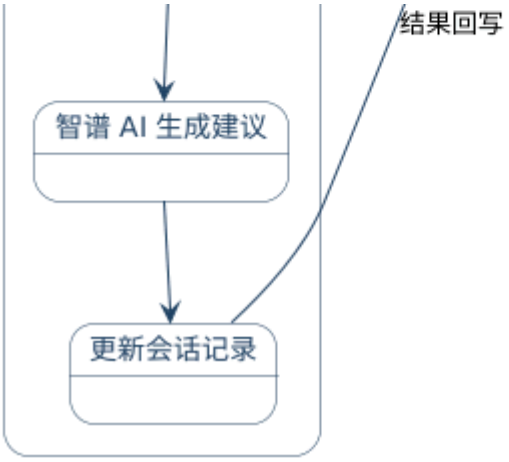


图5-15 数据生命周期状态图

分级存储与归档策略:

- **热数据 (Hot)**：最近 1 年的数据存储在主表 `sessions` 中，提供毫秒级读写。
- **冷数据 (Cold)**：超过 1 年的数据自动迁移至 `sessions_archive` 分区表（Partition Table）。
- **空间优化**：归档时执行 **ETL 清洗**，保留统计指标（Metrics），但移除占用空间巨大的 `landmarks_snapshot`（骨骼关键点快照），预计节省 80% 存储空间。

5.6.4 数据迁移与版本管理 (Migration & Versioning)

鉴于 JSONB 数据的灵活性，系统采用 **Schema-on-Read** 策略配合工具管理变更。

**1. Schema 演进策略

- **向后兼容**：新增字段时，旧数据视为 `null`，应用层需做好空值处理。
- **软废弃**：字段废弃时，保留历史数据但标记为 `deprecated`，不执行物理删除以保障历史回溯能力。

2. 迁移工具链

- **工具**：使用 **Alembic** (Python) 管理数据库 Schema 版本。
- **回滚机制**：所有迁移脚本（Migration Scripts）必须包含对应的 `upgrade()` 和 `downgrade()` 方法，确保发布失败时可无损回滚。

5.6.5 数据安全与隐私保护

系统严格遵循"隐私设计 (Privacy by Design)"原则，确保符合 SRS 中的合规性要求。

1. 字段级加密

- **敏感信息**：`users.profile` 中的身高、体重等生物指标，在应用层使用 AES-256 加密后存入数据库，密钥由密钥管理服务（KMS）独立保管。

2. 最小化脱敏 (Anonymization)

- **AI 交互**：后端 Advisor Agent 在调用智谱 AI 接口前，执行**动态脱敏**，移除所有 PII（个人身份信息），仅发送统计类数据（如"膝盖角度偏差值"），确保第三方无法反向追踪用户。

3. 数据保留策略 (Retention)

- **主动注销**：用户注销账号后，系统启动 **30 天冷冻期**，期满后物理删除所有关联的 `users` 及 `sessions` 记录（硬删除）。

5.7 设计模式使用视角

本视角描述了 FitnessAI 系统在解决特定设计问题时所采用的标准设计模式。系统遵循“面向接口编程”与“组合优于继承”的原则，广泛应用了 **GoF 设计模式**，以提高代码的可维护性、可扩展性及对第三方库的解耦能力。

5.7.1 模式应用汇总

下表总结了系统中的核心设计模式及其在 FitnessAI 中的具体落地场景：

模式类别	模式名称	应用组件 (Context)	解决的问题
行为型	**策略模式 (Strategy)**	前端 `MotionAnalyzer`	**问题**：深蹲和开合跳的计算逻辑完全不同，`if-else` 难以维护。 **方案**：定义 `IMotionStrategy` 接口，动态切换算法实现。
行为型	**观察者模式 (Observer)**	前端 `StatsDashboard`	**问题**：高频（30fps）分析结果需要驱动 UI 刷新，但逻辑层不应持有 UI 引用。 **方案**：UI 组件订阅数据流，实现松耦合的实时渲染。
结构型	**适配器模式 (Adapter)**	前端 `MediaPipeAdapter`	**问题**：`@mediapipe/pose` 库接口复杂且易变。 **方案**：建立防腐层（ACL），统一转换为系统内部的标准数据结构。
创建型	**工厂方法 (Factory Method)**	前端 `StrategyFactory`	**问题**：创建具体的策略对象需要复杂的初始化参数。 **方案**：封装实例化逻辑，只需传入 `exerciseType` 即可获取对应策略。
创建型	**单例模式 (Singleton)**	后端 `SquatClassifier`	**问题**：TensorFlow 模型文件大，加载慢。 **方案**：确保全局仅有一个模型实例，避免内存溢出。

模式类别	模式名称	应用组件 (Context)	解决的问题
结构型	**外观模式 (Facade)**	后端 `ZhipuAIClient`	**问题**：调用智谱 AI 需要处理 Token、Prompt 组装、重试等繁琐细节。 **方案**：提供简单的 `generate_advice()` 接口供业务层调用。
架构模式	**仓储模式 (Repository)**	后端 `SessionRepository`	**问题**：JSONB 索引查询语法复杂，污染业务代码。 **方案**：在仓储层封装 SQL 细节，向 Service 层暴露语义化接口。

5.7.2 核心模式详解

5.7.2.1 策略与工厂模式 (Strategy & Factory)

这是系统前端**实时分析引擎**的核心设计。通过策略模式实现算法的**可替换性**，通过工厂模式实现对象的**可创建性**。

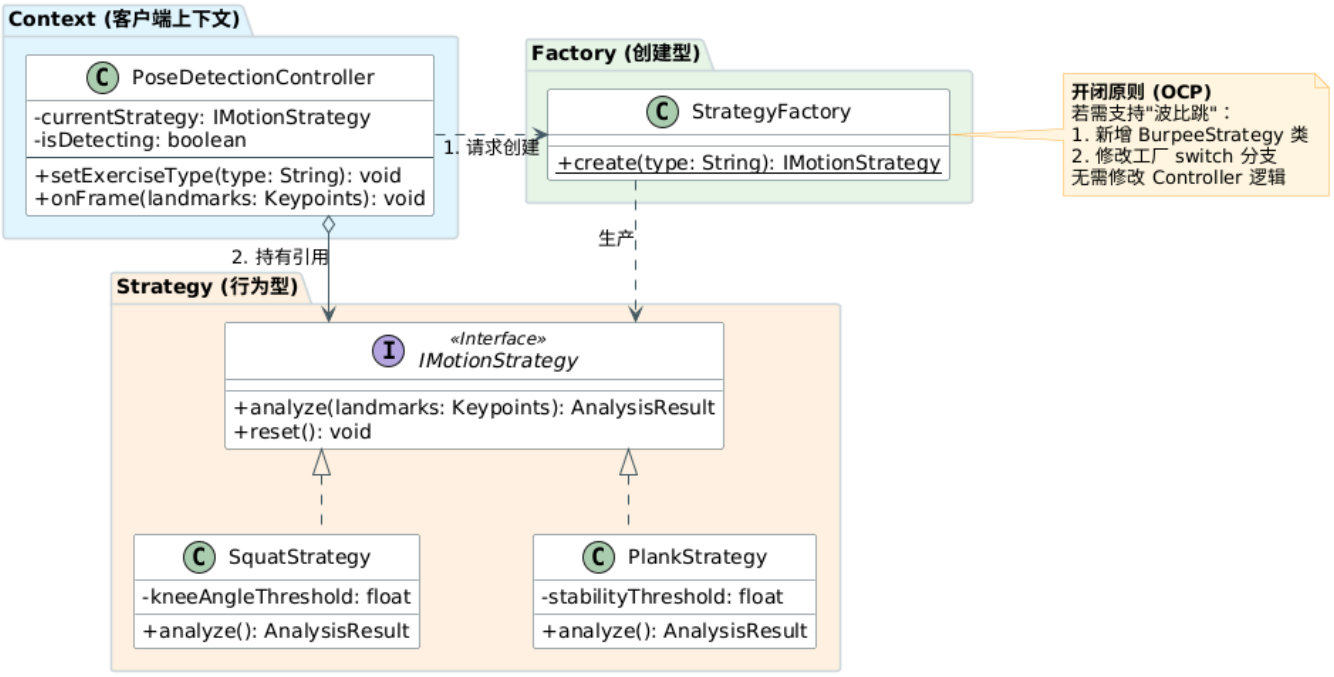


图5-16 运动分析策略模式类图

5.7.2.2 适配器模式 (Adapter)

作为系统的**防腐层 (Anti-Corruption Layer)**，`MediaPipeAdapter` 隔离了外部算法库的变化。

- **Target**: `detect(image): UnifiedLandmarks`
- **Adaptee**: `MediaPipe.Pose.send()` / `onResults()`
- **价值**：即使未来更换为 YOLOv8-Pose，只需重写适配器，上层业务逻辑（策略层）无需任何修改。

5.7.3 设计权衡 (Trade-offs)

在使用模式时，系统进行了以下权衡：

1. **避免过度工程**：对于简单的 CRUD 业务（如用户资料修改），系统直接在 Service 层实现，未引入命令模式（Command Pattern），保持了代码的简洁性。
2. **性能考量**：在前端高频（30fps）渲染循环中，避免频繁创建销毁对象。策略对象在切换运动时**惰性加载 (Lazy Loading)** 并复用，以减少垃圾回收（GC）压力。

5.8 接口视角 (Interface Viewpoint)

接口视角详细描述了 FitnessAI 系统中各组件之间、系统与外部环境之间的接口契约。本视角关注接口的语法（签名、参数）、语义（行为契约）及非功能属性（性能、可靠性），为系统集成与测试提供精确的接口规约。

5.8.1 接口分类与层次

FitnessAI 系统的接口按层次划分为三类：

5.8.1.1 外部接口（External Interfaces）

系统与外部实体（用户、浏览器运行时、第三方服务）的交互边界。

用户界面接口：

- **GUI 交互契约**：基于 React 的事件驱动模型，用户通过点击、拖拽等手势触发状态变更
- **视觉反馈契约**：Canvas 2D 渲染接口，以 30fps 频率绘制骨骼覆盖层

浏览器运行时接口：

- **MediaDevices API**：`getUserMedia(constraints)` 获取摄像头流，返回 `MediaStream` 对象
- **WebRTC 接口**：视频流捕获与处理
- **Fetch API**：HTTP 请求接口，用于前后端通信

第三方服务接口：

- **智谱 AI OpenAPI**：RESTful HTTP 接口，遵循 OpenAPI 3.0 规范

5.8.1.2 内部组件接口（Component Interfaces）

系统内部组件间的接口契约，包括前端模块间、后端服务层间及前后端之间的接口。

前端组件接口：

- **usePoseDetection Hook 接口**：暴露 `{ videoRef, canvasRef, isActive, exerciseStats,`

`startDetection, stopDetection, resetStats }`

- **SquatClassifier 接口**：深度学习推理接口 `predict(landmarks: Landmark[]): SquatState`，返回状态分类及置信度

后端服务接口：

- **RESTful API 接口**：遵循 REST 风格，JSON 序列化
- **Repository 接口**：数据访问抽象层，定义 `save()`, `findById()`, `findByUserId()` 等方法

5.8.1.3 数据接口（Data Interfaces）

跨越系统边界的数据结构定义，包括请求/响应载荷、数据库 Schema。

API 数据契约：

- **请求载荷**：JSON 格式，包含 `exercise_type`, `pose_data`, `user_id` 等字段
- **响应载荷**：JSON 格式，包含 `session_id`, `analysis_result`, `feedback` 等字段

