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.

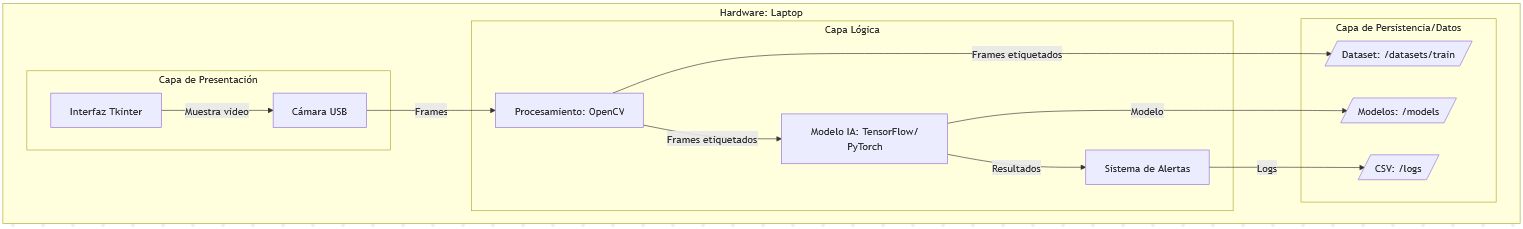
# 2. Visión General de la Arquitectura

## 2.1. Estilo Arquitectónico

El sistema adopta una arquitectura en capas monolítica, ya que no cuenta con capas que estén distribuidas y no utiliza servicios externos (todo se comunica dentro del mismo entorno). Se organiza en tres capas principales dentro de la capa de hardware:

* **Capa de Presentación (Frontend):** Interfaz gráfica pasiva que muestra el video en tiempo real, las métricas (apertura de ojos, bostezos por minuto, inclinación de cabeza), y emisión de la alarma en caso que se identifique somnolencia.
* **Capa de Lógica de Negocio (Backend):** Procesamiento de IA y gestión de alertas.
* **Capa de Persistencia de Datos:** Almacenamiento local de eventos de somnolencia, frames (datasets de entrenamiento), y el modelo .

## 2.2. Diagrama General de Arquitectura:



Mermaid:

flowchart TB

subgraph HW["Hardware: Laptop"]

subgraph Capa de Presentación

A[Interfaz Tkinter] -->|Muestra video| B[Cámara USB]

end

subgraph Capa Lógica

B -->|Frames| C[Procesamiento: OpenCV]

C -->|Frames etiquetados| D[Modelo IA: TensorFlow/PyTorch]

D -->|Resultados| E[Sistema de Alertas]

end

subgraph Capa de Persistencia/Datos

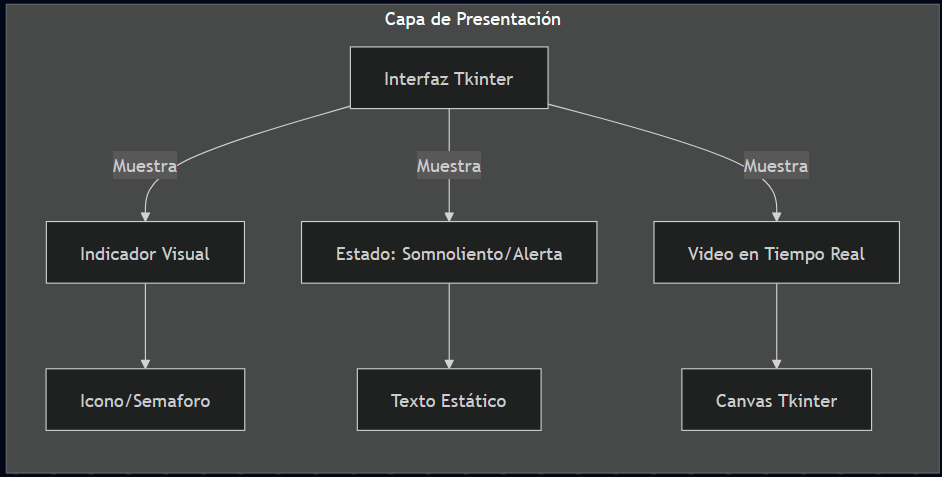
C -->|Frames etiquetados| F[/Dataset: /datasets/train/]

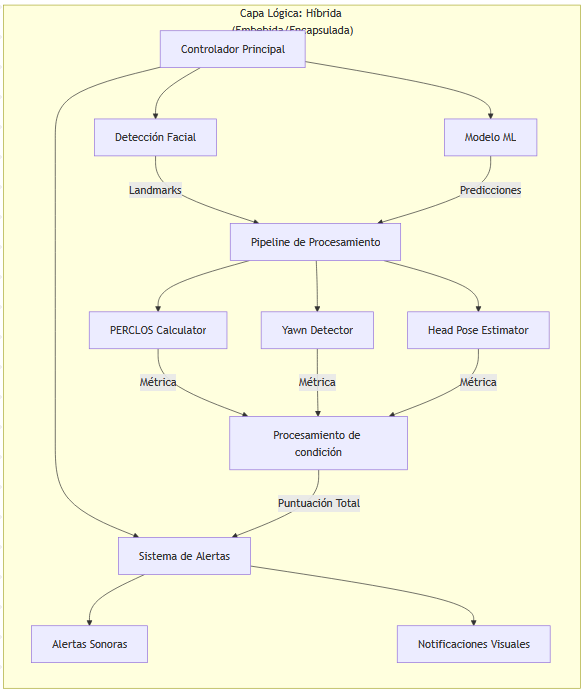
D -->|Modelo| G[/Modelos: /models/]

