Informe Técnico: Implementación de Zoom-Style Bokeh con Data Science

** Estructuras de Datos Eficientes: @dataclass y Perfiles**

Al declarar la clase QualityPreset con el decorador **@dataclass**, Python genera automáticamente métodos especiales, siendo el más importante el **constructor (__init__)**, además de facilitar la representación y comparación de datos.

```
@dataclass
class QualityPreset:
    """Configuraciones de calidad similares a Zoom"""
    name: str
    blur_radius: int
    edge_refinement: int
    temporal_frames: int
    description: str
```

Se establece un diccionario global llamado PRESETS con tres llaves: low, medium y high. Esto permite que, al consultar por ejemplo PRESETS['high'], el sistema devuelva un objeto estructurado con los parámetros exactos:

> QualityPreset(name='High', blur_radius=27, edge_refinement=9, temporal_frames=5, description='Best quality, stronger blur')

** Arquitectura de la Clase ZoomStyleBokeh: Encapsulamiento**

Dentro de la clase, los atributos internos (como self.seg_options y self.preset) están ocultos del usuario final (**Encapsulamiento**). El sistema utiliza argumentos por defecto para facilitar su uso:

- **selfie_segmenter.tflite**: Es el archivo que contiene la "inteligencia" (el modelo). Al ponerlo por defecto, se evita escribir la ruta manualmente siempre que el archivo esté en la misma carpeta "**models/**".
- **preset: str = "high"**: Si se crea el objeto como procesador = ZoomStyleBokeh(), el sistema asume la máxima calidad, extrayendo automáticamente los valores del diccionario (27 de blur, 9 de refinamiento y 5 de frames temporales).
- **mode: BlurMode = BlurMode.BLUR**: No es solo un texto; mediante un **Enum (Enumeración)**, se restringe el sistema para que solo acepte valores definidos en la clase BlurMode (BLUR o REPLACE).

- **background_image: Optional[str] = None**: Indica que, por defecto, no hay imagen de fondo. Si el modo es REPLACE pero no se pasa una imagen, el código incluye lógica interna para manejar ese error.

 Configuración del Modelo: vision.ImageSegmenterOptions

Esta es una clase de la librería **MediaPipe (Google)** que define el comportamiento del modelo antes de procesar imágenes. Es fundamental distinguir dos conceptos:

1. **La Instancia:** Al llamarla con paréntesis y argumentos, creas un objeto de configuración específica.
2. **El Atributo:** Al asignar self.seg_options = ..., guardas ese objeto dentro de la clase para que métodos posteriores puedan acceder a él.

Parámetros Críticos de Segmentación:

- **model_asset_path**: Indica a MediaPipe la ubicación exacta del "cerebro" (.tflite).
- **output_category_mask=False**: Evitamos el uso de máscaras binarias simples (clase 0 o 1) para ahorrar memoria.
- **output_confidence_masks=True**: La verdadera clave del Data Science. Pedimos una **máscara de probabilidades** donde cada píxel tiene un valor entre 0.0 (0% seguro de que es persona) y 1.0 (100% seguro). Esto es lo que permite bordes suaves y profesionales en lugar de recortes toscos.

 Nota sobre Dimensiones y Tensores:

Es fundamental entender que mientras el video original es un tensor de 3 canales (RGB), la máscara generada es una matriz bidimensional.

- **Video:** (720, 1280, 3):(Alto, Ancho, Canales de Color).
- **Máscara:** (720, 1280):(Alto, Ancho).

Esta reducción de dimensionalidad es necesaria para que el sistema pueda realizar operaciones matemáticas de "alpha blending" (mezcla de capas) de manera eficiente sin desajustes (*mismatches*) de memoria.

 Programación Defensiva y Manejo de Datos

El código está diseñado para ser robusto ante errores del usuario mediante técnicas de **programación defensiva**:

1. **Uso de .get()**: En la línea self.preset = PRESETS.get(preset, PRESETS["high"]), si el usuario ingresa una clave inexistente (como "ultra"), el programa no se rompe con un KeyError, sino que usa el preset "high" por defecto.
2. **Validación de Imagen de Fondo**:
    - if background_image: Verifica que el usuario haya pasado un valor.
    - os.path.exists(...): Confirma que el archivo realmente esté en el disco duro.
    - cv2.imread(...): Carga la imagen y la convierte en una **matriz de NumPy (píxeles)** para su manipulación matemática.

 Flexibilidad de Experimentos

La clase permite probar diferentes escenarios de manera inmediata según la necesidad del proyecto:

- **Para rapidez:** ZoomStyleBokeh(preset="low")
- **Para fondos virtuales:** ZoomStyleBokeh(mode=BlurMode.REPLACE, background_image="playa.jpg")

Ajuste de Umbrales (Threshold Tuning)

self.base_threshold = 0.08

Este es el umbral mínimo para considerar que un píxel **podría** ser parte de una persona.

- **Por qué es tan bajo:** Un valor de 0.08 significa que si la IA tiene apenas un 8% de seguridad, ya lo tomamos en cuenta.
- **Objetivo:** No perder detalles finos y difíciles como **cabellos sueltos, hilos de ropa o los bordes de las orejas**. Si pusiéramos un 0.5 (50%), el recorte se vería muy "mordido" y artificial.

---

2. self.protection_threshold = 0.12

Este umbral se usa más adelante en el código para "proteger" el cuerpo del usuario del desenfoque.

- **Qué hace:** Identifica las zonas donde la IA está un poco más segura (12%).
- **Objetivo:** Asegurar que el **interior del cuerpo** (tu cara, pecho, brazos) sea 100% sólido y nítido. Evita que el efecto de desenfoque "se filtre" hacia adentro de tu silueta, algo que pasa mucho en aplicaciones de baja calidad.

---

3. self.edge_threshold = 0.03

Es el umbral más sensible de todos (solo 3% de confianza).

- **Qué hace:** Se usa para crear la máscara binaria inicial que luego se procesa con los contornos de OpenCV.
- **Objetivo:** Detectar la **silueta máxima posible**. Al ser tan bajo, permite que el sistema detecte el "aura" de la persona, lo que da margen para que los filtros de suavizado (como el *Guided Filter*) trabajen sobre una zona amplia y logren un difuminado natural entre tú y el fondo.

 `Morfología por Regiones (Region-Specific Dilation)`

El sistema no aplica el mismo filtro a todo el cuerpo. Utiliza diferentes **kernels** (matrices de transformación) para expandir la máscara de forma inteligente:

- **`self.head_dilate = (7, 5)`**: Es una dilatación pequeña.
    - **Propósito:** En la cabeza (especialmente el cabello) queremos precisión. Si dilatamos mucho, el fondo se filtrará entre los cabellos. Se usa un kernel más alto que ancho para respetar la verticalidad de la cabeza.
- **`self.shoulder_dilate = (30, 9)`**: Es una dilatación muy ancha y plana.
    - **Propósito:** Los hombros suelen tener bordes más definidos pero la IA a veces los "recorta" de más. Al usar un valor de `30` en horizontal, nos aseguramos de cubrir toda la extensión de la ropa hacia los lados.
- **`self.body_dilate = (20, 12)`**: Un balance para el resto del torso.
    - **Propósito:** Da estabilidad al cuerpo principal, asegurando que el movimiento de los brazos no rompa la máscara.

Extensiones Laterales (Shoulder Extension)

self.right_extend = 25
self.left_extend = 15

Estos valores son una **solución heurística** (una regla basada en la experiencia) para resolver un problema común en las videollamadas:

- **Asimetría:** Normalmente, los humanos no estamos perfectamente centrados. Estas extensiones añaden píxeles extra a los lados de los hombros de forma artificial.
- **Efecto de "Halo" Suave:** Se usan para crear un degradado (alpha) que haga que la transición entre tu ropa y el fondo desenfocado no sea un choque visual brusco, sino un desvanecimiento suave.

Filtrado de Falsos Positivos (Heurística Geométrica)

Un modelo de Deep Learning puede cometer errores. Para asegurar que solo procesamos a la **persona real**, el sistema aplica una serie de filtros basados en la geometría del objeto detectado:

1. Ratio de Área (`area_ratio`)

- **`min_person_area_ratio = 0.05`**: Si el objeto detectado ocupa menos del 5% de la pantalla, es demasiado pequeño para ser el usuario principal (podría ser una mano a lo lejos o ruido).
- **`max_person_area_ratio = 0.85`**: Si ocupa más del 85%, probablemente es un error de la cámara (como cuando se tapa el lente) o un fondo mal segmentado que llenó toda la pantalla.

2. Relación de Aspecto (`aspect_ratio`)

- **`min_aspect_ratio = 0.3`** y **`max_aspect_ratio = 4.0`**:
  - Un ser humano frente a una cámara web suele tener una forma más alta que ancha o proporcional.
  - Si el objeto es extremadamente ancho y plano (como una mesa) o excesivamente delgado, el sistema lo descarta porque no cumple con las proporciones humanas estándar.

3. Área Mínima de Contorno (`min_contour_area = 5000`)

- Elimina pequeñas "manchas" o fragmentos de píxeles que la IA cree que son personas pero que son solo ruido visual. Esto asegura que solo nos enfoquemos en masas de píxeles significativas.

4. Margen de Borde (`edge_margin = 50`)

- Este valor define una "zona de seguridad" de 50 píxeles alrededor del borde del video.
- Se usa para penalizar objetos que están "cortados" en los bordes, ayudando a decidir si un objeto en la esquina es realmente una persona entrando a cuadro o simplemente una distorsión del fondo.

Estabilidad Temporal (Temporal Stability)

Para lograr un acabado profesional tipo Zoom, el sistema implementa una **"memoria de corto plazo"**. En lugar de calcular cada cuadro de forma aislada, el algoritmo analiza la coherencia entre el cuadro actual y los anteriores.

1. Historial de Máscaras (`mask_history`)

- **`deque(maxlen=self.preset.temporal_frames)`**: Un `deque` es una lista optimizada que, al llenarse, expulsa el elemento más antiguo para dejar entrar al nuevo.

- **Técnica de Sliding Window (Ventana Deslizante):** El sistema promedia las últimas máscaras (3, 4 o 5 según el preset). Si un píxel parpadea por error en un solo frame, el promedio de los otros 4 lo corrige, eliminando el ruido visual.

2. Estado Previo (`prev_frame_gray` y `prev_mask`)

- Estas variables almacenan la información del cuadro anterior en escala de grises y su máscara correspondiente.
- **Propósito:** Permiten calcular el **Flujo Óptico (Optical Flow)**. Si el sistema detecta que te moviste hacia la derecha, "empuja" la máscara vieja hacia la derecha antes de compararla con la nueva, logrando un seguimiento mucho más suave.