5.8.2 RESTful API 接口规约

后端暴露的 RESTful API 接口遵循统一的设计原则：资源导向、无状态、幂等性。

5.8.2.1 认证接口组

接口标识：`/api/auth/*`

接口路径	HTTP 方法	功能	请求体	响应体	认证要求
`/api/auth/register`	POST	用户注册	`{ username, password, email?, nickname? }`	`{ token, user }`	无
`/api/auth/login`	POST	用户登录	`{ username, password }`	`{ token, user }`	无
`/api/auth/me`	GET	获取当前用户信息	无	`{ user_id, username, nickname, email, profile }`	Bearer Token
`/api/auth/change-password`	POST	修改密码	`{ old_password, new_password }`	`{ message }`	Bearer Token

接口契约细节：

- **认证机制**：JWT Token，通过 `Authorization: Bearer {token}` Header 传递
- **Token 有效期**：24 小时
- **密码安全**：SHA-256 哈希存储，最小长度 6 位

5.8.2.2 会话管理接口组

接口标识：`/api/session/*`

接口路径	HTTP 方法	功能	请求体	响应体	认证要求
`/api/session/start`	POST	开始训练会话	`{ exercise_type, user_id? }`	`{ session_id, message }`	可选
`/api/session/{session_id}/data`	POST	提交运动数据	`{ pose_data, is_correct, score, feedback }`	`{ message, session_stats }`	无
`/api/session/{session_id}/end`	POST	结束训练会话	无	`{ session_id, summary }`	无

接口契约细节：

- **会话 ID 格式**：`{user_id}_{YYYYMMDD_HHMMSS}`
- **数据提交频率**：前端 Telemetry Buffer 批量提交，避免高频请求
- **会话状态**：`active` → `completed`

5.8.2.3 AI 服务接口组

接口标识：`/api/ai/*`

接口路径	HTTP 方法	功能	请求体	响应体	认证要求
`/api/ai/generate-plan`	POST	AI 生成健身计划	`{ height?, weight?, age?, gender? }`	`{ daily_goals, weekly_goals, suggestions, bmi, fitness_level }`	Bearer Token

接口契约细节：

- **降级机制**：API Key 未配置或调用失败时，自动降级使用规则引擎
- **响应时间**：AI 调用超时 30 秒，重试 2 次
- **数据脱敏**：发送给 AI 的数据不包含用户 PII

5.8.3 前端组件接口规约

5.8.3.1 usePoseDetection Hook 接口

接口签名：

```
interface UsePoseDetectionReturn {
  videoRef: RefObject<HTMLVideoElement>;
  canvasRef: RefObject<HTMLCanvasElement>;
  isActive: boolean;
  poseResults: PoseResults | null;
  exerciseStats: ExerciseStats;
  startDetection: () => Promise<void>;
  stopDetection: () => void;
  resetStats: () => void;
}
```

接口行为契约：

- **startDetection()**：异步初始化 MediaPipe，启动摄像头流，设置 `isActive = true`
- **stopDetection()**：停止摄像头流，释放资源，设置 `isActive = false`
- **resetStats()**：重置分析器状态，清零计数与得分

5.8.3.2 SquatClassifier 接口

接口签名：

```
interface SquatClassifier {
  predict(landmarks: Landmark[]): SquatState;
  extractFeatures(landmarks: Landmark[]): number[];  // 24维特征向量
  reset(): void;
}

interface SquatState {
  state: 0 | 1 | 2;         // 0:站直, 1:半蹲, 2:完全蹲下
  confidence: number;        // 0-1，模型置信度
  probabilities: number[];  // [p0, p1, p2] 三种状态的概率
}
```

接口行为契约：

- **predict()**：执行深度学习模型推理，返回状态分类及置信度
- **extractFeatures()**：从关键点提取24维特征向量
- **reset()**：重置内部状态（EMA历史、状态计数器等）

5.8.4 接口图

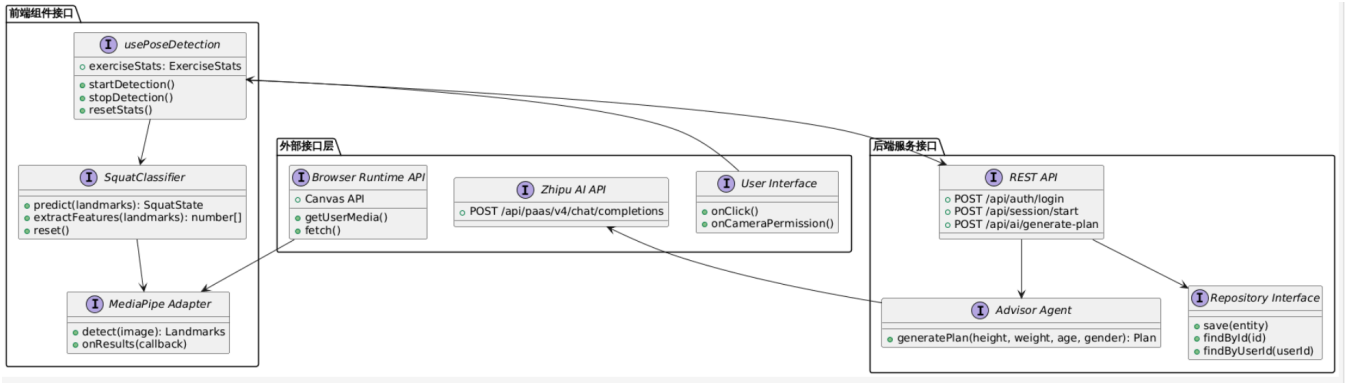


图5-17 系统接口层次图

5.8.5 接口非功能属性

5.8.5.1 性能约束

- **前端实时分析接口**：`analyze()` 方法执行时间 < 16ms（满足 60fps）
- **REST API 响应时间**：P95 < 200ms，P99 < 500ms
- **AI 接口超时**：30 秒，重试 2 次

5.8.5.2 可靠性约束

- **接口可用性**：REST API 可用性 ≥ 99.5%
- **错误处理**：所有接口返回标准错误格式 `{ error: string, message?: string }`
- **降级策略**：AI 接口失败时自动降级使用规则引擎

5.8.5.3 安全性约束

- **认证接口**：密码传输使用 HTTPS，存储使用 SHA-256 哈希
- **数据脱敏**：AI 接口调用前移除用户 PII
- **CORS 策略**：前端仅允许从 `http://localhost:3000` 访问后端

5.9 结构视角 (Structure Viewpoint)

结构视角描述系统的静态组织结构，包括模块的层次结构、组件间的组合关系及数据结构的组织方式。本视角通过包图、组件图及数据结构图，揭示系统的静态架构骨架。

5.9.1 系统层次结构

FitnessAI 系统采用**分层架构**，严格遵循**关注点分离**原则。系统从下至上分为四层：

5.9.1.1 表示层（Presentation Layer）

职责：用户界面渲染与交互事件处理。

组件：

- **App Container**：React 应用根容器，管理全局状态（AuthContext）
- **CameraView**：摄像头视图组件，集成视频流与 Canvas 覆盖层
- **StatsPanel**：实时统计面板，显示计数、得分、反馈
- **ExerciseSelector**：运动类型选择器
- **Profile**：用户资料管理界面

设计模式：组件化、状态提升（Lifting State Up）

5.9.1.2 业务逻辑层（Business Logic Layer）

职责：核心业务规则实现，不依赖 UI 或数据存储细节。

前端业务逻辑：

- **usePoseDetection Hook**：姿态检测生命周期管理
- **SquatClassifier**：深度学习模型推理引擎（特征提取、模型预测、结果平滑）
- **Telemetry Buffer**：数据缓冲与批量提交

后端业务逻辑：

- **SessionService**：会话业务编排
- **PoseAnalysisEngine**：姿态分析引擎
- **AdvisorAgent**：AI 代理服务

设计模式：策略模式、工厂模式、服务层模式

5.9.1.3 数据访问层（Data Access Layer）

职责：封装数据持久化逻辑，提供领域对象与数据存储的映射。

组件：

- **SessionRepository**：会话数据访问
- **UserRepository**：用户数据访问
- **PlanRepository**：健身计划数据访问

设计模式：仓储模式（Repository Pattern）

5.9.1.4 基础设施层（Infrastructure Layer）

职责：提供技术能力支撑，包括外部库适配、网络通信、数据存储。

组件：

- **MediaPipeAdapter**：MediaPipe 库适配器
- **ApiClient**：HTTP 客户端封装（Axios/Fetch）

- **DatabaseConnection**：PostgreSQL 连接池管理

设计模式：适配器模式、外观模式

5.9.2 前端模块结构

前端采用 **React + TypeScript** 架构，按功能域划分为以下模块：

5.9.2.1 核心模块（Core Modules）

```
frontend/src/
├── components/          # UI 组件
|   ├── CameraView.tsx
|   ├── ExerciseSelector.tsx
|   ├── StatsPanel.tsx
|   └── Profile.tsx
├── hooks/               # React Hooks
|   └── usePoseDetection.ts
├── utils/               # 工具函数
|   ├── featureExtractor.ts  # 特征提取算法
|   └── squatClassifier.ts   # 深度学习分类器
├── services/            # API 服务
|   └── api.ts
└── contexts/            # Context 提供者
    └── AuthContext.tsx
```

模块依赖关系：

- `components` → `hooks` → `utils` → `services`
- `components` → `contexts`（全局状态注入）

5.9.2.2 数据流结构

前端采用**单向数据流**（Unidirectional Data Flow）：

1. **用户交互** → 触发事件处理器
2. **事件处理器** → 调用 Hook 方法
3. **Hook** → 更新本地状态
4. **状态变更** → 触发组件重渲染
5. **组件** → 展示最新 UI

5.9.3 后端模块结构

后端采用 **Flask + Python** 架构，按分层原则组织：

5.9.3.1 模块划分

```
backend/
├── app.py              # Flask 应用入口，路由定义
├── pose_analyzer.py    # 姿态分析引擎
├── repositories/       # 数据访问层（未来扩展）
├── services/           # 业务服务层（未来扩展）
└── models/             # 领域模型（未来扩展）
```

当前实现：采用**扁平化结构**，所有逻辑集中在 `app.py`，符合小型项目快速迭代需求。未来可重构为分层结构。

5.9.4 数据结构组织

5.9.4.1 领域对象结构

User（用户聚合根）：

```typescript
interface User {
  user_id: string;
  username: string;
  password_hash: string;
  email: string;
  nickname: string;
  profile: {
    height: number;    // cm
    weight: number;    // kg
    age: number;
    gender: 'male' | 'female' | 'other';
  };
  created_at: string;  // ISO 8601
}
```

Session（训练会话）：

```typescript
interface Session {
  session_id: string;
  user_id: string;
  exercise_type: string;
  start_time: string;
  end_time: string | null;
  total_count: number;
  correct_count: number;
  scores: Array<{
    timestamp: string;
    score: number;
    is_correct: boolean;
    feedback: string;
    pose_data?: Landmark[];
  }>;
  status: 'active' | 'completed';
}
```

ExerciseAnalysis（分析结果）：

```
interface ExerciseAnalysis {
  isCorrect: boolean;
  score: number;        // 0-100
  feedback: string;
  count?: number;
  duration?: number;    // 秒
}
```

