Diseño de Arquitectura del Sistema - Modelo basado en IA para detección de somnolencia

**Fecha:** 10/04/2025

**Autor: Tirza Buendia**

**Versión:** 1.0

# 1. Introducción

Este documento describe de manera detallada la arquitectura del sistema para el modelo basado en IA, cuyo propósito es detectar la somnolencia al volante y generar una alerta temprana y precisa. Se define la estructura general del sistema, los componentes que lo conforman, sus interacciones y las tecnologías empleadas.

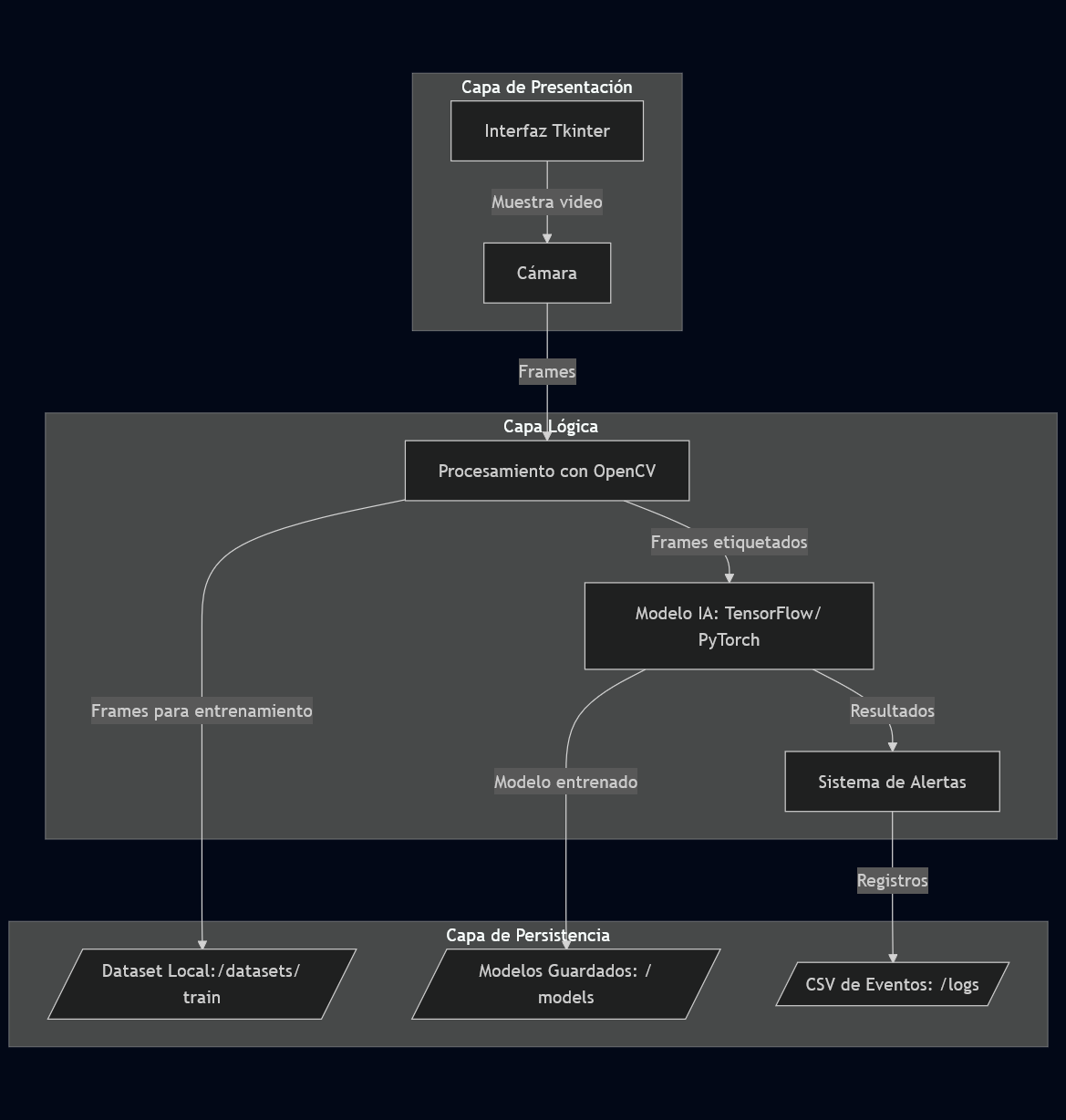
# 2. Visión General de la Arquitectura

## 2.1. Estilo Arquitectónico

El sistema adopta una arquitectura en capas, basada en el patrón cliente-servidor, y se organiza en tres capas principales:

* **Capa de Presentación (Frontend):** Interfaz gráfica para interacción con el conductor.
* **Capa de Lógica de Negocio (Backend):** Procesamiento de IA y gestión de alertas.
* **Capa de Persistencia de Datos:** Almacenamiento de eventos de somnolencia.

## 2.2. Diagrama General de Arquitectura:



# 3. Descripción de Componentes Principales

## 3.1. Frontend (Presentación)

* Framework/Librería: Tkinter (Python) para interfaz minimalista.
* Responsabilidades:
  + Mostrar video en tiempo real de la cámara.
  + Visualizar métricas (PERCLOS, bostezos, cabeceos).
  + Renderizar alertas visuales (cambio de color, iconos).

## 3.2. Backend (Lógica de negocio)

* Lenguaje y Frameworks: Python con OpenCV, TensorFlow/PyTorch.
* Responsabilidades:
  + Detección facial: Uso de OpenCV/MediaPipe para landmarks.
  + Modelo de IA: CNN para análisis de somnolencia (PERCLOS, bostezos, inclinación).
  + Alertas: Generación de tonos (85 dB) y notificaciones visuales.
  + API REST: Opcional para futura integración móvil (FastAPI/Flask).