3. Umbral de Movimiento (`motion_threshold = 0.02`)

- Define qué tanto cambio debe haber entre dos cuadros para que el sistema considere que hubo movimiento real.
- **Lógica de Eficiencia:** Si el cambio es menor al 2%, el sistema asume que el usuario está quieto y mantiene la máscara anterior. Esto ahorra procesamiento y evita que el fondo "vibre" debido al ruido digital de la cámara en condiciones de poca luz.

Estado de Doble Pasada (Two-Pass State Management)

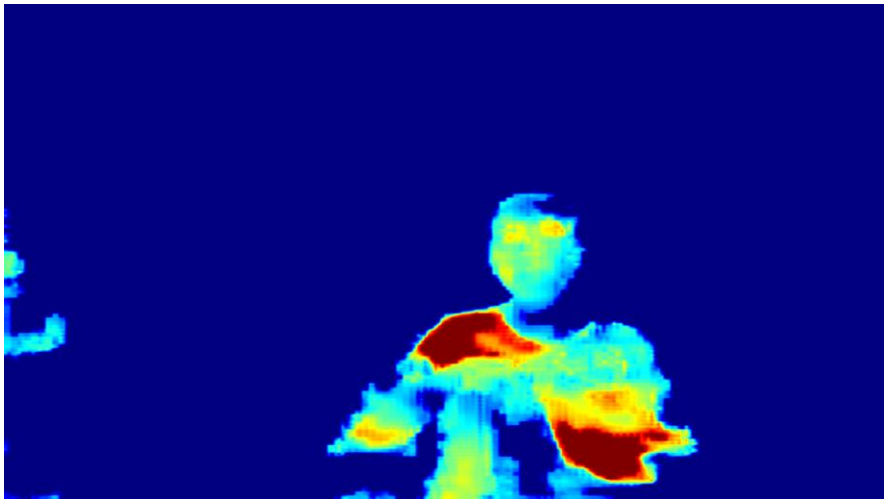
A diferencia de un filtro sencillo, este procesador utiliza una arquitectura de **análisis previo y ejecución posterior**. Estas variables permiten al algoritmo "conocer" la escena completa antes de renderizar el primer segundo de video.

1. Variables de Selección de Élite (best_mask y best_quality)

- **self.best_mask**: Almacena la máscara de segmentación del cuadro que obtuvo la mayor puntuación de confianza y contigüidad en todo el video.
- **self.best_quality**: Es el valor numérico (score) de esa máscara. Se usa como punto de referencia para comparar cada cuadro nuevo. Si un cuadro es mejor que el anterior, "destrona" al antiguo y se guarda como el nuevo estándar de oro.

2. Máscara Acumulada (accumulated_mask)

- Es una de las piezas más ingeniosas del script. A medida que el video avanza en la Pass 1, esta variable va "sumando" todas las áreas donde detectó a una persona.
- **Impacto técnico:** Crea un mapa de calor de la zona de actividad del usuario. Esto ayuda a que, en la Pass 2, el sistema sepa que si hay una detección fuera de esa zona acumulada, probablemente sea un error o un objeto del fondo.



3. Seguimiento Espacial (person_center y person_bbox)

- **self.person_center**: Guarda las coordenadas (x, y) del centro de masa del usuario.
- **self.person_bbox**: Guarda el cuadro delimitador (Bounding Box) que encierra a la persona.
- **Propósito en Data Science:** Estas métricas permiten calcular la **estabilidad de la pose**. Si el centro de la persona salta bruscamente de un cuadro a otro sin explicación física, el sistema detecta una anomalía y suaviza la transición mediante interpolación.

 Monitoreo de Rendimiento (Performance Tracking)

Para que un sistema sea considerado de "calidad de producción", debe ser capaz de autoevaluar su eficiencia. Estas variables permiten auditar el comportamiento del pipeline en diferentes condiciones de hardware.

1. Registro de Tiempos (`frame_times`)

- **`self.frame_times = []`**: Es una lista que almacena la duración exacta (en milisegundos o segundos) que toma procesar cada cuadro, desde la captura hasta el renderizado final.
- **Uso Estadístico:** Al finalizar el proceso, esta lista permite calcular:
  - **Media y Mediana:** Para saber el rendimiento promedio.
  - **Percentil 99 (P99):** Para identificar los cuadros más lentos y entender por qué hubo picos de lag (por ejemplo, cuando hay mucho movimiento y la IA trabaja más).

2. Contador de Cuadros (`frame_count`)

- **`self.frame_count = 0`**: Un contador simple pero esencial.

- **Propósito:** Sirve como denominador para los cálculos de progreso y para sincronizar la lógica de las dos pasadas. También permite imprimir en consola el estado del proceso (ej: *"Procesando frame 500/1200"*).

□ Método: `_filter_false_positives` (Filtrado Geométrico)

Este método actúa como un "limpiador" de la máscara cruda. Su objetivo es convertir una nube de puntos probables en objetos sólidos con forma definida.

1. Binarización de la Máscara

binary = (mask > 0.5).astype(np.uint8)

Qué hace: Toma la máscara de confianza (que tiene valores como 0.72, 0.45, etc.) y aplica un corte radical en 0.5

Resultado: Todo lo que tenga más del 50% de probabilidad se convierte en **1 (Blanco)** y el resto en **0 (Negro)**. Esto crea una imagen en blanco y negro pura, necesaria para que OpenCV pueda detectar bordes.

2. Detección de Contornos (cv2.findContours)

contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

Esta es una de las funciones más potentes de OpenCV. Imagina que pasas un lápiz por todo el borde de las manchas blancas que quedaron en la imagen.

- **RETR_EXTERNAL**: Le dice a la función que solo le importan los contornos de afuera. Si tienes un "hueco" dentro de la persona (como el espacio entre el brazo y el torso), lo ignorará para enfocarse en la silueta general.
- **CHAIN_APPROX_SIMPLE**: Es una técnica de compresión. En lugar de guardar todos los puntos de una línea recta, solo guarda los dos puntos de los extremos. Esto ahorra mucha memoria y hace que el cálculo posterior sea mucho más rápido.

□ Lógica de Validación y Puntuación

Una vez obtenidos los contornos, el sistema no los acepta a ciegas. Comienza un proceso de **filtrado selectivo** basado en el contexto de la imagen.

1. El Seguro de Fallos (if not contours)

if not contours:

return mask

- **Qué hace:** Verifica si OpenCV encontró al menos una mancha blanca en la imagen.
- **Impacto técnico:** Si la IA no detectó absolutamente nada (por ejemplo, la cámara está tapada), el programa devuelve la máscara original vacía y evita que el resto del código intente procesar datos que no existen. Esto previene el error más común en programación: el IndexError o intentar acceder a una lista vacía.

2. Contextualización Espacial (frame_area)

frame_area = h * w

- **Propósito:** Calcula el área total del video en píxeles.
- **Por qué es importante:** Para que los umbrales que definiste antes (min_person_area_ratio = 0.05) funcionen, el sistema necesita saber cuánto mide la "cancha de juego". Al calcular h * w, el sistema puede decir: *"Este contorno mide 50,000 píxeles, ¿es eso más o menos del 5% del total del video?"*. Esto hace que tu código sea **agnóstico a la resolución** (funciona igual en un video 4K que en uno de 720p).

3. Preparación para el Ranking (scored_contours)

scored_contours = []

- **Qué hace:** Inicializa una lista vacía para almacenar los contornos que pasen las pruebas, junto con una "calificación".
- **Lógica de Data Science:** En lugar de simplemente elegir el primer contorno que aparezca, el sistema va a evaluar a todos los candidatos. Esto es útil si, por ejemplo, hay una persona en primer plano y un cuadro con una cara al fondo; el sistema puntuará más alto a la persona real para ignorar el cuadro.

 Análisis Geométrico: Momentos de Imagen y Distancia Euclidiana

1. ¿Qué es cv2.moments(contour)?

En física, los "momentos" se usan para calcular la distribución de masa de un objeto. En visión artificial, tratamos los píxeles blancos de la máscara como "masa". El diccionario M contiene varios valores, pero los tres más importantes para nosotros son:

- **m00**: Es el momento de orden cero. Representa el **área total** (sumatoria de todos los píxeles blancos).

- **m10**: Es el momento de primer orden en el eje X. Representa la suma de las posiciones X de todos los píxeles.
- **m01**: Es el momento de primer orden en el eje Y. Representa la suma de las posiciones Y de todos los píxeles.

2. Cálculo del Centroide (cx, cy)

Para hallar el centro exacto de la persona (el "centroide"), usamos las fórmulas:

cx = m10/m00

cy = m01/m00

- **Lógica de Resiliencia Matemática if M["m00"] > 0**: Es una medida de seguridad. Si el área (m00) es cero, no podemos dividir por ella (error matemático). En ese caso, el código usa el centro del cuadro delimitador (x + bw // 2) como plan B.

 Desglose de (x + bw // 2)

Para entender esta fórmula, primero debemos recordar cómo OpenCV entiende las coordenadas:

- **x**: Es la coordenada del borde **izquierdo** del rectángulo.
- **bw** (*Bounding Width*): Es el **ancho total** del rectángulo.
- **// 2**: Es una división entera por dos (elimina los decimales).

La lógica paso a paso:

1. Si hiciéramos x + bw, llegaríamos efectivamente al borde **derecho**.
2. Al hacer bw // 2, estamos calculando la **mitad del ancho**.
3. Al sumar esa mitad a la posición inicial x, nos desplazamos desde la izquierda hasta el **centro horizontal**.

 Desglose del Eje Vertical

- **y**: Es la coordenada del borde **superior** (el punto más alto de la persona).
- **bh** (*Bounding Height*): Es la **altura total** del rectángulo.
- **y + bh**: Es el borde **inferior** (la base del rectángulo).

- **Resultado**: Obtenemos las coordenadas (x, y) que marcan el punto medio exacto de la silueta del usuario.

3. Cálculo de la Distancia al Centro (dist)

Una vez que sabemos dónde está el usuario (cx, cy), queremos saber qué tan lejos está del centro de la pantalla (w/2, h/2). Para esto usamos el **Teorema de Pitágoras** (Distancia Euclidiana):

dist = np.sqrt((cx - w/2)**2 + (cy – h/2)**2)

- **(cx - w/2)^2**: Diferencia horizontal al cuadrado.
- **(cy - h/2)^2**: Diferencia vertical al cuadrado.
- **np.sqrt**: La raíz cuadrada de la suma nos da la distancia en línea recta (la hipotenusa).