5.9.5 结构图

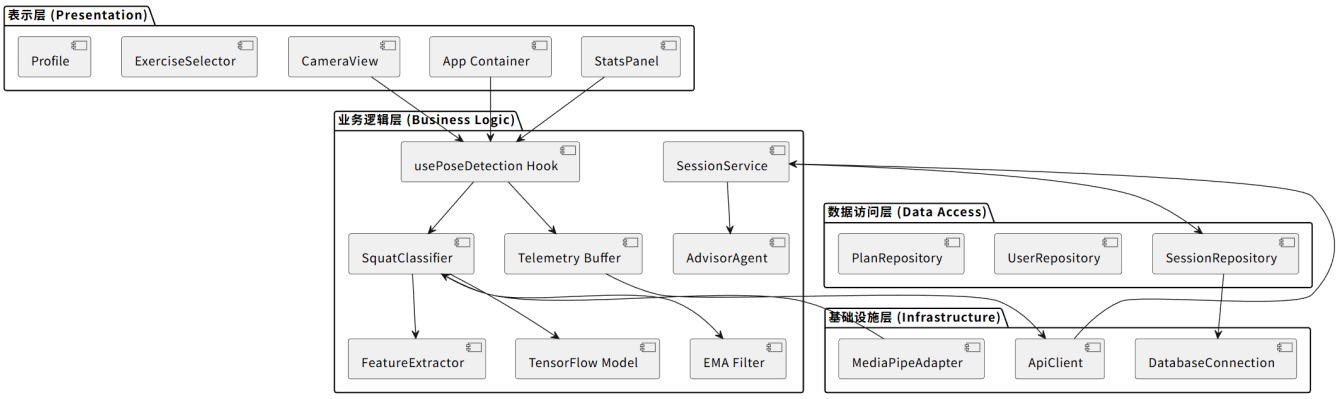


图5-18 系统分层结构图

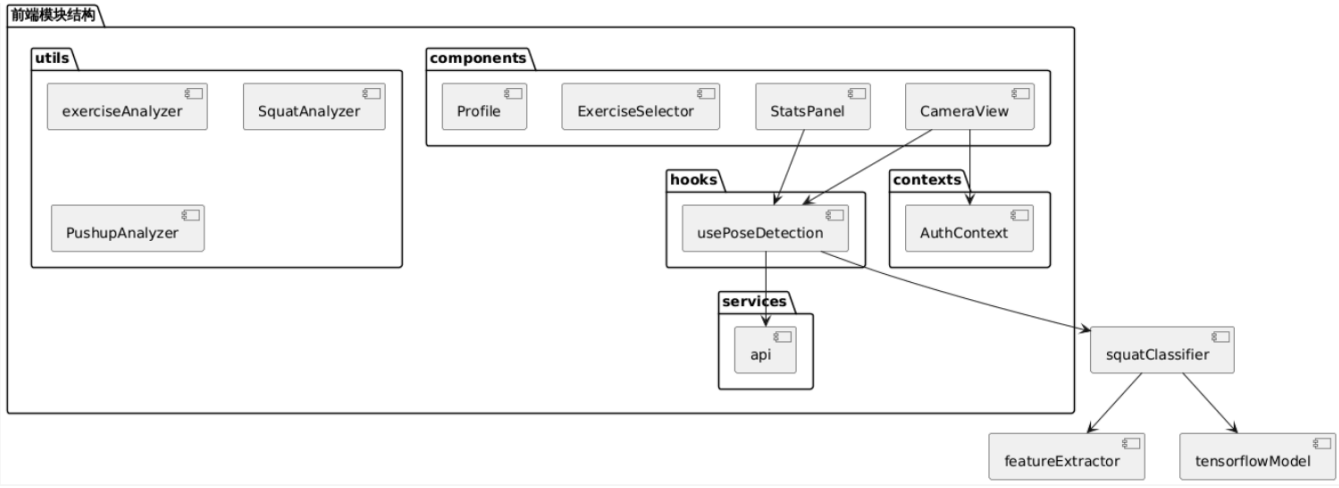


图5-19 前端模块结构图

5.10 交互视角 (Interaction Viewpoint)

交互视角描述系统组件之间的动态协作关系，通过时序图、协作图展示对象/组件在特定场景下的消息传递与调用序列。本视角重点关注关键业务流程中的组件交互模式。

5.10.1 核心交互场景

5.10.1.1 场景一：实时姿态检测闭环

场景描述：用户启动摄像头，系统实时检测姿态并给出反馈，整个过程无需后端参与。

参与者：

- User（用户）
- CameraView（视图组件）
- usePoseDetection（Hook）
- MediaPipeAdapter（适配器）
- SquatClassifier（深度学习分类器）
- Canvas（渲染层）

交互序列：

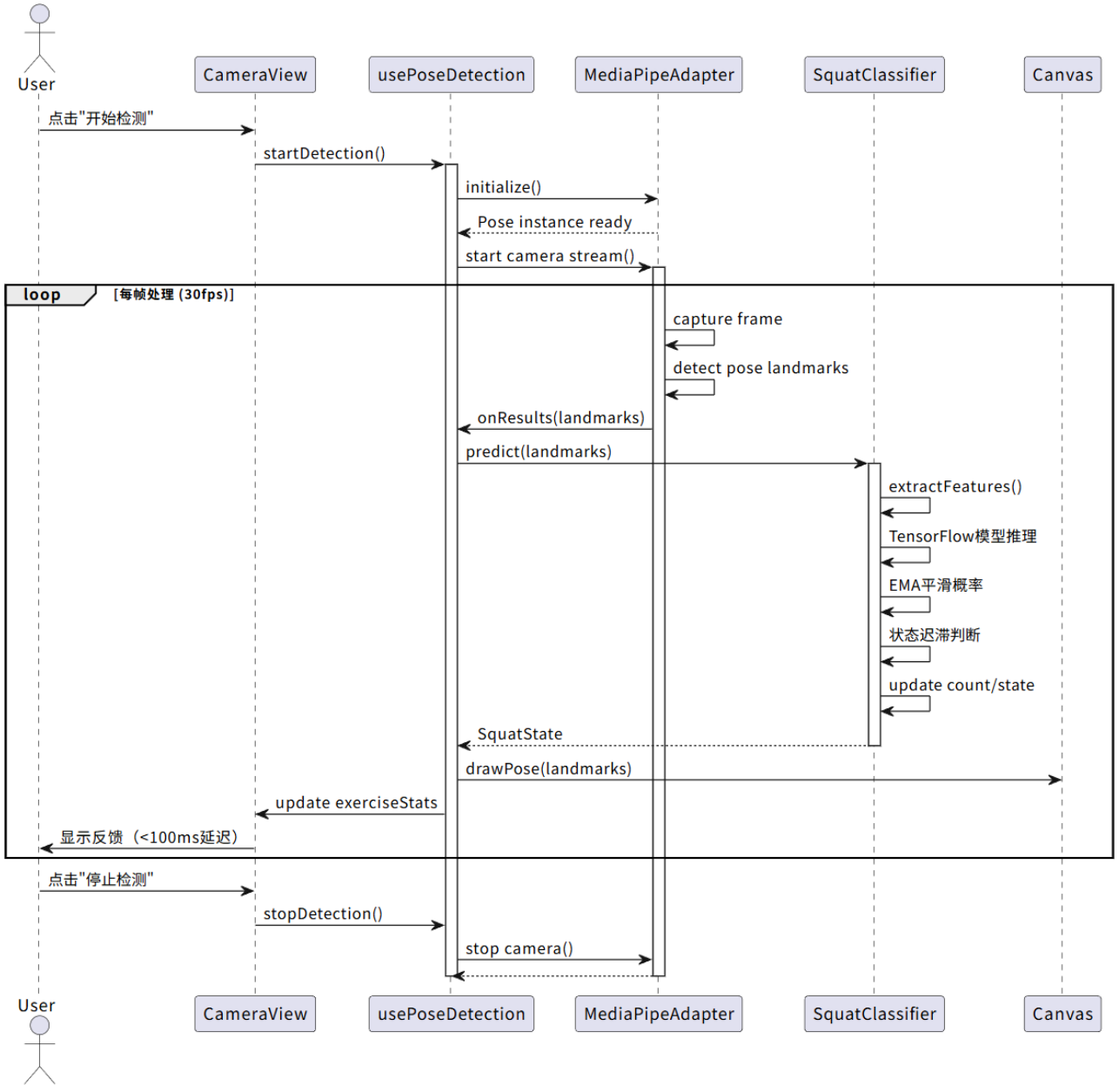


图5-20 实时姿态检测交互时序图

关键交互点：

1. **初始化阶段**：Hook 创建 MediaPipe 实例，配置检测参数
2. **实时处理循环**：30fps 帧率，每帧执行检测→分析→渲染→反馈
3. **低延迟保证**：整个闭环在浏览器内完成，延迟 < 100ms

5.10.1.2 场景二：训练会话生命周期

场景描述：用户开始训练，系统创建会话，持续提交数据，最后结束会话并生成报告。

参与者：

- Frontend（前端）
- SessionController（控制器）
- SessionService（服务层）
- SessionRepository（仓储层）
- Database（数据库）
- AdvisorAgent（AI 代理）

