## **Estructura del sistema de inferencia**

Este sistema está diseñado para operar de forma modular y “eficiente” sobre múltiples IAs de visión por computadora. El enfoque está en torno a separar responsabilidades clave y evitar condicionales innecesarios mediante **callables especializados** generados en tiempo de inicialización a partir de configuraciones JSON por IA.

### **Flujo general del controlador**

El controlador opera 7 pasos principales:

1. **Obtener el frame** Captura del frame desde el cliente o fuente de video. *(por implementar)*
2. **Preprocesar el frame (genérico)** A partir del JSON asociado a la IA activa, el módulo transformers.py genera una función de preprocesamiento. Esta función convierte un frame crudo a un formato tensorial, con operaciones como resize, normalización, y letterboxing. Este paso es **genérico**, independiente de la IA, y reutilizable entre distintos modelos con entrada similar.
3. **Adaptar preprocesamiento a la IA específica** Algunos modelos requieren entradas con estructuras o layouts que no pueden expresarse únicamente con el preprocesamiento general (por ejemplo, un layout NCHW, tensores adicionales, u orden de canales específico).  
    Aquí entra en juego adapters.py, que aplica transformaciones específicas al tensor resultante para que se alinee con las necesidades internas del modelo cargado. *(por implementar)*
4. **Generar la inferencia** Se ejecuta una **función de inferencia predefinida** (y no genérica) que fue entregada por model\_loader. Esta función encapsula completamente la forma particular de ejecución del modelo (TFLite, ONNX, TorchScript, etc.).  
    model\_loader simplemente actúa como un **dador de la función de inferencia**, no como un adaptador ni un manejador de entrada/salida.
5. **Adaptar raw\_output al formato genérico del controlador** El output bruto de cada IA puede tener estructuras arbitrarias y no siempre compatibles con el flujo común del sistema (por ejemplo, vectores en orden [cx, cy, w, h] en vez de [y1, x1, y2, x2], o sin índice de clase explícito).  
    adapters.py proporciona funciones que transforman esta salida a un formato genérico y estructurado, de modo que transformers.py pueda aplicar el postproceso sin condiciones. *(en desarrollo)*
6. **Postprocesar las detecciones** A partir del output ya adaptado, transformers.py aplica pasos como filtrado por confianza, Non-Max Suppression (NMS) y limpieza de estructuras. Esta etapa es también **precompilada en una función callable** según el JSON de la IA.
7. **Devolver al cliente** Se retransforman los resultados (por ejemplo, dibujando detecciones en una imagen base) y se devuelven al cliente. *(por implementar)*

## **Modelo JSON: explicación detallada línea por línea**

Cada IA se acompaña de un archivo JSON con su configuración precisa, que define sus requerimientos de entrada y salida. A continuación se desglosa un ejemplo completo:

json

CopyEdit

{

"model\_type": "detection",

"input": {

"width": 640,

"height": 640,

"channels": 3,

"normalize": true,

"mean": [0.0, 0.0, 0.0],

"std": [1.0, 1.0, 1.0],

"letterbox": true,

"auto\_pad\_color": [114, 114, 114],

"preserve\_aspect\_ratio": true

},

"output": {

"confidence\_threshold": 0.5,

"nms\_threshold": 0.45,

"apply\_nms": true,

"tensor\_structure": {

"format": "yxyx",

"coordinates": {

"y1": 0,

"x1": 1,

"y2": 2,

"x2": 3

},

"confidence\_index": 4,

"class\_index": 5

}

}

}

### **🔹 "model\_type": "detection"**

Indica el tipo de IA que se está utilizando. Esto se usa para activar módulos especializados (por ejemplo, en postprocesado) según sea detección, segmentación o clasificación.

### **🔹 input**

Define cómo debe preprocesarse la imagen para alimentar la IA:

* "width" y "height": dimensiones esperadas por el modelo. Se redimensiona el input a este tamaño.
* "channels": cantidad de canales de color (comúnmente 3 para RGB).
* "normalize": si se deben normalizar los valores del input.
* "mean" y "std": media y desviación estándar por canal para normalización.
* "letterbox": indica si se debe aplicar letterbox para preservar aspecto.
* "auto\_pad\_color": color del padding si se aplica letterbox.
* "preserve\_aspect\_ratio": si se debe mantener la proporción original al redimensionar (requiere letterbox).

### **🔹 output**

Contiene los parámetros que el postprocesado usará para limpiar los resultados:

* "confidence\_threshold": mínimo valor de confianza para aceptar una predicción.
* "nms\_threshold": umbral de superposición para aplicar NMS.
* "apply\_nms": si debe aplicarse o no Non-Max Suppression.

#### **tensor\_structure**

Define cómo interpretar el output del modelo crudo:

* "format": "yxyx": orden esperado de las coordenadas de las cajas.
* "coordinates": mapeo explícito de índices de cada coordenada.
* "confidence\_index": índice en el tensor donde se encuentra la confianza.
* "class\_index": índice donde se encuentra la clase predicha.

Esto es vital para que el adaptador de outputs sepa cómo transformar un output arbitrario a una estructura homogénea compatible con transformers.py.

## **Adaptadores (adapters.py)**

Los adaptadores se volvieron necesarios al evidenciar que el controlador transforma y consume entradas y salidas de manera **genérica y predefinida**, pero no todas las IAs respetan estos formatos.

Por ejemplo:

* Algunas redes esperan inputs como [1, 3, 640, 640] en vez de [1, 640, 640, 3].
* Algunos outputs están codificados en layouts diferentes, o devuelven diccionarios en vez de tensores.

El controlador no puede manejar estos casos con condicionales. La solución fue centralizar toda esta lógica en **adaptadores de entrada y salida** que se generan automáticamente a partir de los JSONs.

### **Objetivos de los adaptadores:**

* Transformar la salida del preprocesamiento (tensor genérico) al formato requerido por la IA.
* Transformar el output crudo del modelo al formato estandarizado esperado por el postproceso.

Esto permite que:

* model\_loader se mantenga enfocado solo en entregar la función de inferencia.
* transformers.py no tenga condiciones internas ni suposiciones de estructura.
* El controlador utiliza funciones homogéneas, sin preocuparse por el modelo.

## **Cómo se generan los callables al registrar una nueva IA**

Cuando el sistema detecta que se debe usar una nueva IA (por solicitud del cliente), el controlador sigue este flujo:

1. **Lee el JSON asociado al nombre del modelo.**
2. **Valida el JSON con Pydantic** para asegurar que todos los campos son correctos y completos.
3. **Ejecuta los generadores de transformers.py**, creando:  
   * preprocess\_fn: función para preparar el input.
   * postprocess\_fn: función para limpiar el output.
4. **Ejecuta los generadores de adapters.py**, creando:  
   * input\_adapter\_fn: transforma el input genérico al requerido por el modelo. *(por hacer)*
   * output\_adapter\_fn: transforma el output del modelo al formato estándar. *(por hacer)*
5. **Obtiene la función de inferencia desde model\_loader** según el tipo de modelo.
6. **Asigna todo en un entorno local (contexto del modelo actual)**.

A partir de ahí, el ciclo de inferencia puede ejecutarse con llamadas puramente funcionales y sin ramificaciones.