Selección del Sujeto Principal (Best Contour Selection)

Una vez que tenemos la lista de scored_contours llena de datos, el algoritmo debe quedarse con el candidato más probable de ser el usuario.

1. El Segundo Seguro de Fallos

if not scored_contours:

return np.zeros((h, w), dtype=np.float32)

- **Qué hace:** Si después de aplicar todos los filtros de área, aspecto y bordes, **ningún** objeto sobrevivió, el sistema asume que no hay nadie frente a la cámara.
- **Resultado:** Devuelve una máscara completamente negra. Esto es mucho mejor que dejar que el sistema intente adivinar, ya que evita que el fondo se desenfoque de forma errática.

2. El Ranking de Candidatos (sort)

scored_contours.sort(key=lambda x: (x['score'], x['area']), reverse=True)

best = scored_contours[0]

Esta es una pieza de **lógica de datos** muy elegante:

- **Criterio Doble:** Ordena la lista primero por la **puntuación** (score) y, en caso de empate, por el **área**.
- **reverse=True**: Coloca al mejor candidato (el de mayor puntaje) en la primera posición (índice 0).

- **Resultado:** best ahora contiene toda la información de la persona real, descartando cualquier otro reflejo o ruido.

3. El Diccionario best: Selección de Atributos Críticos y Actualización Global

Es fundamental notar que best no es solo una imagen o un recorte; es un **contenedor de inteligencia** (o "metadata"). Este proviene de la lista scored_contours y contiene toda la "hoja de vida" del mejor candidato tras el proceso de scoring.

Aunque este diccionario almacena múltiples datos (score, área, confianza promedio), en este paso el código realiza una **extracción selectiva** de atributos críticos:

- **self.person_center = best['center']**: Extraemos el punto (x, y) del centroide. Es equivalente a poner un "pin" en el centro de la persona para monitorizar su ubicación exacta.
- **self.person_bbox = best['bbox']**: Extraemos las dimensiones del rectángulo delimitador. Esto permite al sistema estimar la proximidad del usuario a la cámara y su escala dentro del frame.

Importancia del Estado Global: Al guardar estos valores, la clase "recuerda" al sujeto. Si en el siguiente frame la persona aparece repentinamente a una distancia físicamente imposible (ej. 500 píxeles), el sistema detecta la anomalía y puede aplicar correcciones de estabilidad temporal.

4. Creación de la Máscara Filtrada (Ground Truth)

En este punto, el sistema genera la **Máscara Binaria Perfecta**. Es el paso donde se descarta definitivamente el ruido y se consolida la figura del usuario.

- **El Lienzo (np.zeros)**: Se inicializa un lienzo totalmente negro (valor **0.0**), que representa el "fondo" o área a ignorar.
- **El Relleno (1.0)**: Se utiliza cv2.drawContours con un grosor de -1 (relleno sólido) para dibujar la silueta de la persona detectada.
- En la función cv2.drawContours(filtered, [best['contour']], -1, 1.0, -1), el **último -1** es el parámetro thickness (grosor).
- **La "Persona Blanca"**: El resultado es una imagen donde el sujeto es un bloque sólido de color blanco puro y todo lo demás (ruido, sillas, cuadros, otras personas lejanas) es negro absoluto.

□ Recuperación de Segmentos Adyacentes (Hand & Object Recovery)

Este bloque de código se encarga de "reunificar" el cuerpo del usuario si la IA lo ha fragmentado.

1. El Bucle de Compañeros (for other in scored_contours[1:])

- **scored_contours[1:]**: Ya elegimos al "mejor" (índice 0, el cuerpo principal). Ahora revisamos a todos los demás candidatos que quedaron en la lista para ver si alguno es en realidad parte del usuario.

2. La Distancia de Conectividad

dist = np.sqrt((other['center'][0] - best['center'][0])**2 + ...)

- El sistema mide la distancia entre el centro del cuerpo principal y el centro del objeto flotante.

3. El Umbral de Proximidad Dinámico

if dist < max(best['bbox'][2], best['bbox'][3]) * 0.5 and other['confidence'] > 0.3:

Aquí aplicas una regla física:

- **`max(best['bbox'][2], best['bbox'][3]) * 0.5`**: El sistema calcula el tamaño de la persona (el lado más largo de su rectángulo) y lo divide por dos. **`max(...)`**: Selecciona la dimensión más grande de la persona. Si estás de pie, será tu **altura**; si extiendes los brazos a los lados, podría ser tu **ancho**.

 Anatomía de best['bbox']

Cuando OpenCV genera un boundingRect, entrega una lista o tupla con 4 valores en este orden exacto: (x, y, w, h). Por lo tanto:

- **best['bbox'][0]**: Posición **X** (Izquierda).
- **best['bbox'][1]**: Posición **Y** (Arriba).
- **best['bbox'][2]**: Representa el **Ancho** (bw).
- **best['bbox'][3]**: Representa la **Altura** (bh).

- **La Regla:** "Si este objeto extraño está lo suficientemente cerca de mi cuerpo (dentro del radio de mi propio tamaño) y tiene una confianza decente (> 30%), entonces **es parte de mí**".

4. La Fusión de Máscaras

cv2.drawContours(filtered, [other['contour']], -1, 1.0, -1)

Si el objeto pasa la prueba, se dibuja en la misma máscara blanca. Ahora tu "Escudo de Confianza" protege el cuerpo **y** la mano levantada.

5. La Operación Final: Máscara de Atenuación (Bitwise-AND)

La línea return mask * filtered no es una simple multiplicación; es la ejecución de una **Máscara de Atenuación** que actúa como un filtro físico sobre los datos crudos de la IA.

- **mask (La entrada de la IA):** Es la máscara de confianza original de MediaPipe. Aunque es precisa, suele contener "fantasmas" o ruido (pequeñas detecciones de objetos en el fondo que la IA confundió con personas).
- **filtered (Tu filtro geométrico):** Es la silueta sólida blanca (1.0) que construiste basándote en el "mejor contorno" y sus partes adyacentes.

 Método: _get_bbox (Extracción del Bounding Box Principal)

Este método es una función de utilidad que extrae las coordenadas espaciales definitivas del usuario. Se utiliza para actualizar el estado del sistema y permitir que otros módulos (como el de estabilidad) sepan exactamente qué área ocupa la persona.

1. Preparación de Datos: Escalamiento a 8 bits

mask_uint8 = (mask * 255).astype(np.uint8)

- **Por qué es necesario:** La máscara con la que trabajamos en Data Science suele estar en rango [0.0, 1.0] (flotantes). Sin embargo, las funciones de Visión Artificial de OpenCV para encontrar contornos requieren imágenes de **8 bits** (rango 0 a 255).
- **Operación:** Multiplicamos cada píxel por 255 para convertir el "blanco" de 1.0 a 255.

2. Selección del "Líder" (max por área)

largest = max(contours, key=cv2.contourArea)

- **Diferencia con el anterior:** Aquí el código es más directo. En lugar de puntuar a todos, simplemente utiliza la función max de Python para identificar el contorno que tiene la mayor superficie física (cv2.contourArea).
- **Lógica:** Se asume que, tras los filtros previos, el contorno más grande es indiscutiblemente el usuario principal.

3. Generación del Diccionario de Coordenadas

x, y, w, h = cv2.boundingRect(largest)
return {'x': x, 'y': y, 'w': w, 'h': h}

- **Encapsulamiento:** En lugar de devolver variables sueltas, el método devuelve un **diccionario**. Esto es una buena práctica de programación porque permite acceder a los datos de forma semántica (por ejemplo, bbox['w']) en lugar de recordar posiciones en una lista.

 Funcionamiento de cv2.boundingRect(largest)

Cuando ya hemos identificado el contorno más grande (largest), se lo pasamos a esta función. Su trabajo no es filtrar valores de una lista, sino **calcular los límites geográficos** de esa mancha blanca.

Imagina que el contorno de la persona es una forma irregular (con curvas y picos). cv2.boundingRect lo que hace es buscar:

1. El punto que esté más a la **izquierda** (x).
2. El punto que esté más **arriba** (y).
3. Calcula cuántos píxeles hay desde el más a la izquierda hasta el más a la **derecha** (w).
4. Calcula cuántos píxeles hay desde el más arriba hasta el más **abajo** (h).

La Desempaquetación (Unpacking)

x, y, w, h = cv2.boundingRect(largest)

En Python, esto se llama **secuencia de desempaquetado**. La función devuelve una tupla de 4 valores y nosotros los asignamos directamente a 4 variables individuales en una sola línea.

 Método: _extend_shoulders (Extensión y Difuminado Lateral)

El objetivo de este método es añadir "píxeles de seguridad" a los lados de los hombros y crear un degradado suave. Esto evita que el recorte parezca una calcomanía pegada sobre el fondo.

1. Definición de la "Zona de Hombros"

shoulder_top = bbox['y'] + int(bbox['h'] * 0.18)

shoulder_bottom = min(bbox['y'] + int(bbox['h'] * 0.68), h)

El sistema no extiende toda la silueta (no queremos extender la cabeza ni los pies). Utiliza las **guías** del Bounding Box para identificar la región anatómica correcta:

- **Inicio (Top):** Baja un 18% desde la parte superior de la cabeza (zona del cuello/trapecio).
- **Fin (Bottom):** Baja hasta el 68% de la altura total (zona del torso medio).

bbox['y'] es la coordenada vertical (en píxeles) del punto superior del Bounding Box, es decir, la posición en la imagen donde empieza la caja en la parte superior (generalmente cerca del cuello o la cabeza, dependiendo del recorte).
bbox['h'] es la altura total de esa caja, es decir, la distancia en píxeles desde la coordenada bbox['y'] hasta el punto inferior de la caja.

2. Localización de los Límites Reales Escaneo Horizontal (np.where)

Una vez delimitada la zona vertical (entre el 18% y el 68% de la altura), el sistema ejecuta un proceso de **segmentación por filas**:

nonzero = np.where(mask[row, :] > 0.5)[0]

left, right = nonzero[0], nonzero[-1]

- **mask[row, :]**: El sistema toma una "rebanada" horizontal de la imagen (una sola fila de píxeles).
- **np.where(... > 0.5)**: Identifica todos los índices de esa fila donde la IA dice "aquí hay una persona".
- **left y right**: Captura el primer y el último índice. Esto define los bordes exactos de la silueta en esa fila específica. Es el punto de partida desde donde empezaremos a "estirar" los hombros hacia afuera.