交互序列：

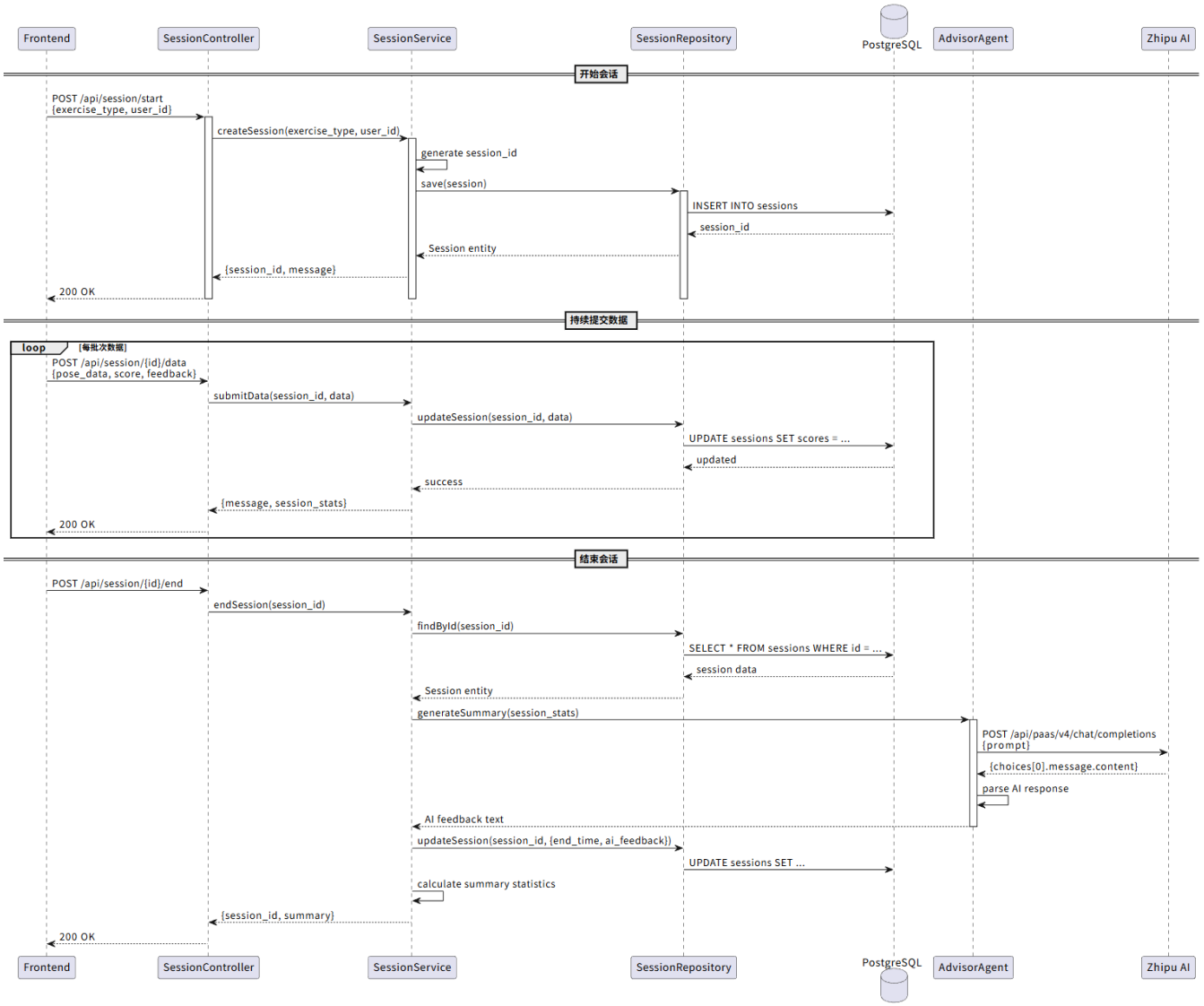


图5-21 训练会话生命周期交互时序图

关键交互点：

1. **会话创建**：前端发起请求，后端生成唯一 session_id
2. **数据提交**：Telemetry Buffer 批量提交，减少 HTTP 请求频率
3. **会话结束**：触发 AI 总结生成，异步调用外部服务

5.10.1.3 场景三：AI 健身计划生成

场景描述：用户请求 AI 生成个性化健身计划，系统调用智谱 AI API，解析返回结果。

参与者：

- Frontend（前端）
- AIController（AI 控制器）
- AdvisorAgent（AI 代理）
- ZhipuAI（外部服务）
- PlanRepository（计划仓储）

交互序列：

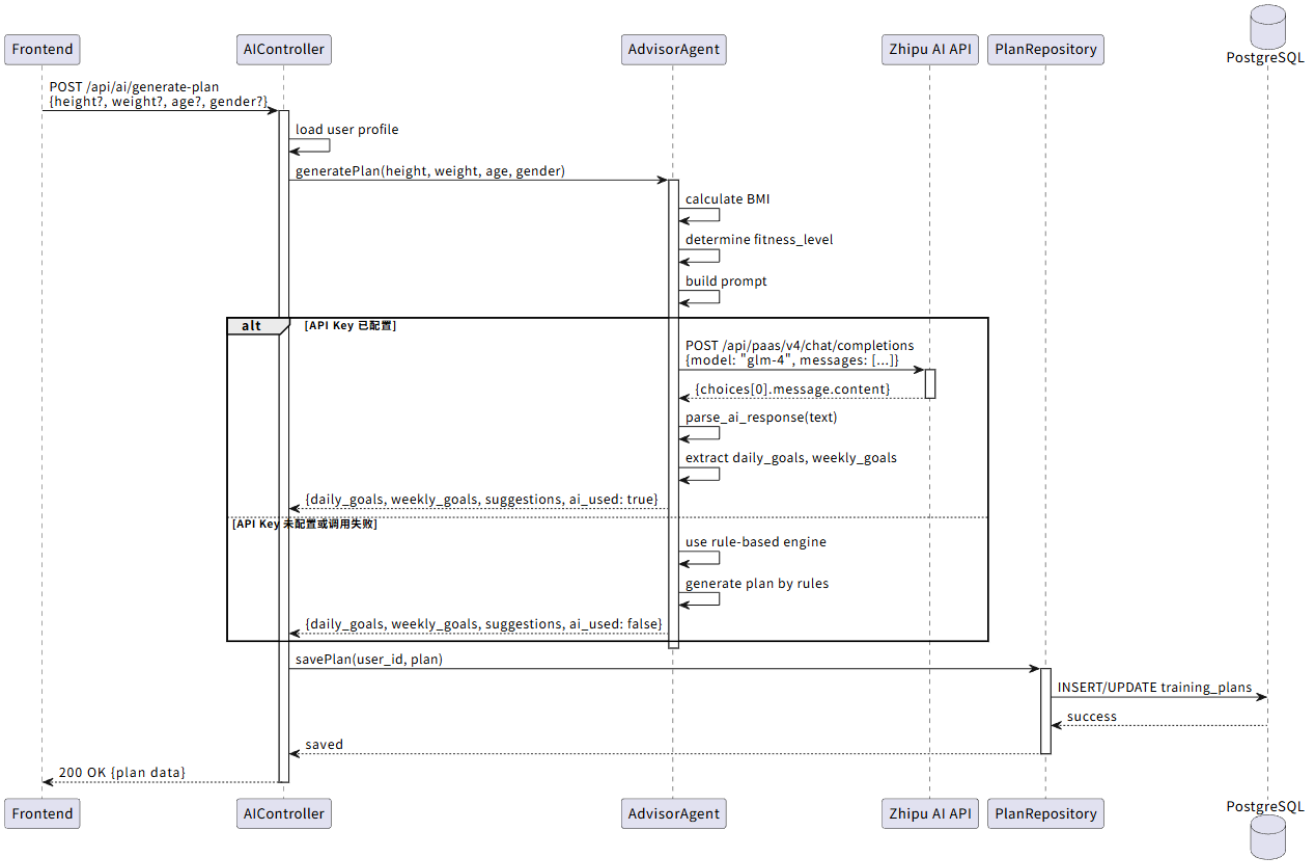


图5-22 AI 健身计划生成交互时序图

关键交互点：

1. **降级机制**：API 调用失败时自动使用规则引擎
2. **数据解析**：使用正则表达式提取结构化数据
3. **结果持久化**：生成的计划保存到数据库

5.10.2 组件协作模式

5.10.2.1 前端组件协作

前端采用**观察者模式**实现组件间通信：

- **usePoseDetection Hook** 作为**被观察者**（Subject），维护 `exerciseStats` 状态
- **StatsPanel** 作为**观察者**（Observer），订阅状态变更并自动更新 UI

5.10.2.2 后端服务协作

后端采用**服务层模式**实现业务编排：

- **Controller** 负责 HTTP 协议解析，调用 **Service** 执行业务逻辑
- **Service** 协调多个 **Repository** 和 **Agent**，实现复杂业务流程
- **Repository** 封装数据访问细节，向 Service 暴露领域对象接口

5.10.3 交互图

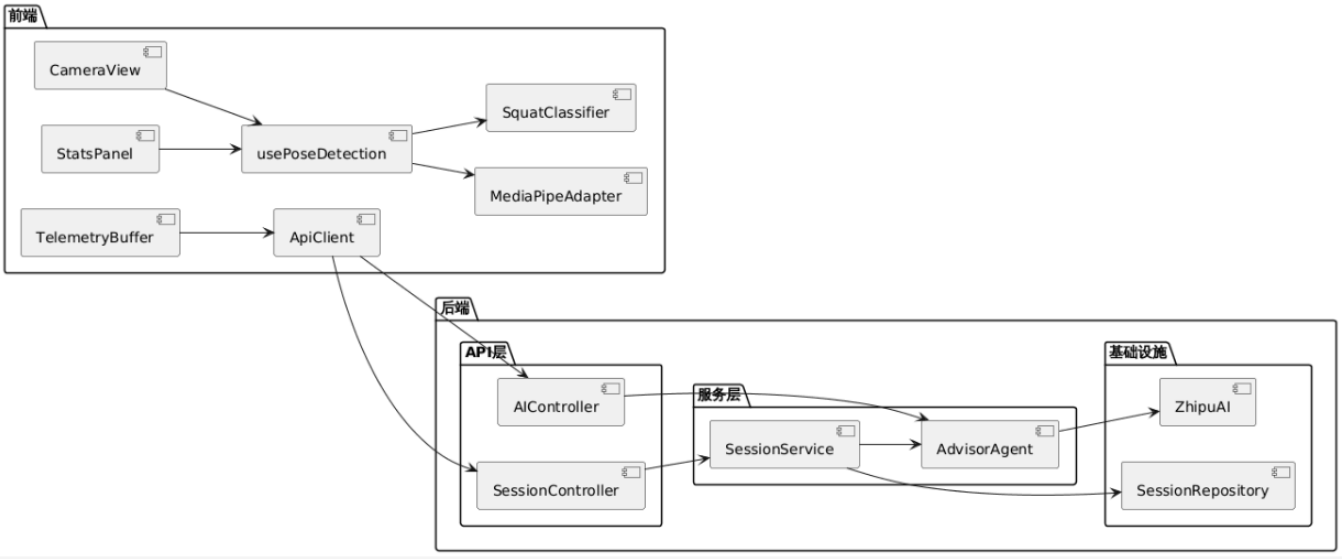


图5-23 组件协作关系图

5.11 状态动态视角 (State Dynamics Viewpoint)

状态动态视角描述系统及其组件在生命周期内的状态变迁，通过状态图、状态转换表揭示系统的动态行为模式。本视角重点关注关键实体的状态机模型。

5.11.1 训练会话状态机

Session（训练会话）是系统的核心状态实体，其生命周期包含以下状态：

5.11.1.1 状态定义

1. **INITIALIZED（已初始化）**：会话对象已创建，但尚未开始数据采集
2. **ACTIVE（活跃）**：会话正在进行，持续接收运动数据
3. **PAUSED（暂停）**：会话临时暂停，数据采集中断但会话未结束
4. **COMPLETED（已完成）**：会话正常结束，数据已保存，AI 总结已生成
5. **CANCELLED（已取消）**：会话异常终止，数据可能未完整保存

5.11.1.2 状态转换规则

当前状态	触发事件	下一状态	动作
INITIALIZED	start_session()	ACTIVE	设置 start_time，初始化计数器
ACTIVE	submit_data()	ACTIVE	更新 scores 数组，累计计数
ACTIVE	pause()	PAUSED	暂停数据采集
PAUSED	resume()	ACTIVE	恢复数据采集
ACTIVE	end_session()	COMPLETED	设置 end_time，调用 AI 生成总结
PAUSED	end_session()	COMPLETED	设置 end_time，调用 AI 生成总结
ACTIVE	cancel()	CANCELLED	标记为取消，不生成总结
PAUSED	cancel()	CANCELLED	标记为取消
INITIALIZED	cancel()	CANCELLED	删除会话记录