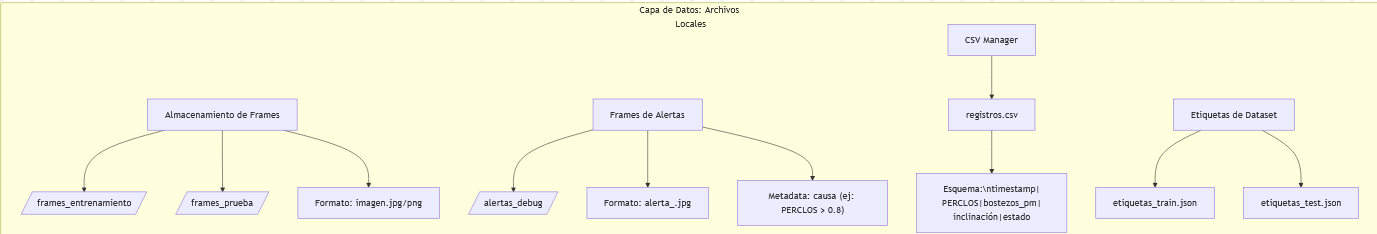
E -->|Logs| H[/CSV: /logs/]

end

en







Mermaid (para visualización de diagrama):

flowchart TD

subgraph "Capa de Datos: Archivos Locales"

%% Almacenamiento principal

A[Almacenamiento de Frames] --> B[/frames\_entrenamiento/]

A --> C[/frames\_prueba/]

A --> D[Formato: imagen.jpg/png]

%% Almacenamiento para alertas (debugging)

K[Frames de Alertas] --> L[/alertas\_debug/]

K --> M["Formato: alerta\_<timestamp>.jpg"]

K --> N["Metadata: causa (ej: PERCLOS > 0.8)"]

%% CSV y etiquetas

E[CSV Manager] --> F[registros.csv]

F --> G["Esquema:\ntimestamp|PERCLOS|bostezos\_pm|inclinación|estado"]

H[Etiquetas de Dataset] --> I[etiquetas\_train.json]

H --> J[etiquetas\_test.json]

end

# 3. Descripción de Componentes Principales

## 3.1. Frontend (Presentación)

El tipo de capa de presentación será una **Interfaz Gráfica local (GUI)** puesto que solo se mostrará la salida del modelo (estado de conductor), las alertas si se detecta somnolencia, y el video en tiempo real. Todo esto se realizará sin la necesidad de recibir input del usuario, por lo que es una interfaz de salida pasiva.

* Framework/Librería: Tkinter (Python) para interfaz minimalista.
* Responsabilidades:
  + Mostrar video en tiempo real de la cámara, procesado para la detección de somnolencia.
  + Visualizar métricas relacionadas con el estado del conductor, como PERCLOS, bostezos y ángulo de cabeza.
  + Renderizar alertas visuales (cambio de color de fondo, mostrar iconos, mensajes de alerta) si se detecta somnolencia.

## 3.2. Backend (Lógica de negocio)

El tipo de capa lógica será una combinación de lógica embebida y lógica encapsulada debido a que el modelo corre en un mismo programa (embebido) y usa módulos/funciones diferentes para separar diferentes tareas (encapsulado).

* Lenguaje y Frameworks: Python con OpenCV, TensorFlow/PyTorch.
* Responsabilidades:
  + **Detección facial**: Uso de OpenCV o MediaPipe para identificar los landmarks faciales y procesar la imagen.
  + **Modelo de IA**: Implementación de machine learning para analizar la somnolencia mediante métricas como PERCLOS, bostezos y la inclinación de la cabeza.
  + **Alertas**: Generación de alertas sonoras (85 dB) y notificaciones visuales (cambio en la interfaz) para alertar al conductor si el sistema detecta signos de somnolencia.

### 3.2.1. Módulos del Backend

* deteccion\_facial.py: Módulo encargado de la captura y procesamiento de la imagen para identificar el rostro del conductor y sus características faciales.
* analisis\_somnolencia.py: Cálculo de las métricas de somnolencia como PERCLOS (porcentaje de ojos cerrados), cantidad de bostezos por minuto y ángulo de inclinación de la cabeza.
* gestor\_alertas.py: Gestiona la creación y disparo de alertas, tanto sonoras (tono de 85 dB) como visuales en la interfaz (cambio de color, iconos, mensajes).
* registro\_csv.py: Registra los eventos y resultados del sistema en un archivo CSV para seguimiento y análisis posterior (por ejemplo, hora de la alerta, PERCLOS, etc.).

## 3.3. Base de Datos (Persistencia)

El tipo de capa de datos será una combinación de archivos planos (CSV) y dataset local ya que los frames de entrenamiento del modelo, al igual que los frames de prueba del modelo, serán almacenados localmente en la capa de hardware por lo que no se necesitará una base de datos. Asimismo, los registros de actividad serán almacenados en archivos CSV también de manera local.

El tipo de capa de datos será una combinación de archivos planos (CSV) y dataset local ya que los frames de entrenamiento del modelo y los frames de prueba serán almacenados localmente en la capa de hardware. Asimismo, los registros de actividad se almacenarán en archivos CSV de manera local. Y adicionalmente, se guardarán frames de alertas críticas para debugging en una carpeta específica.

* Sistema Gestor: Archivos CSV.
* Esquema de Datos:
  + timestamp | PERCLOS | bostezos\_por\_minuto | inclinación\_cabeza | estado\_conductor
* Formato: .jpg/png
* Estructura:
  + /frames\_entrenamiento/ (dataset inicial)
  + /frames\_prueba/ (evaluación)
  + /alertas\_debug/ (frames con eventos críticos)

# 4. Integraciones Externas (Opcionales)

* Hardware: Cámara estándar (720p).
* 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.