3. Generación del Degradado (Alpha Gradient)

Aquí es donde ocurre la magia del **Data Science aplicado al diseño visual**:

extended[row, right + i] = max(0, 1.0 - i * 0.05)

Esta operación realiza un proceso de **Calado (Feathering)** manual, píxel por píxel, que transforma un borde digital rígido en una transición óptica natural.

Análisis del Gradiente:

- **No es un bloque sólido:** En lugar de expandir la máscara con un color blanco puro (1.0), el código resta un **5% de opacidad (0.05)** por cada píxel que se aleja del cuerpo.
- **Simulación de Lente Real:**
    - **Paso 1 (i=0):** El borde original mantiene el valor **1.0** (opacidad total).
    - **Paso 2 (i=1):** El primer píxel de extensión tiene **0.95**.

- **Paso 20 (i=20):** El valor llega a **0.0**, desapareciendo por completo.

- **Asimetría Natural:** El sistema utiliza valores diferenciados para cada lado (self.right_extend = 25px y self.left_extend = 15px). Esta asimetría rompe la rigidez geométrica, logrando un aspecto más orgánico y menos procesado.

El Efecto Visual:

Al aplicar este desvanecimiento (*fade-out*) artificial, engañamos al ojo humano para que perciba una **transición óptica suave**, emulando la forma en que las lentes profesionales (réflex o mirrorless) capturan los bordes de la ropa o el cabello, los cuales nunca son perfectamente afilados en la realidad. El resultado es que la persona no parece "recortada", sino integrada naturalmente en el desenfoque del fondo.

 Método: _create_mask (Procesamiento Multizona y Refinamiento)

Este método es el núcleo del post-procesamiento. Transforma la salida probabilística de la IA en una máscara final de alta fidelidad utilizando una estrategia de **"divide y vencerás"**.

 El Origen del Dato: raw_mask

Antes de entrar al método, el sistema ejecuta el motor de **MediaPipe (Google)**. El archivo .tflite genera lo que llamamos **raw_mask**.

- **Naturaleza:** Es un **Mapa de Confianza**. No es una imagen de blanco y negro, sino una matriz de intensidades (0.0 a 1.0).

- **Función:** Cada píxel representa la probabilidad estadística de que ese punto pertenezca a una persona. Por ejemplo, el centro del rostro suele tener un valor de 0.98, mientras que los bordes del cabello pueden variar entre 0.4 y 0.6.

 Fase 1: Binarización y Purificación de Datos

Antes de segmentar por zonas, el sistema realiza una limpieza profunda para eliminar el ruido ambiental.

1. Binarización Sensible (edge_threshold):

Mediante la operación binary = (raw_mask > self.edge_threshold), el sistema aplica un umbral muy bajo (3% de confianza). Esto permite capturar detalles extremadamente finos, como hilos de ropa o mechones de cabello, que una segmentación estándar descartaría.

2. Filtrado Geométrico:

La máscara binarizada se envía a _filter_false_positives. Aquí se aplican las reglas de área, relación de aspecto y distancia al centro para asegurar que solo el Sujeto de Interés permanezca en la escena.

3. **El Seguro de Silencio (Zero-Sum Check):**

```
if filtered.sum() == 0:
   return np.zeros((h, w), dtype=np.float32)
```

4. Si tras el filtrado la suma de la matriz es cero, el sistema detecta que no hay nadie frente a la cámara. Devolver un lienzo nulo inmediatamente es una técnica de **computación eficiente** que evita procesar cálculos matemáticos innecesarios en un frame vacío.

 Fase 2: Tripartición Anatómica (Head, Shoulder, Body)

Una vez purificada la máscara mediante el filtrado de falsos positivos, el sistema utiliza las dimensiones del Bounding Box para fragmentar la silueta. Este proceso no es un recorte visual, sino una **segmentación de la matriz de datos** utilizando la técnica de *Slicing* de NumPy.

Para la computadora, la imagen es una cuadrícula de coordenadas y el código actúa como una guillotina que corta en dos direcciones para crear tres "rebanadas" horizontales:

- **mask_h (Cabeza):** Definida como [:head_end, :]. Captura desde el píxel 0 (origen) hasta el **28%** de la altura.
- **mask_s (Hombros):** Definida como [head_end:shoulder_end, :]. Cubre la franja entre el **28% y el 52%**.
- **mask_b (Cuerpo/Torso):** Definida como [shoulder_end:, :]. El espacio en blanco tras los dos puntos indica que el sistema debe tomar todo lo que quede hasta la última fila de la imagen (**100%**).

  Nota sobre la Inclusividad Horizontal: Es crucial notar el uso del operador : en la segunda dimensión de cada corte (ej. [:, :]). Estos dos puntos solitarios garantizan que el "escáner" cubra el **100% del ancho del video**. Esto permite que, aunque el usuario se incline lateralmente o mueva los brazos hacia los extremos del frame, la silueta nunca quede fuera del área de procesamiento.

 Fase 3: Dilatación Diferencial y Reensamblaje

La innovación de este método radica en que, gracias a que tenemos las tres regiones aisladas en memorias diferentes, podemos aplicar **Morfología Matemática diferenciada**:

1. **Dilatación por Región:**
   - **Cabeza:** Se aplica una dilatación sutil (kernel pequeño) para preservar la forma orgánica del cabello y evitar el efecto de "casco".

- **Hombros:** Se utiliza una dilatación agresiva y predominantemente horizontal para asegurar que el recorte no "muerda" los bordes de la ropa al movernos.
- **Cuerpo:** Un balance intermedio que otorga estabilidad estructural al torso.

2. Fusión de Capas (np.maximum.reduce):

Tras procesar cada zona por separado, el sistema las vuelve a unir. Al usar la función maximum.reduce, NumPy compara los píxeles de las tres capas y elige el valor más alto. Esto genera una unión perfecta y sin costuras en las fronteras donde antes aplicamos la guillotina del slicing.

3. **Toques Finales de Calidad:**
    - **Extensión de Hombros:** Se invoca a _extend_shoulders para inyectar el degradado asimétrico y el *Feathering*.
    - **Clausura Morfológica (MORPH_CLOSE):** Se aplica un kernel de 11 * 11 que actúa como un "sellador", rellenando cualquier poro o hueco interno para garantizar una máscara sólida y profesional.

 Método: _calculate_quality (Scoring Multivariable)

Este método actúa como el **"Juez de Calidad"** del sistema. En Data Science, esto se conoce como una **Función de Scoring Multi-objetivo**. Su trabajo no es crear la máscara, sino someterla a un examen matemático para devolver un valor entre **0.0 y 1.0**, donde 1.0 representa una detección perfecta.

El algoritmo analiza tres dimensiones críticas mediante una **Media Ponderada**:

1. Confianza Estadística (confidence)

- **La Lógica:** confidence = raw_mask[binary].mean()
- **Qué mide:** La certeza promedio de la red neuronal. Extrae el promedio de probabilidad de los píxeles que la IA identificó como persona.
- **Significado:** Si la IA tiene una confianza del 95% (0.95), la detección es sólida. Si hay mala iluminación o desenfoque, este valor baja, alertando al sistema de que la máscara podría ser un "falso positivo" o estar degradada.

2. Cobertura de Pantalla (coverage)

- **La Lógica:** min(coverage / 0.25, 1.0)
- **El Benchmark (0.25):** El sistema utiliza un estándar de composición: en una videollamada profesional, el usuario debe ocupar aproximadamente el **25%** del encuadre.

- **Significado:** Si el usuario ocupa el 25% o más, obtiene la puntuación máxima en esta categoría. Si la figura es muy pequeña o está lejos (poca cobertura), el puntaje se penaliza para evitar procesar objetos irrelevantes en el fondo.

3. Índice de Contigüidad o Solidez (contiguity)

Para calcular esta métrica, el sistema debe primero "traducir" la información. Es aquí donde ocurre un paso técnico crítico: **La IA (MediaPipe) trabaja en un mundo de Probabilidades, mientras que OpenCV trabaja en un mundo de Píxeles.**

El Traductor de Datos: mask_uint8

Para que OpenCV pueda analizar la geometría, ejecutamos la línea: mask_uint8 = (mask * 255).astype(np.uint8)

Este proceso realiza dos operaciones esenciales:

- **Re-escalado Lineal:** Multiplica cada píxel para que el 1.0 (Persona segura) se convierta en 255 (Blanco puro) y el 0.0 (Fondo) en 0 (Negro).
- **Casteo de Tipo:** Transforma los decimales en números enteros de 8 bits. Mientras que float32 es preciso para cálculos de probabilidad, uint8 es el estándar de la industria para el procesamiento y visualización de imágenes.

La Detección de Siluetas (cv2.findContours)

Una vez obtenida la máscara en formato de píxeles, el sistema utiliza la función findContours. Imagina que le das un marcador a la computadora y le pides que dibuje una línea cerrando todos los grupos de píxeles blancos:

- **RETR_EXTERNAL**: Le dice a OpenCV que solo le importan los contornos de **afuera**, ignorando "agujeros" internos para enfocarse en la silueta general.
- **CHAIN_APPROX_SIMPLE**: Optimiza la memoria guardando solo los puntos esenciales (esquinas) de la silueta.

Medición de la Integridad (Lógica de Áreas)

Con los contornos detectados, el sistema aplica un análisis de áreas para determinar qué tan "limpia" es la detección:

1. **Cálculo de la Mancha Principal (largest)**: De todos los objetos detectados (el usuario, una silla, ruido), busca el que tiene el área más grande. En el 99% de los casos, este es el usuario.
2. **Suma de Todo lo Detectado (total)**: Suma el área de absolutamente todos los contornos encontrados, incluyendo el ruido y manchas pequeñas.
3. **El Coeficiente de Contigüidad**: Se obtiene mediante la división largest / total.

- **Valor 1.0**: Significa que la mancha principal y el total son iguales. Solo hay un objeto sólido y único. La máscara es perfecta.
- **Valor bajo (ej. 0.5)**: Indica que la máscara está "rota" o fragmentada; la mitad de lo detectado es ruido disperso. El sistema utiliza esta métrica para priorizar siluetas íntegras y descartar errores visuales.

¿Qué estamos comparando realmente?

Estamos midiendo qué tan **dominante** es el cuerpo del usuario frente al ruido del fondo:

Si el resultado es 1.0 (100%): Significa que el contorno más grande es el **único** que existe. No hay ruido. La "Contigüidad" es perfecta porque toda la "masa" de la máscara está concentrada en un solo bloque.

- **Si el resultado es 0.7 (70%):** Significa que el 70% del área blanca es tu cuerpo, pero hay un 30% de "basura visual" o fragmentos flotando alrededor.