5.11.1.3 状态图

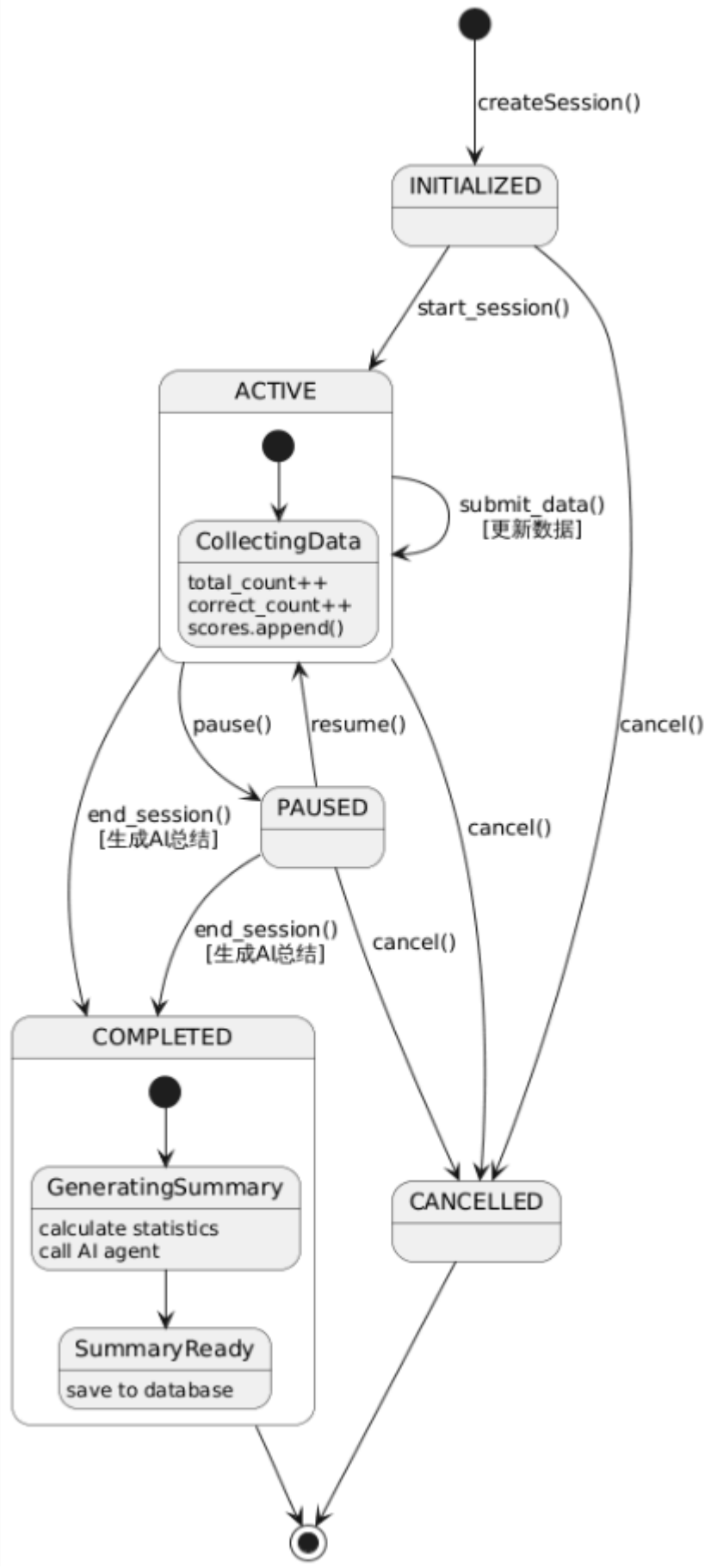


图5-24 训练会话状态机图

5.11.2 姿态检测状态机

usePoseDetection Hook 管理前端姿态检测的生命周期状态：

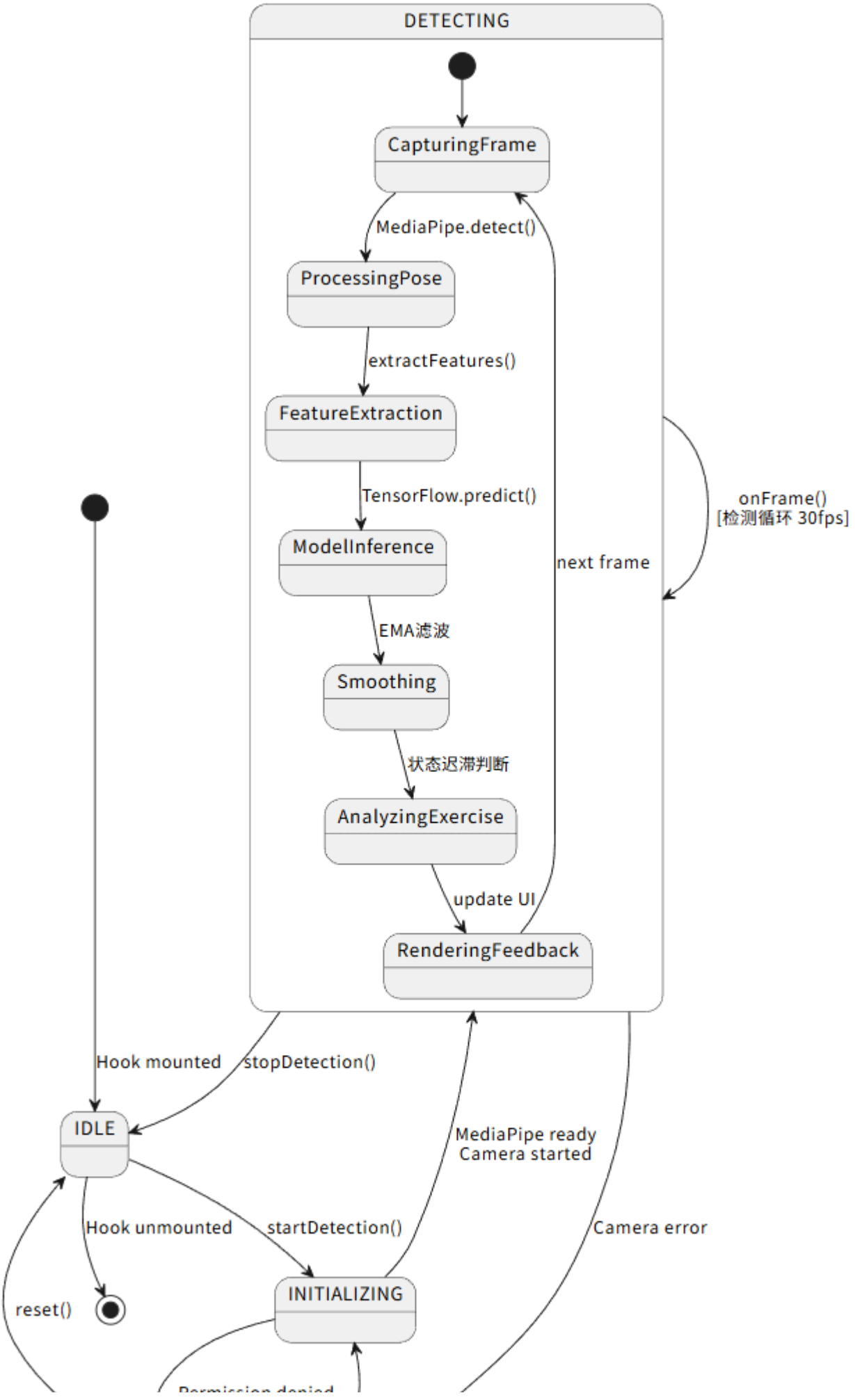
5.11.2.1 状态定义

1. **IDLE（空闲）**： Hook 已初始化，但未启动检测
2. **INITIALIZING（初始化中）**： 正在加载 MediaPipe 模型和启动摄像头
3. **DETECTING（检测中）**： 摄像头已启动，持续进行姿态检测
4. **ERROR（错误）**： 检测过程中发生错误（如摄像头权限被拒绝）

5.11.2.2 状态转换规则

当前状态	触发事件	下一状态	动作
IDLE	startDetection()	INITIALIZING	加载 MediaPipe，请求摄像头权限
INITIALIZING	onMediaPipeReady()	DETECTING	启动摄像头流，开始检测循环
INITIALIZING	onError()	ERROR	设置错误消息
DETECTING	stopDetection()	IDLE	停止摄像头，释放资源
DETECTING	onCameraError()	ERROR	设置错误消息
ERROR	retry()	INITIALIZING	重新初始化
ERROR	reset()	IDLE	重置状态

5.11.2.3 状态图


DETECTING
CapturingFrame
MediaPipe.detect()
ProcessingPose
extractFeatures()
FeatureExtraction
TensorFlow.predict()
ModelInference
EMA滤波
Smoothing
状态迟滞判断
AnalyzingExercise
update UI
RenderingFeedback
next frame
onFrame()
[检测循环 30fps]
Hook mounted
stopDetection()
IDLE
MediaPipe ready
Camera started
Hook unmounted
startDetection()
Camera error
reset()
INITIALIZING

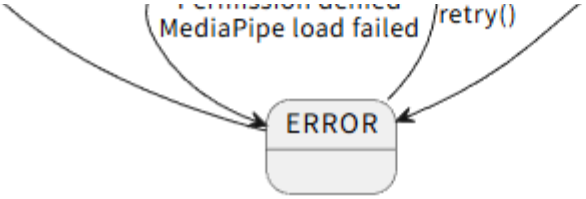


图5-25 姿态检测状态机图

5.11.3 运动分析器状态机

SquatClassifier（深蹲分类器）的状态机模型，基于深度学习模型的三类状态分类：

5.11.3.1 状态定义

深度学习模型输出三种状态分类：

1. **状态 0（站直）**：用户处于站立姿态，膝盖角度接近 180°
2. **状态 1（半蹲）**：用户处于半蹲姿态，膝盖角度在 90° 至 150° 之间
3. **状态 2（完全蹲下）**：用户处于完全蹲下姿态，膝盖角度小于 90°

5.11.3.2 状态转换规则

当前状态	条件	下一状态	动作
状态 0	模型预测状态1且置信度>0.7持续3帧	状态 1	进入半蹲阶段
状态 1	模型预测状态2且置信度>0.7持续3帧	状态 2	到达底部，记录时间戳
状态 2	模型预测状态1且置信度>0.7持续3帧	状态 1	开始起立
状态 1	模型预测状态0且置信度>0.7持续3帧	状态 0	完成一次，count++
*	置信度不足或异常	保持当前状态	等待稳定