### 3.2.1. Módulos del Backend

* deteccion\_facial.py: Captura y procesamiento de rostros.
* analisis\_somnolencia.py: Cálculo de PERCLOS, MAR (bostezos), ángulo de cabeza.
* gestor\_alertas.py: Gestión de alertas sonoras/visuales.
* registro\_csv.py: Registro de eventos en CSV.

## 3.3. Base de Datos (Persistencia)

* Sistema Gestor: Archivos CSV.
* Esquema de Datos:
  + timestamp | PERCLOS | bostezos\_por\_minuto | inclinación\_cabeza | estado\_conductor

# 4. Integraciones Externas (Opcionales)

* Hardware: Cámara estándar (720p), posible integración con Raspberry Pi en futuras versiones.
* Monitorización: Logs con Python logging para depuración.

# 5. Seguridad

# 6. Escalabilidad y Despliegue

Posibles problemas que pueden surgir durante el desarrollo.

* **Frontend**: Desplegado en Vercel o Netlify
* **Backend**: Deploy en Render, Railway o Heroku
* **Base de Dato**s: PostgreSQL en Supabase o ElephantSQL
* La arquitectura permite escalar horizontalmente el backend y separar la base de datos en instancias dedicadas

# 7. Conclusiones

La arquitectura propuesta combina eficiencia (procesamiento en tiempo real con latencia <1s), modularidad (capas independientes para fácil mantenimiento) y escalabilidad, permitiendo su adaptación desde prototipos hasta implementaciones en hardware embebido como Raspberry Pi mediante Python y librerías optimizadas (OpenCV, TensorFlow Lite), con capacidad para integrar nuevos componentes mediante modificaciones puntuales sin afectar el sistema completo, ofreciendo así un rendimiento robusto tanto en entornos de desarrollo como en dispositivos con recursos limitados, lo que la convierte en una solución flexible y preparada para el futuro.