El Cálculo Final: Ecuación de Ponderación

El sistema fusiona estos tres pilares mediante pesos específicos que definen la jerarquía de calidad:

Score = (0.35 * Confianza) + (0.35 * Cobertura) + (0.30 * Contigüidad)

Peso	Atributo	Importancia Técnica
35%	**Confianza**	Evita "alucinaciones" de la IA en fondos complejos.
35%	**Cobertura**	Asegura que el usuario sea el protagonista del encuadre.
30%	**Contigüidad**	Penaliza el ruido y garantiza una silueta sólida.

Refinamiento Estético y Generación de Canal Alpha

El método _create_alpha actúa como el **"Maquillaje de Alta Fidelidad"** del sistema. Su objetivo es transformar la máscara binaria (que es rígida y toscas) en un **Canal Alpha** profesional, logrando transiciones suaves similares a las de una cámara DSLR.

1. Preparación y Gestión de Memoria

Al inicio del proceso, el sistema define las dimensiones y crea una instancia de trabajo:

```
h, w = mask.shape
alpha = mask.copy()
```

- **Estrategia de Inmutabilidad:** Al ejecutar .copy(), el sistema guarda la máscara original en una "caja de cristal" y entrega una copia idéntica para aplicar los filtros. Esto protege el molde original para comparaciones futuras y evita que los errores de procesamiento se propaguen de forma acumulativa.

---

2. El Concepto: ¿Qué es un Filtro Guiado (Guided Filter)?

Imagina que tienes un recorte hecho con una tijera sin filo (la máscara de la IA) y necesitas que se ajuste perfectamente a los detalles de una fotografía real (el frame). El **Filtro Guiado** toma la máscara pixelada y la imagen original como **"Guía"** para transferir los detalles finos de la foto a la máscara.

 Parámetros Clave del Algoritmo:

- **guide**: Es la imagen original convertida a escala de grises. Funciona como una "plantilla" que le indica al filtro dónde residen los bordes reales (cabello, fibras de ropa, piel).
- **alpha_8**: Es la máscara de trabajo re-escalada a formato de 8 bits (0-255).
- **radius**: Define el radio de búsqueda. Controla qué tan grande es el área que el filtro analiza para suavizar el borde.
- **eps (Epsilon)**: Define la sensibilidad al contraste. Un valor de 100 permite ignorar el ruido digital de la cámara y concentrarse exclusivamente en los bordes estructurales.

---

3. Ejecución y Normalización de Precisión

La línea de procesamiento principal ejecuta tres acciones críticas en una sola operación:

alpha = cv2.ximgproc.guidedFilter(guide, alpha_8, radius, eps).astype(np.float32) / 255.0

`ximgproc` significa **Extended Image Processing**. El Filtro Guiado es un algoritmo tan sofisticado que no está en el paquete básico; se encuentra en este módulo de procesamiento avanzado de imágenes. El **guidedFilter** es un filtro que preserva bordes.

1. **Refinamiento Espacial:** El algoritmo busca bordes en la imagen guide que coincidan con la máscara. Si detecta texturas de alta frecuencia (como mechones de pelo) que la IA ignoró, el filtro ajusta la máscara automáticamente para cubrirlos con precisión quirúrgica.
2. **Conversión a Alta Precisión (astype(np.float32))**: Se transforman los números enteros (0-255) en números decimales. Un Canal Alpha de calidad profesional requiere gradientes suaves (ej. 0.557) que no pueden representarse con números enteros, permitiendo transparencias parciales.
3. **Normalización Matemática (/ 255.0)**: Se realiza la operación inversa al escalado inicial. Al dividir por 255, el rango numérico regresa al estándar de probabilidad de **0.0 (totalmente transparente)** a **1.0 (totalmente opaco)**.

 ¿Qué es el Joint Bilateral Filter?

cv2.ximgproc.**jointBilateralFilter()**

Es un filtro de suavizado inteligente. A diferencia de un desenfoque normal que mezcla todo, este filtro decide qué píxeles suavizar basándose en dos cosas:

1. **La distancia física** (píxeles cercanos).
2. **La similitud de color** (píxeles que se parecen en tono).

Se llama "**Joint**" (Conjunto) porque usa el **frame original** para saber dónde hay cambios de color y aplica ese conocimiento para limpiar la **máscara**.

 Desglose de los Parámetros

- **frame**: Es la imagen original a todo color. El filtro la mira para entender la textura de la piel y la ropa.
- **alpha_8**: Es nuestra máscara (0-255). Es lo que el filtro va a "limpiar".
- **d (Diámetro)**: Determina el tamaño de la vecindad de píxeles que se consideran para el filtro. Proviene de edge_refinement en tus ajustes.
- **sigmaColor=20**: Define qué tanto debe parecerse el color de dos píxeles para que se suavicen entre sí. Un valor de 20 es ideal para ignorar el ruido de la piel pero mantener la separación con el fondo.
- **sigmaSpace=20**: Define qué tan lejos puede estar un píxel de otro para influir en el suavizado.

 ¿Cuál es el objetivo final de esta línea?

El objetivo es eliminar el **"ruido de sal y pimienta"** (esos puntitos blancos o negros que a veces quedan dentro de la silueta).

 Lógica de Protección de Cuerpo (Body Protection)

El código utiliza **Indexación Booleana** de NumPy para encontrar píxeles específicos que cumplen dos condiciones al mismo tiempo y forzar su valor de opacidad (alpha).

1. La Zona de Máxima Seguridad (100% Opaco)

alpha[(raw_mask > 0.5) & (mask > 0.3)] = 1.0

- **Lógica:** Si la IA original está muy segura (raw_mask > 0.5) Y la máscara filtrada confirma que hay alguien ahí (mask > 0.3), no hay duda.
- **Resultado:** El píxel se vuelve **1.0 (completamente sólido)**. Esto garantiza que el centro de tu cara o pecho nunca se desenfoquen por error.

2. Las Capas de Refuerzo Degradado

Las siguientes tres líneas usan np.maximum. Esto es vital: **"Solo aumenta la opacidad, nunca la bajes"**.

- Capa 0.98: alpha[...] = np.maximum(alpha[...], 0.98)

Si la confianza es media-alta (> 0.25), nos aseguramos de que el cuerpo sea casi totalmente opaco (98%).

- Capa 0.92: alpha[...] = np.maximum(alpha[...], 0.92)

Utiliza un umbral personalizado (protection_threshold) para asegurar una opacidad del 92% en zonas donde la IA tiene dudas leves.

- Capa 0.88: alpha[mask > 0.5] = np.maximum(alpha[mask > 0.5], 0.88)

Es la red de seguridad final. Cualquier píxel que esté claramente dentro de la máscara debe tener al menos un 88% de opacidad.

¿Por qué es necesaria esta estructura?

Imagina que estás usando una camisa blanca y el fondo también es claro. Los filtros de suavizado podrían confundirse y hacer que los bordes de tu camisa se vuelvan un poco transparentes, dejando ver el fondo borroso "a través" de ti.

Esta técnica de **"Máximo entre el filtro y el umbral"** garantiza que:

1. Los **bordes externos** (cabello, silueta) sigan siendo suaves gracias a los filtros.
2. El **centro del cuerpo** (masa principal) se mantenga sólido y nítido.

Etapa Final: Lógica de "Feather Edges" y Suavizado Perimetral

Esta es la etapa de **finalización estética**. Una vez que se ha protegido el núcleo del cuerpo para que sea sólido, el sistema necesita que el borde exterior se funda con el fondo de forma natural. Sin este proceso, el recorte parecería pegado artificialmente, un error visual conocido como efecto **"sticker cut-out"**.

1. Creación del "Anillo de Transición"

```
edge_k = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
outer = cv2.dilate(mask, edge_k) - mask
```

- **edge_k**: Crea un pincel pequeño y circular de 3 * 3 píxeles. Se usa una elipse porque es más suave que un cuadrado para formas humanas.
- **cv2.dilate**: Engorda la máscara original ligeramente hacia afuera.
- **La Resta (- mask)**: Aquí ocurre la magia. Al tomar la máscara "engordada" y restarle la "original", nos queda únicamente un **borde delgado** (un anillo de 3 píxeles de ancho) que rodea toda tu silueta.

2. Mezcla de Bordes (Alpha Blending)

```
alpha = np.where(outer > 0, alpha * 0.7 + 0.3 * outer, alpha)
```

- **¿Qué hace?**: Esta línea busca exclusivamente los píxeles que están en ese "anillo" (outer > 0).
- **La Fórmula**: En esos píxeles, mezcla la opacidad actual con un 30% de suavidad extra.
- **Resultado**: Crea una transición degradada. El borde no pasa de "Persona" a "Fondo" de golpe, sino que se desvanece suavemente, permitiendo que un poco del color del fondo se mezcle en los bordes de tu ropa o piel, lo cual es físicamente correcto en la fotografía real.

Esta es la etapa de **finalización estética**. Una vez que hemos protegido el cuerpo para que sea sólido, necesitamos que el borde exterior se funda con el fondo de forma natural. Sin esto, el recorte parecería pegado con pegamento, un efecto conocido como "sticker cut-out".

Aquí tienes el desglose de esta técnica de **suavizado perimetral**:

1. Creación del "Anillo de Transición"

```
edge_k = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
outer = cv2.dilate(mask, edge_k) – mask
```

- **edge_k**: Crea un pincel pequeño y circular de 3 * 3 píxeles. Se usa una elipse porque es más suave que un cuadrado para formas humanas.
- **cv2.dilate**: Engorda la máscara original ligeramente hacia afuera.

- **La Resta (- mask)**: Aquí ocurre la magia. Al tomar la máscara "engordada" y restarle la "original", nos queda únicamente un **borde delgado** (un anillo de 3 píxeles de ancho) que rodea toda tu silueta.

2. Mezcla de Bordes (Alpha Blending)

```
alpha = np.where(outer > 0, alpha * 0.7 + 0.3 * outer, alpha)
```

- **¿Qué hace?**: Esta línea busca exclusivamente los píxeles que están en ese "anillo" (outer > 0).
- **La Fórmula**: En esos píxeles, mezcla la opacidad actual con un 30% de suavidad extra.
- **Resultado**: Crea una transición degradada. El borde no pasa de "Persona" a "Fondo" de golpe, sino que se desvanece suavemente, permitiendo que un poco del color del fondo se mezcle en los bordes de tu ropa o piel, lo cual es físicamente correcto en la fotografía real.