状态迟滞机制：为防止状态边界反复跳变，仅当新状态的置信度**持续 3 帧**超过阈值 0.7 时，才触发状态切换。

5.11.3.3 状态图

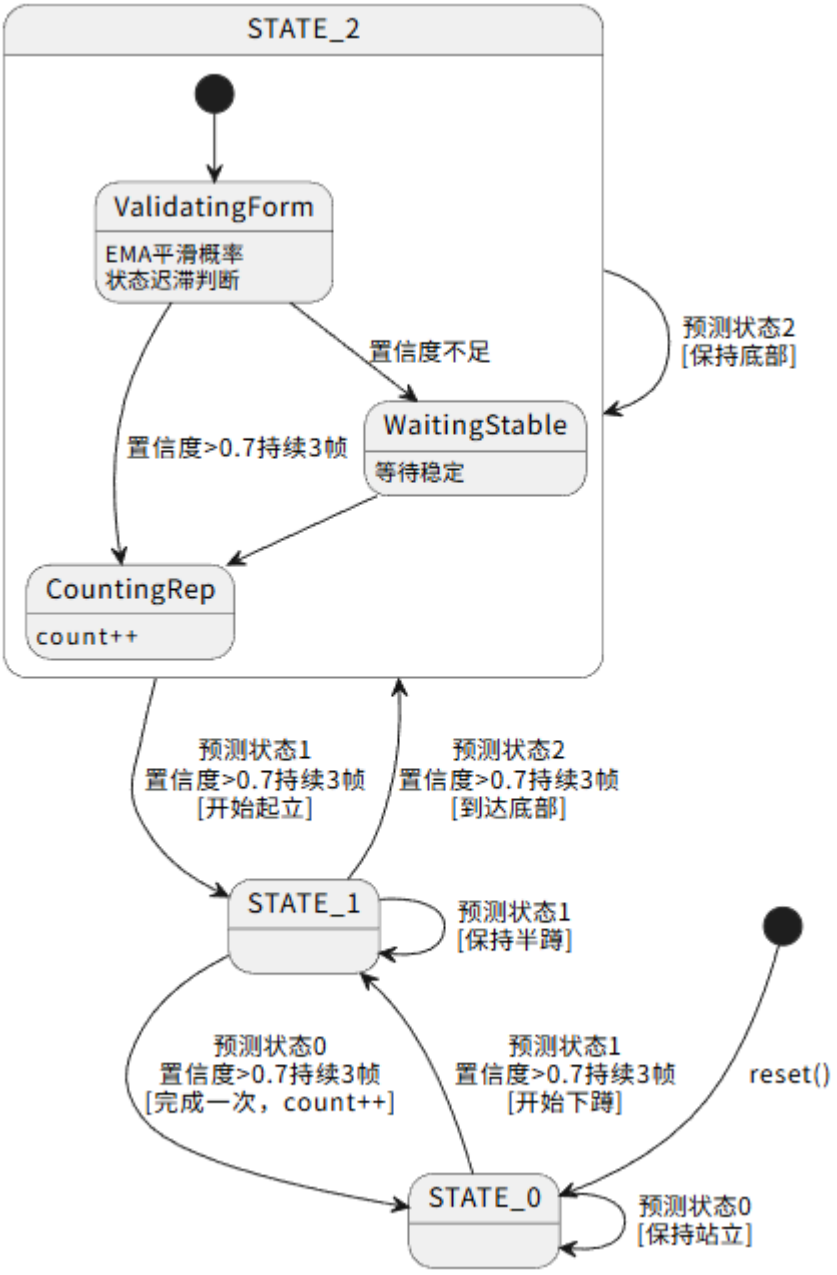


图5-26 深蹲分类器状态机图（基于深度学习模型）

5.11.4 用户认证状态机

AuthContext 管理用户认证状态：

5.11.4.1 状态定义

1. **UNAUTHENTICATED（未认证）**：用户未登录
2. **AUTHENTICATING（认证中）**：正在验证用户凭证
3. **AUTHENTICATED（已认证）**：用户已登录，Token 有效
4. **TOKEN_EXPIRED（Token 过期）**：Token 已过期，需要重新登录

5.11.4.2 状态转换规则

当前状态	触发事件	下一状态	动作
UNAUTHENTICATED	login(username, password)	AUTHENTICATING	发送登录请求
AUTHENTICATING	onSuccess(token)	AUTHENTICATED	保存 Token，设置用户信息
AUTHENTICATING	onError()	UNAUTHENTICATED	显示错误消息
AUTHENTICATED	logout()	UNAUTHENTICATED	清除 Token
AUTHENTICATED	tokenExpired()	TOKEN_EXPIRED	清除 Token，提示重新登录
TOKEN_EXPIRED	login()	AUTHENTICATED	重新认证

5.11.4.3 状态图

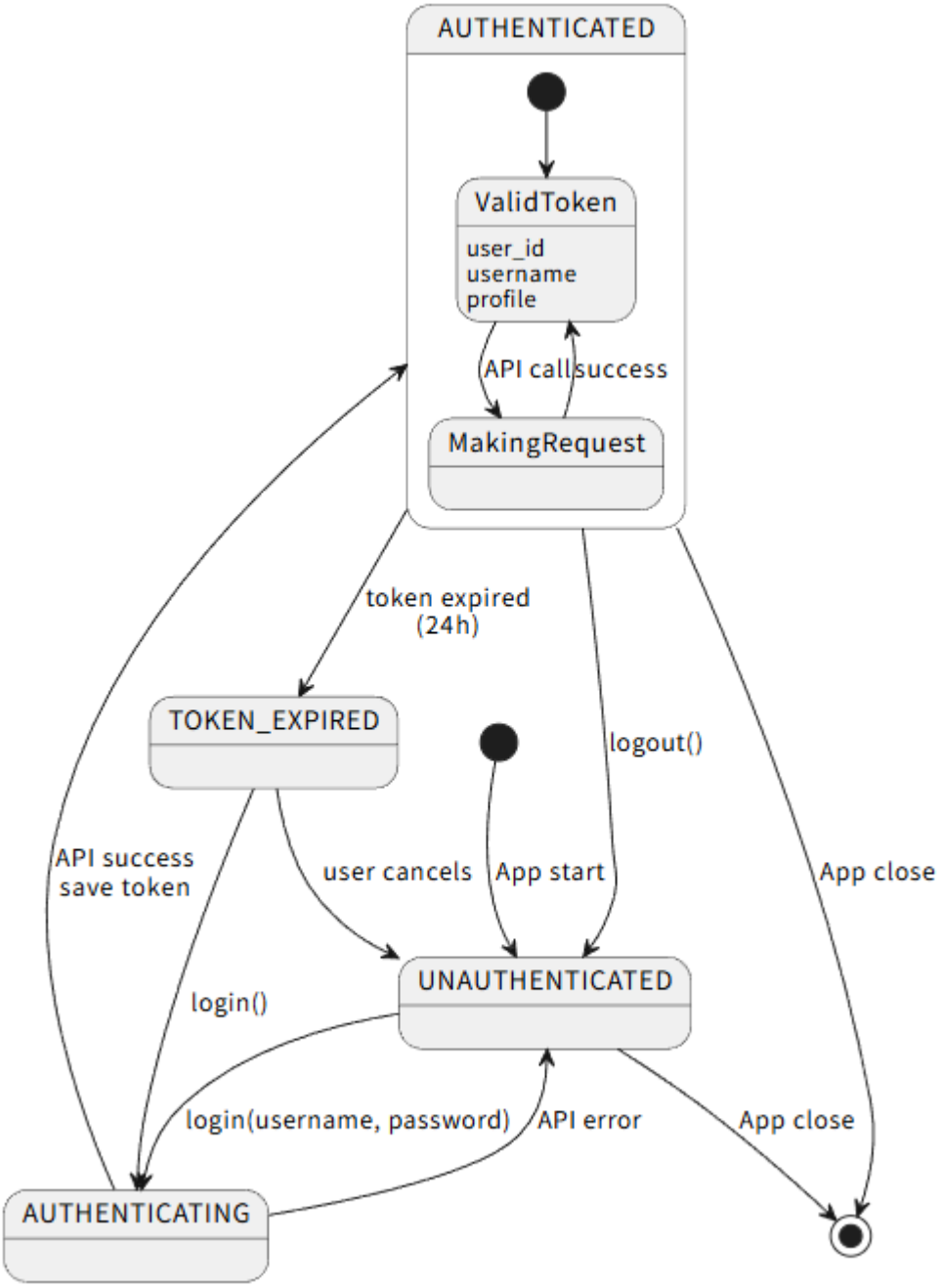


图5-27 用户认证状态机图

5.12 算法视角

算法视角详细说明 FitnessAI 系统中深度学习算法的处理步骤、计算公式、训练流程及预测推理的实现细节。本视角聚焦于深蹲状态分类器的核心算法实现，为开发人员提供精确的算法规约，确保模型训练与推理的一致性。

5.12.1 算法概述

FitnessAI 的深度学习模块采用**监督学习**方法，基于 MediaPipe 提取的 33 个姿态关键点坐标，通过特征工程与深度神经网络，实现对深蹲动作状态的自动分类。系统支持三种状态分类：

- **状态 0（站直）**：用户处于站立姿态，膝盖角度接近 180°。
- **状态 1（半蹲）**：用户处于半蹲姿态，膝盖角度在 90° 至 150° 之间。
- **状态 2（完全蹲下）**：用户处于完全蹲下姿态，膝盖角度小于 90°。

算法流程分为四个核心阶段：**特征提取**、**模型构建**、**模型训练**、**状态预测**。

5.12.2 特征提取算法

5.12.2.1 输入数据格式

特征提取算法的输入为 MediaPipe Pose 模型输出的 33 个关键点坐标。每个关键点 P_i 包含：

$$P_i = \{x, y, z, v\}$$

其中 $x, y, z \in [0, 1]$ 为归一化坐标，$v \in [0, 1]$ 为可见性置信度。

5.12.2.2 关键点索引映射

算法仅使用与深蹲动作相关的 8 个关键点（定义于 `MediaPipeIndices` 枚举）：

关键点名称	索引	用途
左/右肩 (SHOULDER)	11, 12	计算髋部角度、肩膀垂直高度
左/右髋 (HIP)	23, 24	计算膝盖角度、髋部垂直高度
左/右膝 (KNEE)	25, 26	计算膝盖角度、水平偏移量
左/右踝 (ANKLE)	27, 28	计算膝盖角度、高度基准点

5.12.2.3 角度计算算法

角度计算是特征工程的核心。计算以点 B 为顶点的夹角 $\angle ABC$。

计算公式：

构建向量 $\vec{BA}$ 和 $\vec{BC}$：

$$\vec{v_1} = A - B, \quad \vec{v_2} = C - B$$

计算夹角 θ（单位：度）：

$$\theta = \arccos\left(\frac{\vec{v_1} \cdot \vec{v_2}}{|\vec{v_1}| \, |\vec{v_2}|} \right) \times \frac{180}{\pi}$$

数值保护：

若 $|\vec{v_1}| \approx 0$ 或 $|\vec{v_2}| \approx 0$，则返回默认值 180.0°。

5.12.2.4 特征向量构建

函数 `extract_squat_features()` 输出一个 **24 维** 特征向量 $\mathbf{X}$，包含几何特征与原始坐标。

1. 基础几何特征（8 维）：

索引	特征名称	单位	物理意义
0, 1	左/右膝盖角度	度 ($^\circ$)	衡量腿部弯曲程度 (核心指标)
2, 3	左/右髋部角度	度 ($^\circ$)	衡量躯干前倾程度
4	髋关节相对高度	标量	$(y{hip} - y{ankle})$，衡量下蹲深度
5	肩膀相对高度	标量	$(y{shoulder} - y{ankle})$，衡量整体高度
6, 7	左/右膝水平偏移	标量	$(x{knee} - x{ankle})$，检测膝盖是否过脚尖