3. El Seguro de Rango (np.clip)

```
return np.clip(alpha, 0, 1)
```

- **Propósito**: Tras aplicar filtros, multiplicaciones y restas, algunos píxeles podrían terminar con valores ilegales (como 1.1 o -0.1).
- **Acción**: clip obliga a que todo el canal alpha esté estrictamente en el rango [0, 1]. Cualquier cosa menor a 0 se vuelve 0, y cualquier cosa mayor a 1 se vuelve 1. Esto evita errores de renderizado (puntos negros o brillantes extraños en el video final).

✨ MÉTODO CREATE_BACKGROUND

1. Opción de Reemplazo (Fondo Virtual)

```
if self.mode == BlurMode.REPLACE and self.background_image is not None:
    return cv2.resize(self.background_image, (w, h))
```

- **Qué hace:** Antes de calcular cualquier desenfoque, verifica si el usuario prefiere **reemplazar** el fondo por completo (como en Zoom o Teams).
- **Lógica:** Si hay una imagen cargada, simplemente la ajusta al tamaño exacto del video (resize) y la devuelve.

2. Simulación de Bokeh: El Corazón del Lente

Para que un desenfoque se vea "profesional" (estilo **Bokeh**), no basta con que la imagen se vea borrosa; los puntos de luz deben expandirse en formas circulares.

```
radius = self.preset.blur_radius
size = radius * 2 + 1
kernel = np.zeros((size, size), np.uint8)
cv2.circle(kernel, (radius, radius), radius, 1, -1)
kernel = kernel.astype(np.float32) / kernel.sum()
```

El Tamaño del "Pincel" (size)

```
radius = self.preset.blur_radius
size = radius * 2 + 1
```

- Cuando más adelante en _create_background se escribe self.preset.blur_radius, el código ya sabe que debe usar 15, 21 o 27; valores del Diccionario PRESETS.
- **¿Por qué * 2 + 1?** Los filtros de imagen (kernels) siempre deben tener un tamaño **impar** (ej. 21x21, 35x35). Esto es para que el filtro tenga un **píxel central exacto**. Si fuera par, el desenfoque se desplazaría un píxel hacia un lado, haciendo que la imagen se vea "movida" en lugar de desenfocada.

Creación del Lienzo Negro (np.zeros)

```
kernel = np.zeros((size, size), np.uint8)
```

- Crea una matriz cuadrada totalmente negra (llena de ceros). Imagina que es un lienzo oscuro donde vamos a dibujar nuestro "agujero" por donde pasará la luz.

Dibujando la Apertura del Lente (cv2.circle)

```
cv2.circle(kernel, (radius, radius), radius, 1, -1)
```

- **¿Qué hace?** Dibuja un **círculo blanco** perfecto en el centro del lienzo negro.
- **(radius, radius)**: Es el centro del círculo.
- **radius**: Es el radio (qué tan grande es el desenfoque).

- **1**: Es el color (blanco).
- **-1**: Indica que el círculo debe estar **relleno**, no solo el borde.
- **Resultado:** Tienes un "disco" de luz. En óptica, esto se llama el **Disco de Airy** o círculo de confusión.

Normalización de Energía (/ kernel.sum())

kernel = kernel.astype(np.float32) / kernel.sum()

- **Este es el paso más importante.** Si aplicas un filtro lleno de "1", la imagen resultante sería miles de veces más brillante y se vería blanca.
- **¿Cómo funciona?** Al dividir cada píxel por la suma total de todos los píxeles del círculo, aseguras que el promedio de luz sea exactamente **1**.
- **Efecto:** La luz se "reparte". Un punto brillante en el fondo se distribuye en forma de círculo, pero la cantidad total de luz en la escena sigue siendo la misma.

3. Aplicación del Efecto y Suavizado Final

bokeh = cv2.filter2D(frame, -1, kernel)

return cv2.GaussianBlur(bokeh, (35, 35), 0)

- **cv2.filter2D**: Aquí es donde ocurre la magia. El sistema recorre cada píxel del fondo y lo expande usando el círculo que creamos antes. Esto hace que las luces del fondo se conviertan en "discos" de luz, que es la firma visual de un lente caro.
- **`cv2.GaussianBlur`**: Finalmente, se aplica un ligero desenfoque Gaussiano extra. Esto sirve para suavizar los bordes de esos discos y que el fondo no distraiga al espectador, logrando una textura de "mantequilla" muy suave.

□ Método: Estabilidad Temporal mediante Flujo Óptico y Deformación de Máscara

¿Qué es el Flujo Óptico?

Imagina que lanzas un puñado de confeti frente a ti y te mueves; el confeti sigue la dirección de tu movimiento. El **Flujo Óptico** es un algoritmo que analiza dos cuadros consecutivos y calcula hacia dónde se movió cada píxel.

Estas líneas representan la **fase de inicialización y preparación** del sistema de detección de movimiento. Es el punto de partida para que el algoritmo pueda comparar el "antes" y el "después".

Aquí tienes el desglose paso a paso:

Conversión a Escala de Grises

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

- **¿Qué hace?**: Toma el cuadro actual (que viene en color BGR) y lo transforma a blanco y negro (escala de grises).
- **¿Por qué es necesario?**: Para detectar movimiento o flujo óptico, no necesitamos saber si tu camisa es roja o azul. Solo necesitamos ver los cambios en la **intensidad de la luz** (brillo). Además, procesar un solo canal (gris) es **3 veces más rápido** que procesar tres canales (color), lo que ahorra mucha memoria y CPU.

El "Primer Cuadro" (Condición Inicial)

```
if self.prev_frame_gray is None:
    self.prev_frame_gray = gray
    return True, None
```

- **self.prev_frame_gray is None**: Esta condición solo se cumple una vez: en el **primerísimo cuadro** del video. En ese momento, el sistema no tiene nada con qué comparar (no hay un "pasado").
- **self.prev_frame_gray = gray**: El código guarda el cuadro actual como el "cuadro anterior". A partir del siguiente segundo, este será el punto de referencia.
- **return True, None**:
    - True: Le dice al sistema que considere que "hay movimiento" (para que la IA se active por primera vez).
    - None: Indica que todavía no hay un mapa de movimiento (flow), porque para calcular movimiento se necesitan al menos **dos** fotos, y apenas tenemos la primera.

La Resta Absoluta (cv2.absdiff)

```
diff = cv2.absdiff(gray, self.prev_frame_gray)
```

- **¿Qué hace?**: Resta el valor de cada píxel del cuadro actual (gray) del cuadro anterior (prev_frame_gray).
- **¿Por qué "abs" (absoluto)?**: Porque no nos importa si un píxel se volvió más oscuro o más claro, solo nos importa **cuánto cambió**. El valor absoluto asegura que la diferencia siempre sea un número positivo.
- **Resultado**: Crea una nueva imagen donde todo lo que se quedó quieto se ve **negro** (diferencia 0) y todo lo que se movió se ve como una **silueta brillante**.

El Cálculo del Umbral (has_motion)

has_motion = np.mean(diff) / 255.0 > self.motion_threshold

Aquí es donde el código toma la decisión lógica basándose en estadísticas:

- **np.mean(diff)**: Calcula el promedio de brillo de toda la imagen de diferencia. Si hay mucho movimiento, el promedio será alto; si no hay nada, será cercano a cero.
- **/ 255.0**: Normaliza el valor. Como los píxeles van de 0 a 255, al dividir por 255 obtenemos un número entre **0.0 y 1.0** (un porcentaje de cambio).
- **> self.motion_threshold**: Compara ese porcentaje con el límite que definiste al principio (en tu código es 0.02, es decir, un 2% de cambio).
- **Resultado**:
    - Si el cambio es mayor al 2%, has_motion es **True** (¡A trabajar! Hay que recalcular la máscara).
    - Si es menor, es **False** (Podemos ahorrar energía, el usuario está quieto).

La Condición de Activación

if has_motion and self.prev_mask is not None:

El sistema solo trabaja si se cumplen dos cosas:

1. **has_motion**: Confirmamos que algo se movió (para no gastar CPU innecesariamente).
2. **self.prev_mask is not None**: Necesitamos tener una máscara previa para poder "empujarla" hacia la nueva posición.

El Algoritmo de Farneback (cv2.calcOpticalFlowFarneback)

Esta función calcula el **Flujo Óptico Denso**. A diferencia de otros métodos que solo siguen puntos sueltos, este analiza **cada píxel de la imagen**.

- **self.prev_frame_gray, gray**: Compara la foto anterior con la actual (en gris).
- **Los números técnicos (0.5, 3, 15...)**: Son los parámetros de la "cámara virtual":
    - **0.5 (Escala)**: Crea una pirámide de imágenes para detectar movimientos grandes y pequeños.
    - **15 (Winsize)**: Es el tamaño de la ventana de búsqueda. Un valor de 15 es ideal para detectar movimientos humanos normales (como mover las manos al hablar).

¿Qué es exactamente el flow resultante?

El resultado no es una imagen normal, es una matriz de **vectores de movimiento**. Cada píxel del flow contiene dos valores:

- **Valor X**: Cuántos píxeles se movió ese punto horizontalmente.
- **Valor Y**: Cuántos píxeles se movió verticalmente.

Ejemplo: Si mueves tu cabeza 10 píxeles a la derecha, el flow en esa zona dirá [10, 0].

Actualización de la Memoria Temporal

self.prev_frame_gray = gray

Esto se parece mucho más a un **ffill (forward fill)**, pero con una diferencia clave en el momento en que ocurre.

Aquí te explico la analogía técnica:

¿Por qué es como un ffill?

En Pandas, ffill toma un valor y lo propaga hacia la siguiente posición vacía. Aquí, self.prev_frame_gray = gray está haciendo una **propagación de estado**:

- Tomas el valor **presente** (gray).
- Lo "empujas" hacia la variable que representará el **pasado** (prev_frame_gray) en el siguiente ciclo.

- Esto asegura que la referencia siempre esté un paso por delante, manteniendo la cadena de comparación unida.

** Representación en Columna (Ventana Deslizante)**

Imagina que tenemos una columna llamada Imagen_Cámara. El código genera dos variables que se van "rellenando" así:

Frame (Tiempo)	**Entrada (Cámara)**	**gray (Presente)**	**self.prev_frame_gray (Pasado)**	**Operación**
t = 0	Foto A	**Foto A**	None	*Inicialización*
t = 1	Foto B	**Foto B**	**Foto A**	Comparar B vs A
t = 2	Foto C	**Foto C**	**Foto B**	Comparar C vs B
t = 3	Foto D	**Foto D**	**Foto C**	Comparar D vs C

** ¿Cómo ocurre el movimiento? (El efecto "ffill")**

Si lo ves como una sola columna que se desplaza hacia abajo:

1. **Captura:** El frame nuevo llega a la celda gray.
2. **Cálculo:** Se hace la resta contra la celda de abajo (prev_frame_gray).
3. **Desplazamiento:** La línea self.prev_frame_gray = gray actúa como un **copiar y pegar** el valor de arriba hacia abajo para que esté listo en el siguiente segundo.