2. 原始坐标特征（16 维）：

包含 8 个关键点的 (x, y) 原始归一化坐标，用于保留空间位置信息，辅助模型学习非线性关系。

5.12.3 深度学习模型架构

5.12.3.1 网络结构设计

深蹲分类器采用**全连接前馈神经网络 (MLP)**。下图展示了各层的连接关系及维度变换。

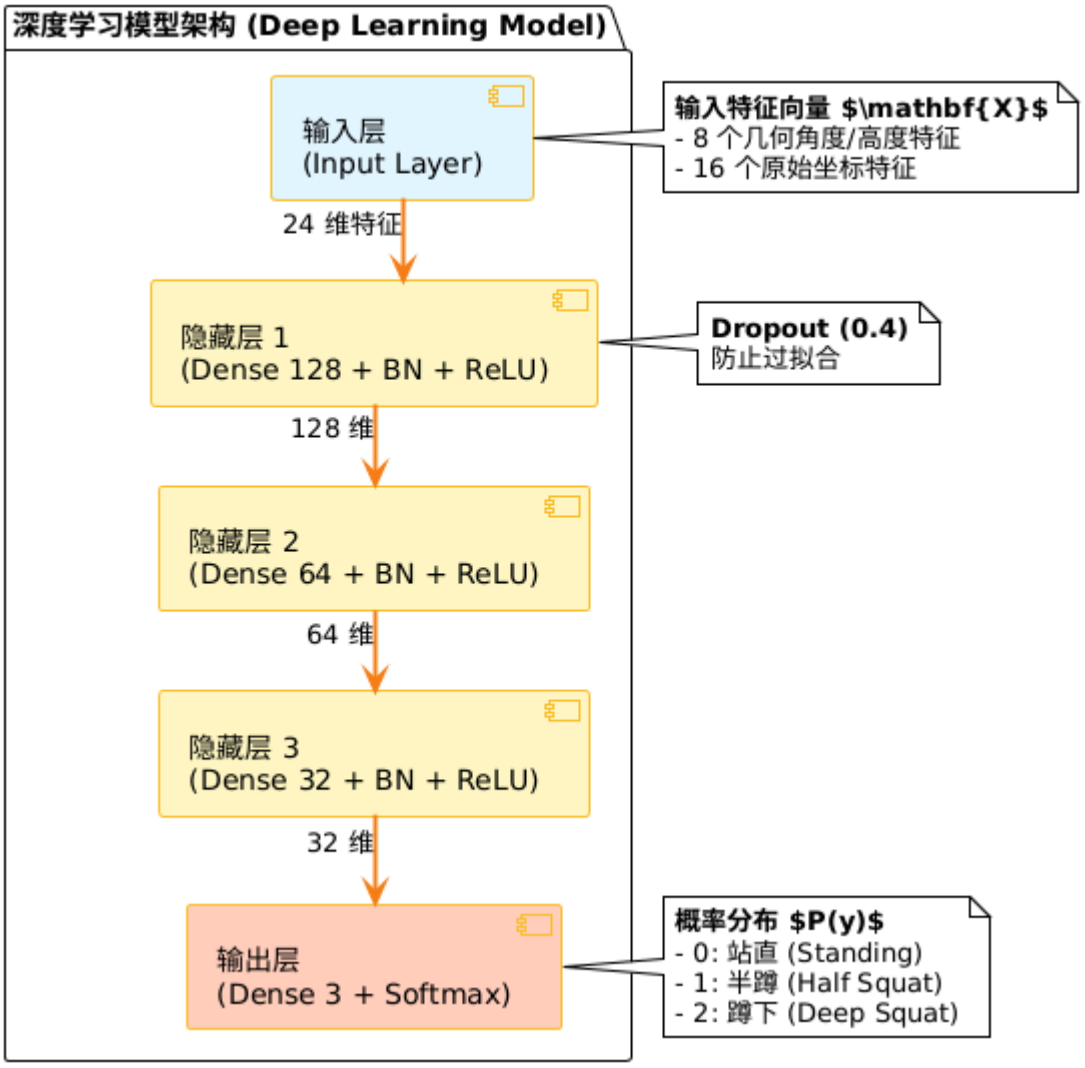


图5-28 深度学习模型架构图

5.12.3.2 核心层数学原理

1. 全连接层 (Dense)

执行线性变换与非线性激活：

$$\mathbf{h} = \text{ReLU}(\mathbf{W}\mathbf{x} + \mathbf{b})$$

2. 批归一化 (Batch Normalization)

标准化输入分布，加速收敛：

$$\hat{x} = \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \cdot \gamma + \beta$$

3. 输出层 (Softmax)

将 logits 转换为概率分布 $P(y)$：

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

其中 $C=3$ 为类别总数。

5.12.3.3 模型参数统计

层级	类型	输出形状	参数量
Input	InputLayer	(None, 24)	0
Hidden 1	Dense + BN	(None, 128)	3,456
Hidden 2	Dense + BN	(None, 64)	8,384
Hidden 3	Dense + BN	(None, 32)	2,144
Output	Dense (Softmax)	(None, 3)	99
Total	-	-	**~14,083**

5.12.4 模型训练算法

5.12.4.1 损失函数与优化器

- 损失函数：稀疏分类交叉熵 (Sparse Categorical Crossentropy)

  $$L = -\frac{1}{N} \sum_{i=1}^{N} \log(P(y_i^{true}))$$

- **优化器**：Adam
  - 初始学习率 $\alpha = 0.001$
  - 一阶矩衰减 $\beta_1 = 0.9$
  - 二阶矩衰减 $\beta_2 = 0.999$

5.12.4.2 数据增强策略

为了提高模型的泛化能力，防止过拟合，训练过程采用了以下增强手段：

1. 高斯噪声注入

在关键点坐标上叠加随机噪声，模拟摄像头抖动：

$$\mathbf{x}_{aug} = \mathbf{x}_{orig} + \mathcal{N}(0, \sigma^2)$$

其中 $\sigma = 0.01$。

2. 水平翻转

通过对"左/右"关键点进行索引互换及 X 坐标镜像，将样本量扩充 2 倍。

3. SMOTE 过采样

在特征空间对少数类（如"半蹲"）生成合成样本，平衡数据集分布。

5.12.4.3 训练策略优化

1. 类别权重平衡：

   针对"半蹲"样本较少的情况，自动计算权重：

   $$w_j = \frac{N_{total}}{C \times N_j}$$

2. 早停机制 (Early Stopping)：

   监控验证集损失 L_{val}，若连续 patience=15 个 Epoch 不下降，则终止训练。

5.12.5 预测推理算法

5.12.5.1 推理数据流

预测过程是一个从原始关键点到语义状态的转化流水线。

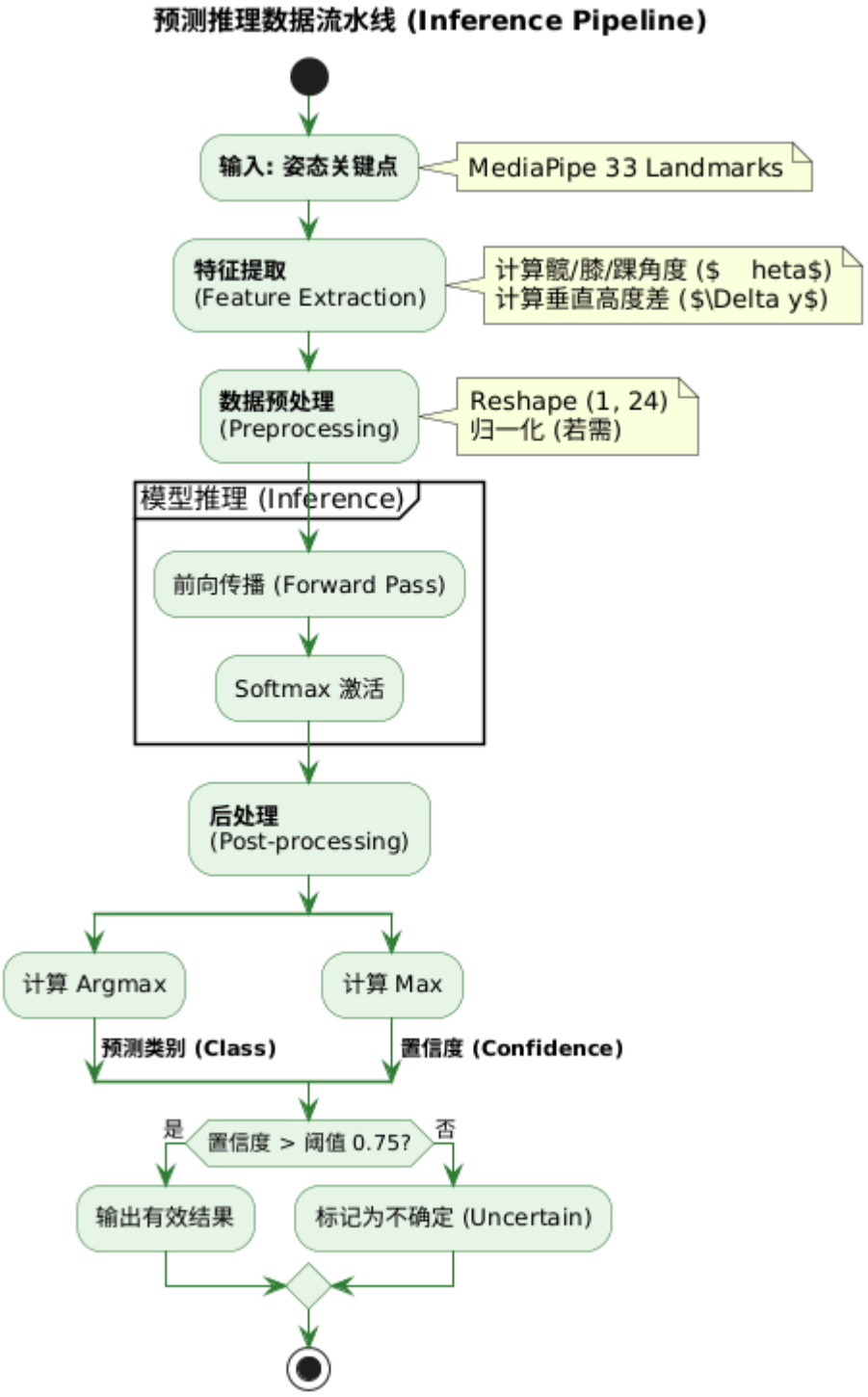


图5-29 预测推理数据流图

5.12.5.2 详细处理步骤

1. **输入接收**：接收 MediaPipe 每一帧的 `pose_landmarks`。
2. **特征提取**：调用 `extract_squat_features()` 获取 24 维向量。
3. **预处理**：执行 `reshape(1, 24)` 以匹配 batch 维度。
4. **前向传播**：模型输出 3 维概率向量 $[p_0, p_1, p_2]$。
5. **后处理与决策**：
    - 预测类别：$c = \operatorname{argmax}(p)$

- 置信度：$conf = \max(p)$

5.12.6 结果平滑与去抖动

由于视频流输入存在波动，系统引入 **EMA 滤波器** 和 **状态迟滞切换** 机制。

5.12.6.1 指数移动平均滤波 (EMA)

对概率向量进行时间序列平滑：

$$\hat{P}*t = *alpha *cdot P_t + *(1 - *alpha)* *cdot *hat{P}*{t-1}$$

其中 $\alpha=0.7$。此操作能显著减少 UI 闪烁。

5.12.6.2 状态迟滞切换

为防止状态边界反复跳变，系统规定：仅当新状态的置信度**持续 3 帧**超过阈值 0.7 时，才触发状态切换。

5.12.7 模型持久化与部署

1. 模型序列化

训练完成的模型被导出为 HDF5 (.h5) 格式，文件包含：

- 模型架构配置 (JSON)
- 权重参数矩阵 (Weights)
- 优化器状态（用于断点续训）

2. 运行时加载

后端服务启动时，SquatClassifier 单例类通过 TensorFlow/Keras API 加载模型文件至内存，并执行一次预热推理，输入全零向量，以消除首次请求的延迟。