- **¿Qué hace?**: Guarda el cuadro actual (que ya procesamos) en la variable de "cuadro anterior".
- **¿Por qué es vital?**: Para que el ciclo continúe. En el próximo frame que entre de tu cámara, este cuadro gray será el pasado.
- **Importancia**: Sin esta línea, el sistema siempre se compararía contra la primera foto que sacó al abrir la aplicación, y el detector de movimiento se volvería loco porque pensaría que te estás moviendo constantemente respecto al inicio.

Entrega de Datos al "Cerebro" del Programa

return has_motion, flow

Aquí el método termina y le devuelve dos piezas de información cruciales al resto del código:

- **has_motion (Booleano - True/False)**: Es la señal de alerta. Si es True, el programa sabe que debe aplicar efectos de movimiento.
- **flow (Matriz de vectores o None)**: Es el "mapa de navegación". Si hubo movimiento, entrega las coordenadas de hacia dónde se movió cada píxel. Si no hubo movimiento, entrega None (nada) para ahorrar memoria.

EL MÉTODO _WARP_MASK (Deformación de Máscara)

Si el **Flujo Óptico** es el mapa que dice hacia dónde se movió la gente, el método _warp_mask es el que **mueve los objetos** siguiendo ese mapa.

```
def _warp_mask(self, mask: np.ndarray, flow: np.ndarray) -> np.ndarray:
    h, w = mask.shape
    x, y = np.meshgrid(np.arange(w), np.arange(h))
    new_x = (x + flow[:, :, 0]).astype(np.float32)
    new_y = (y + flow[:, :, 1]).astype(np.float32)
    return cv2.remap(mask, new_x, new_y, cv2.INTER_LINEAR,
                     borderMode=cv2.BORDER_REPLICATE)
```

¿Qué está pasando aquí? (Paso a paso)

1. **np.meshgrid (La Red)**: Crea una cuadrícula de coordenadas de toda la imagen. Imagina que pones una red de pescador invisible sobre tu máscara.
2. **new_x y new_y (El Empuje)**: Aquí se suma el movimiento.
    - Si el flow dice que tu cabeza se movió +5 en X, la red se estira +5 en esa zona.
    - **Es como si la máscara fuera de gelatina** y el flujo óptico la empujara hacia donde tú te moviste.
3. **cv2.remap (El Dibujo)**: Toma la máscara original y la "proyecta" sobre la nueva red deformada.

La cuadrícula base (x e y)

La Red np.meshgrid creó dos matrices, x e y, que guardan las coordenadas originales de cada píxel (ejemplo: el píxel en la columna 10, fila 20 tiene x=10, y=20).

El "Empuje" del Flujo Óptico (flow)

Recuerda que el flow es un mapa de flechas.

- flow[:, :, 0] son las flechas horizontales (eje X).
- flow[:, :, 1] son las flechas verticales (eje Y).

La Estructura de la Matriz flow

La matriz tiene tres dimensiones: (Alto, Ancho, 2).

- La **primera** dimensión es la fila (Y).
- La **segunda** es la columna (X).
- La **tercera** es el "canal" donde guardamos el movimiento.

El significado del índice

En Python, los índices empiezan en 0. Por eso:

- **flow[:, :, 0] (Índice 0):** Representa el movimiento **Horizontal (Eje X)**.
    - Si el valor es **+5**, el píxel se movió a la derecha.
    - Si es **-5**, se movió a la izquierda.
- **flow[:, :, 1] (Índice 1):** Representa el movimiento **Vertical (Eje Y)**.
    - Si el valor es **+5**, el píxel se movió hacia abajo.
    - Si es **-5**, se movió hacia arriba.

¿Por qué es una "Autoreferencia" la tercera dimensión?

Es una autoreferencia porque el valor no te dice *dónde está* el píxel (eso ya lo sabes por la posición [20, 10]), sino que te dice **qué le pasó** a ese píxel específico.

- **Eje X (Índice 0):** Referencia al movimiento lateral.
- **Eje Y (Índice 1):** Referencia al movimiento de arriba/abajo.

¿Por qué se escribe [:, :, 0]?

Esa es la sintaxis de "Slicing" (rebanado) de NumPy:

1. El primer **:** significa: "Toma todas las filas".
2. El segundo **:** significa: "Toma todas las columnas".

3. El **0** o **1** significa: "Toma solo la primera o segunda capa de datos".

La Suma: x + flow

Aquí es donde ocurre la física:

new_x = (x + flow[:, :, 0])

- **¿Qué hace?**: A la posición original del píxel (x), le suma cuánto se movió según el flujo óptico.
- **Ejemplo real**:
    1. Tu nariz estaba en la posición x = 100.
    2. Te moviste rápido a la derecha y el flujo detectó un movimiento de +15 píxeles.
    3. new_x ahora vale 115.
- **Resultado**: El sistema ahora sabe que lo que antes estaba en 100, **ahora pertenece a la posición 115**.

¿Por qué .astype(np.float32)?

- Las coordenadas originales son números enteros (píxel 1, 2, 3...).
- Pero el movimiento calculado por el flujo óptico puede ser decimal (ejemplo: te moviste **1.5 píxeles**).
- Usamos float32 (decimales) para que el movimiento sea **suave**. Si solo usáramos números enteros, la máscara se movería "a saltitos" y se vería vibrante o pixelada. Al usar decimales, OpenCV puede calcular colores intermedios entre píxeles (interpolación), logrando un efecto mucho más fluido y profesional.

Visualización de la "Remapeo" (El Paso Final)

Para redondear el punto de **cv2.remap**, puedes añadir esta pequeña nota que explica cómo se usa esa suma que calculaste:

> **El Toque Maestro: cv2.remap**
>
> Imagina que new_x y new_y son una **orden de traslado**. Cuando ejecutas cv2.remap(mask, new_x, new_y...), le estás diciendo a cada píxel de la máscara vieja: *"Busca tu nuevo hogar en las coordenadas que dicen estas matrices"*.

Gracias al np.float32, si el "nuevo hogar" cae entre dos píxeles, OpenCV hace una **mezcla suave (interpolación lineal)**, lo que evita que los bordes de la máscara se vean como escalones de una escalera.

El Resultado

Gracias a esto, la máscara **te persigue**. Si la IA de segmentación se retrasa unos milisegundos, no importa, porque el sistema ya "empujó" la máscara anterior a tu posición actual usando el Flujo Óptico.

EL MÉTODO _FIRST_PASS

Este método es el **"Explorador"** de tu programa. Su trabajo no es aplicar efectos, sino recorrer todo el video para encontrar el momento donde sales mejor recortado y crear una "máscara maestra" que servirá de guía para el resto del proceso.

Preparación y Medición

```
cap = cv2.VideoCapture(input_path)
w, h = int(cap.get(3)), int(cap.get(4))
total = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
self.accumulated_mask = np.zeros((h, w), dtype=np.float32)
```

- Abre el video y obtiene sus dimensiones (ancho y alto).
- Crea la **accumulated_mask**: Una matriz vacía (negra) del tamaño del video. Aquí es donde se irá "sumando" tu silueta a lo largo de todo el video.

El Contador Visual (tqdm)

```
for _ in tqdm(range(total), desc="Analyzing")
```

- **range(total)**: Crea una lista de números desde el 0 hasta el número total de cuadros que tiene el video.
- **tqdm(...)**: Esta es una librería que crea la **barra de progreso** que ves en la consola.
- **desc="Analyzing"**: Es la etiqueta que aparece a la izquierda de la barra para que sepas qué está haciendo el programa en ese momento.

- **El guion bajo (_)**: Se usa como nombre de variable cuando no necesitas el número del índice (0, 1, 2...) dentro del bucle, solo quieres que el ciclo se repita esa cantidad de veces.

La Captura del Cuadro (cap.read)

ret, frame = cap.read()

Esta función de OpenCV es como dar un paso adelante en el video y sacar una foto. Devuelve dos cosas (una tupla):

- **frame**: Es la imagen real (la matriz de píxeles) del cuadro actual. Es lo que vas a procesar.
- **ret (Booleano: True/False)**: Es una señal de éxito.
    - **True**: "Leí el cuadro correctamente".
    - **False**: "No pude leer nada" (ya sea por un error o porque llegamos al final del archivo).

El Sistema de Seguridad (if not ret)

```
if not ret:
    break
```

- **¿Por qué es necesario?**: A veces el cálculo de total (el número de cuadros) puede fallar ligeramente o el archivo de video puede estar corrupto al final.
- **break**: Si ret es False, esta línea "rompe" el bucle inmediatamente. Evita que el programa intente procesar un cuadro vacío, lo cual causaría un error (crash) en las líneas siguientes.

Todo esto sigue ocurriendo **dentro del bucle for**, justo después de leer cada cuadro. Es la parte donde el programa deja de ser un simple lector de video y se convierte en un sistema de Inteligencia Artificial.

Conversión al formato MediaPipe

mp_img = mp.Image(image_format=mp.ImageFormat.SRGB, data=frame)

- **¿Qué hace?**: Envuelve la imagen de OpenCV (frame) en un objeto compatible con MediaPipe.

- **¿Por qué?**: MediaPipe es muy estricto con los formatos. Necesita saber explícitamente que la imagen es **SRGB** para que su red neuronal pueda "entender" los colores y las formas correctamente.

Extracción de la Máscara de Confianza

raw = segmenter.segment(mp_img).confidence_masks[0].numpy_view()

raw = cv2.resize(raw, (w, h))

- **confidence_masks[0]**: Aquí la IA no te da un "sí o no" (persona o fondo). Te da un mapa de **probabilidades** (de 0.0 a 1.0). Un valor de 0.9 significa "estoy 90% seguro de que esto es una persona".
- **numpy_view()**: Convierte esa información técnica en una matriz de NumPy que podemos manipular.
- **cv2.resize**: La IA suele trabajar internamente con imágenes pequeñas (como 256x256). Esta línea estira esa máscara para que coincida exactamente con el tamaño de tu video original.

Creación y Evaluación de la Máscara

mask = self._create_mask(raw, h, w)

quality = self._calculate_quality(raw, mask)

El llamado a los métodos internos

- **_create_mask**: Limpia la imagen "raw" (cruda). Probablemente aplica un umbral para que lo que era "90% seguro" se convierta en un píxel blanco sólido (1) y el resto en negro (0).
- **_calculate_quality**: Esta es la auditoría. Compara la máscara limpia con los datos originales de la IA para ver si el recorte es estable o si hay demasiado "ruido".