5.12.8 算法验证与测试

模型性能通过混淆矩阵与 F1-Score 进行量化评估。

评估指标公式：

$$\text{Precision}*k = *frac{TP_k}{TP_k + *FP_k}, *quad *text{Recall}*k = \frac{TP_k}{TP_k + FN_k}$$

$$F1_k = 2 \times \frac{\text{Precision}*k *times *text{Recall}*k}{\text{Precision}*k + *text{Recall}*k}$$

基准性能要求：

- 总体准确率 (Accuracy) $> 85\%$
- 单次推理耗时 (Latency) $< 15ms$

5.13 资源视角 (Resource Viewpoint)

资源视角描述系统对计算资源、存储资源及网络资源的占用与分配策略。本视角关注性能优化、资源限制及可扩展性设计。

5.13.1 计算资源

5.13.1.1 前端计算资源

CPU 资源占用：

- **MediaPipe 推理**：每帧约 10-15ms CPU 时间（30fps 下）
- **特征提取**：每帧约 1-2ms CPU 时间（24维特征向量计算）
- **深度学习推理**：每帧约 5-10ms CPU 时间（TensorFlow.js 模型前向传播）
- **结果平滑**：每帧约 0.5ms CPU 时间（EMA滤波、状态迟滞判断）
- **Canvas 渲染**：每帧约 2-3ms CPU 时间（绘制骨骼连线）

总 CPU 占用：约 18-30ms/帧，满足 30fps 要求（33.33ms/帧）

GPU 资源占用：

- **MediaPipe WASM**：利用 WebGL 加速，GPU 占用约 20-30%
- **Canvas 2D 渲染**：CPU 渲染，不占用 GPU

内存资源占用：

- **MediaPipe 模型**：约 5-8 MB（WASM 二进制）
- **TensorFlow.js 模型**：约 2-3 MB（深度学习模型权重）
- **React 应用**：约 10-15 MB（组件树、状态）
- **视频流缓冲**：约 2-3 MB（640×480 @ 30fps，1秒缓冲）
- **总内存占用**：约 20-30 MB

5.13.1.2 后端计算资源

CPU 资源占用：

- **Flask 请求处理**：平均 5-10ms/请求（简单 CRUD）
- **AI 代理调用**：异步处理，不阻塞主线程
- **深度学习推理引擎**：TensorFlow 推理，单次约 10-15ms（特征提取+模型推理）

内存资源占用：

- **Flask 应用**：约 50-100 MB（Python 运行时）
- **TensorFlow 模型**：约 200-500 MB（如加载 `.h5` 模型文件）
- **会话数据缓存**：约 10-50 MB（内存中暂存活跃会话）

并发处理能力：

- **Flask 单进程**：约 50-100 并发请求（Gunicorn 多进程可扩展）
- **数据库连接池**：最大 20 个连接

5.13.2 存储资源

5.13.2.1 前端存储

LocalStorage：

- **用户 Token**：约 500 bytes
- **用户偏好设置**：约 1-2 KB
- **总占用**：< 5 KB

SessionStorage：

- **临时会话数据**：约 10-50 KB（训练过程中的中间数据）

5.13.2.2 后端存储

数据库存储（PostgreSQL）：

users 表：

- 单条记录：约 500 bytes
- 10,000 用户：约 5 MB

sessions 表：

- 单条记录：约 2-5 KB（包含 JSONB 遥测数据）
- 100,000 会话：约 200-500 MB

training_plans 表：

- 单条记录：约 1-2 KB（JSONB AI 计划）
- 10,000 计划：约 10-20 MB

总存储估算：

- **小规模**（1,000 用户，10,000 会话）：约 50-100 MB
- **中规模**（10,000 用户，100,000 会话）：约 500 MB - 1 GB
- **大规模**（100,000 用户，1,000,000 会话）：约 5-10 GB

文件存储：

- **日志文件**：每日约 10-50 MB（按日志级别）
- **模型文件**（如使用）：约 200-500 MB（`.h5` 格式）

5.13.3 网络资源

5.13.3.1 前端网络流量

初始加载：

- **HTML**：约 10 KB
- **CSS**：约 50-100 KB

- **JavaScript Bundle**：约 500 KB - 1 MB（压缩后）
- **MediaPipe WASM**：约 5-8 MB（CDN 加载）
- **总初始加载**：约 6-10 MB

运行时流量：

- **API 请求**：平均 1-5 KB/请求
- **数据提交频率**：批量提交，每 5-10 秒一次，约 10-50 KB/批次
- **总运行时流量**：约 100-500 KB/分钟（取决于使用频率）

5.13.3.2 后端网络流量

API 响应：

- **认证接口**：约 500 bytes - 1 KB
- **会话接口**：约 1-5 KB
- **AI 接口**：约 2-10 KB（AI 返回文本）

外部服务调用：

- **智谱 AI API**：请求约 1-2 KB，响应约 2-5 KB
- **调用频率**：每用户每次训练结束调用一次

5.13.4 资源分配策略

5.13.4.1 前端资源优化

代码分割（Code Splitting）：

- 按路由懒加载组件，减少初始 bundle 大小
- MediaPipe 库按需加载，不阻塞首屏渲染

资源缓存：

- MediaPipe WASM 文件使用 CDN 缓存，减少重复下载
- API 响应使用 HTTP 缓存头，减少重复请求

内存管理：

- 及时释放视频流资源（`stream.getTracks().forEach(track => track.stop())`）
- 使用 `useCallback` 和 `useMemo` 避免不必要的重渲染

5.13.4.2 后端资源优化

数据库优化：

- **索引策略**：在 `user_id`, `session_id`, `created_at` 上创建索引
- **JSONB 索引**：对 `telemetry_log` 字段创建 GIN 索引，加速 JSON 查询
- **连接池**：复用数据库连接，减少连接开销

缓存策略：

- **会话数据缓存**：活跃会话数据暂存内存，减少数据库查询
- **AI 响应缓存**：相同输入参数的 AI 请求缓存结果（可选）

异步处理：

- AI 代理调用使用异步任务队列，不阻塞主请求线程
- 数据归档使用后台任务，避免影响实时性能

5.13.5 资源限制与扩展性

5.13.5.1 前端资源限制

浏览器限制：

- **内存限制**：约 2-4 GB（取决于设备）
- **存储限制**：LocalStorage 约 5-10 MB
- **并发请求限制**：同域约 6 个并发连接

应对策略：

- 使用 Web Workers 分担计算密集型任务
- 数据批量提交，减少 HTTP 请求频率
- 使用 IndexedDB 存储大量历史数据（如需要）

5.13.5.2 后端资源限制

服务器限制：

- **单进程并发**：约 50-100 请求/秒
- **数据库连接**：最大 20 个连接（可配置）

扩展策略：

- **水平扩展**：使用 Gunicorn 多进程 + Nginx 负载均衡
- **数据库扩展**：读写分离，主从复制
- **缓存层**：引入 Redis 缓存热点数据

5.13.6 资源监控图

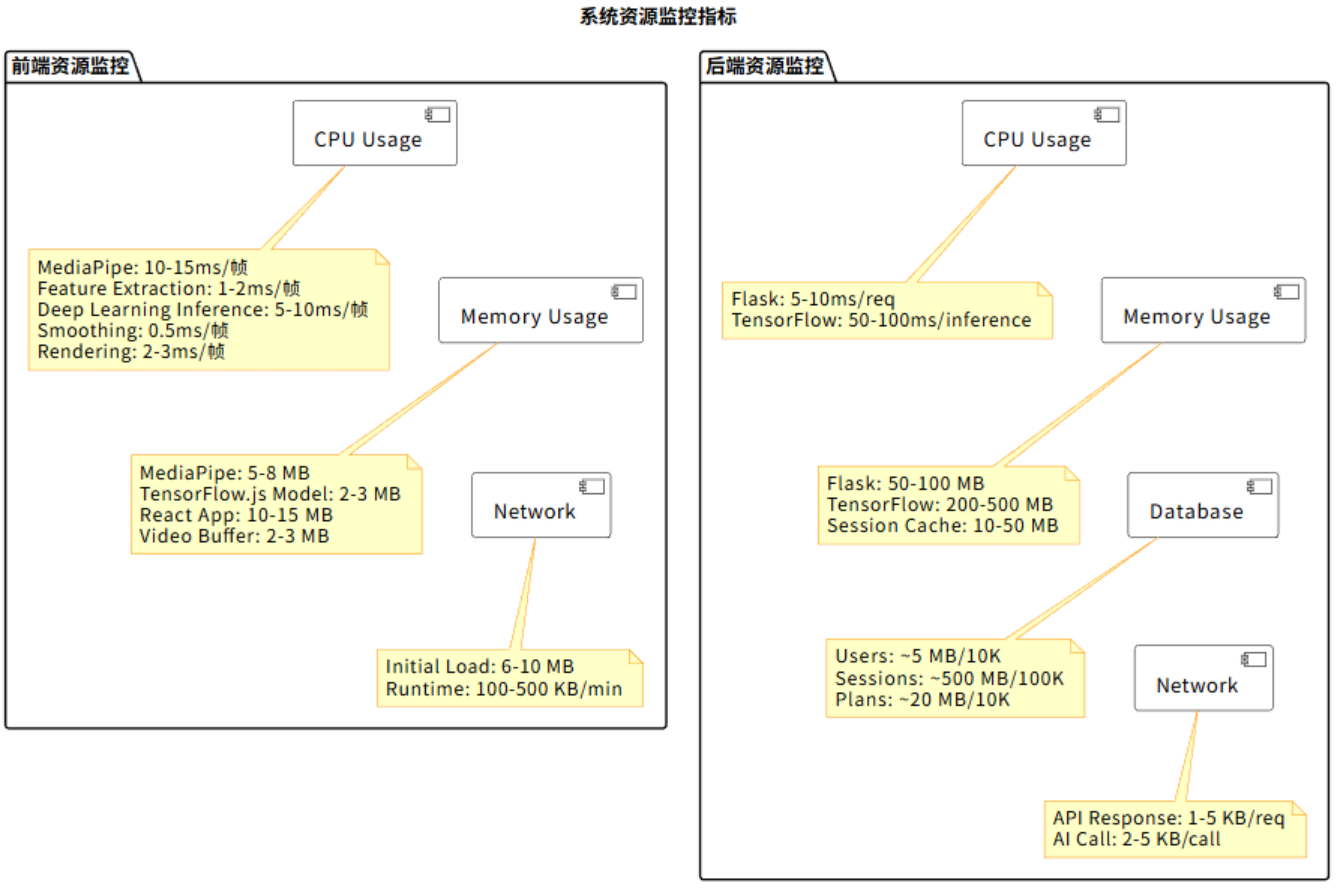


图5-30 系统资源占用概览图

5.13.7 性能基准

5.13.7.1 前端性能指标

指标	目标值	实际值	备注
首屏加载时间	< 3s	~2-3s	取决于网络速度
姿态检测延迟	< 100ms	~50-80ms	浏览器内闭环（包含深度学习推理）
帧率	≥ 30fps	~30fps	稳定运行
内存占用	< 50 MB	~20-30 MB	正常使用

5.13.7.2 后端性能指标

指标	目标值	实际值	备注
API 响应时间 (P95)	< 200ms	~100-150ms	简单 CRUD
API 响应时间 (P99)	< 500ms	~200-300ms	包含数据库查询
AI 接口响应时间	< 30s	~5-15s	取决于网络

指标	目标值	实际值	备注
数据库查询时间	< 50ms	~20-40ms	带索引查询
并发处理能力	≥ 50 req/s	~50-100 req/s	单进程