La "Memoria Colectiva" (accumulated_mask)

self.accumulated_mask = np.maximum(self.accumulated_mask, mask)

Esta es la línea que mencionamos antes como una "larga exposición":

- **np.maximum**: Compara la máscara que ya tenía guardada con la nueva. Si el usuario movió el brazo a un lugar nuevo, ese lugar se queda "marcado" para siempre en la máscara acumulada.

- **Resultado**: Al final del video, esta máscara no muestra a una persona, sino **todo el espacio que la persona ocupó** durante el video. Es como el rastro que deja un caracol.

El Concurso de Calidad

if quality > self.best_quality:

- **¿Qué hace?**: Compara la "nota" que acaba de recibir el cuadro actual (calculada por el método _calculate_quality) con la nota más alta registrada hasta ese momento.
- **El inicio**: Como al empezar self.best_quality es **0**, el primer cuadro que se procese siempre ganará por defecto.

Guardando al Ganador

Si el cuadro actual es mejor que el anterior, se ejecutan tres acciones de almacenamiento:

- **self.best_quality = quality**: Actualiza el récord a batir. Ahora los siguientes cuadros tendrán que ser mejores que este para ganar.
- **self.best_mask = mask.copy()**: Guarda una copia de la máscara (el recorte) de ese cuadro. Es vital usar .copy() para que, si la variable mask cambia en el siguiente ciclo del bucle, nuestra "mejor máscara" guardada no se altere.
- **self.best_frame_idx = self.frame_count**: Anota el "número de página" (el índice del cuadro) donde ocurrió este éxito. Esto es útil para saber en qué momento exacto del video el usuario se veía mejor.

El Contador de Tiempo

Esta línea marca el final del bucle “for”

self.frame_count += 1

- **¿Qué hace?**: Independientemente de si el cuadro fue bueno o malo, el contador avanza. Esto le permite al programa saber en qué cuadro está parado en cada momento del bucle for. Sin esto, el best_frame_idx siempre valdría cero.

- **¿Por qué es importante?** Porque cuando el programa encuentra la "mejor máscara", necesita anotar en su libreta: *"La encontré en el frame número 42"*. Sin este contador, sabríamos cuál es la mejor máscara, pero no sabríamos **cuándo** ocurrió.

Estas dos líneas marcan el final de la "misión de exploración" y preparan al sistema para la fase final de renderizado.

1. Libera el recurso del archivo de video

cap.release()

2. Reinicia el contador para la siguiente fase de procesamiento

self.frame_count = 0 # Reset for second pass

¿Qué está pasando exactamente?

 cap.release() (Cerrar el archivo)

Cuando abres un video con cv2.VideoCapture, el sistema operativo pone un "candado" al archivo y reserva una parte de la memoria RAM para manejar ese flujo de datos.

- **¿Qué hace?:** Cierra el archivo de video y libera la memoria utilizada por la cámara o el archivo.
- **¿Por qué es vital?:** Si no liberas el recurso, el archivo podría quedar bloqueado, impidiendo que otros programas (o el tuyo mismo más adelante) lo vuelvan a abrir. Es como colgar el teléfono después de una llamada; si no cuelgas, la línea sigue ocupada.

 self.frame_count = 0 (Reiniciar el cronómetro)

Como bien notaste antes, el frame_count llegó hasta el final (por ejemplo, hasta 115).

- **¿Qué hace?:** Pone el contador de nuevo en cero.
- **¿Por qué es necesario?:** Tu programa usa un sistema de **"Doble Pasada"**.
    1. **Primera pasada:** Analiza (aquí es donde estamos ahora).
    2. **Segunda pasada:** Aplica el efecto de desenfoque (Bokeh) y guarda el resultado.
- Para la segunda pasada, el programa necesita empezar a leer desde el **Frame 0** otra vez para procesar el video desde el inicio. Sin este reset, el programa pensaría que ya terminó.

Esta es la etapa de **"Post-procesamiento Maestro"**. Una vez que el bucle terminó y el video se cerró, el programa toma la mejor máscara que encontró y la "arregla" para que sea casi perfecta antes de empezar la segunda pasada.

El Refuerzo con la Acumulada (0.85)

self.best_mask = np.maximum(self.best_mask, self.accumulated_mask * 0.85)

- **¿Qué hace?**: Mezcla la mejor máscara encontrada (best_mask) con el "rastro" de todo el movimiento del video (accumulated_mask).
- **¿Por qué el 0.85?**: Es un factor de **atenuación**. Al multiplicar la máscara acumulada por 0.85, le estamos diciendo al programa: *"Confío en la zona donde te moviste, pero no tanto como en el frame actual"*.
- **El objetivo**: Esto ayuda a rellenar partes que la IA pudo haber parpadeado (como un hueco en una pierna o un brazo) usando la información de "memoria" del video completo. Crea una silueta más sólida.

El Cierre Morfológico (MORPH_CLOSE)

```
self.best_mask = cv2.morphologyEx(
    self.best_mask, cv2.MORPH_CLOSE,
    cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (13, 13))
)
```

Aquí aplicas una operación de **Morfología Matemática**.

- **MORPH_CLOSE (Cierre)**: Imagina que la máscara tiene "poros" o pequeños puntos negros internos porque la IA falló en un píxel. El cierre primero expande la máscara y luego la contrae.
- **Resultado**: Rellena huecos pequeños y conecta partes que deberían estar juntas.
- **MORPH_ELLIPSE (13, 13)**: Define la forma de la "herramienta" que limpia la máscara. Al ser una elipse de 13x13 píxeles, suaviza los bordes de forma redondeada, haciendo que el recorte de la persona se vea más natural y menos "dentado".

Información Final y Retorno

print(f"\nBest frame: {self.best_frame_idx} (quality={self.best_quality:.3f})")

return w, h

- **print**: Informa al usuario en qué frame se encontró la mejor calidad. Es útil para depurar y saber si el sistema está funcionando bien.
- **return w, h**: Devuelve el ancho y alto del video original. Esto es vital porque el programa principal usará estas medidas para configurar el archivo de video de salida en la segunda pasada.

EL MÉTODO _SECOND_PASS

Esta parte marca el inicio de la **"Fase de Renderizado"**. Si el first_pass fue la investigación, el second_pass es la ejecución: aquí es donde se aplica el efecto de desenfoque y se guarda el resultado final.

 Parámetros de Entrada (6 parámetros)

Este método es más complejo porque necesita toda la información recolectada:

1. **input_path**: El video original.
2. **output_path**: Dónde se guardará el video con el efecto.
3. **segmenter**: La IA de MediaPipe para seguir detectándote.
4. **w y h**: El ancho y alto que medimos en la primera pasada.
5. **diagnose**: Un interruptor (True/False). Si es True, el programa te mostrará ventanas con lo que está pensando la IA en tiempo real.

 Desglose del Código

Reapertura del Video

cap = cv2.VideoCapture(input_path)

- **¿Qué hace?**: Vuelve a abrir el video desde el principio.
- **¿Por qué?**: Como recordaras, en la primera pasada leímos todo el archivo. Ahora necesitamos empezar desde el **Frame 0** para aplicar los efectos cuadro por cuadro.

Extracción de Metadatos

fps = cap.get(cv2.CAP_PROP_FPS)

total = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

- **fps (Frames Per Second)**: Obtiene la velocidad original del video (ej. 30 o 60 cuadros por segundo). Es vital para que el video resultante no se vea "en cámara rápida" o lenta.
- **total**: Reconfirma cuántos cuadros vamos a procesar.

Configuración del Escritor (VideoWriter)

out = cv2.VideoWriter(output_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))

Esta es la línea más importante para el archivo final. Crea el objeto **out**, que es como una grabadora de video:

- **cv2.VideoWriter_fourcc(*'mp4v')**: Define el "códec" (el lenguaje de compresión). mp4v es el estándar para archivos MP4.
- **fps**: Le dice a la grabadora a qué velocidad debe "tocar" el video.
- **(w, h)**: Define el tamaño del lienzo.

 Desglose del Código en Python

Imprime una línea divisoria de 60 signos de igual para separar visualmente las fases

print(f"\n{'='*60}")

Informa que ha comenzado la Fase 2 y qué configuración de calidad se está usando

print(f"PASS 2: Rendering ({self.preset.name} quality)...")

Imprime otra línea para cerrar el encabezado y añade un salto de línea (\n)

print(f"{'='*60}\n")

¿Qué significan estos elementos técnicos?

1. **{'='*60}**: Es un truco de Python para repetir caracteres. En lugar de escribir manualmente sesenta símbolos de =, Python los multiplica. Esto crea una barrera visual clara en la terminal.
2. **\n**: Es el carácter de **"Nueva Línea"** (salto de línea). Se usa para que el texto no aparezca pegado a lo anterior y sea más legible.

3. **self.preset.name**: Aquí el programa accede a la configuración (Preset) que elegiste al principio (por ejemplo: "Balanced", "Performance" o "Quality"). Esto le confirma al usuario bajo qué parámetros se está creando su video.

Bucle “for” principal con barra de progreso

```
for frame_idx in tqdm(range(total), desc="Rendering"):

 Captura del fotograma actual
   ret, frame = cap.read()

   Control de seguridad (fin de video)
   if not ret:
      break

  Inicio del cronómetro de alta precisión
   start = time.perf_counter()
```

¿Qué está pasando exactamente?

1. **for frame_idx in tqdm(...)**: A diferencia de la primera pasada (donde usábamos un guion bajo _), aquí sí usamos **frame_idx**. Necesitamos saber exactamente qué número de cuadro estamos procesando para sincronizarlo con la "máscara maestra" o para aplicar el flujo óptico correctamente.
2. **cap.read() y if not ret**: Es el mismo mecanismo de lectura que vimos antes. La diferencia es que ahora estamos leyendo el video por **segunda vez**. Al haber reiniciado el contador a cero y abierto el archivo de nuevo, cap.read() nos entrega nuevamente el primer frame del video.
3. **time.perf_counter() (El cronómetro de precisión)**: Esta es una parte muy interesante. perf_counter es el cronómetro más preciso que tiene Python (mide en fracciones de microsegundos).
    - **¿Para qué sirve?**: Se pone al principio del bucle para medir cuánto tarda el programa en procesar **un solo frame**.

- Al final del bucle, el programa restará el tiempo final menos este start para decirte: "Oye, estoy procesando a 15 FPS" o "Me tomó 0.05 segundos este cuadro".

Siguendo dentro del bucle for: Detección de Movimiento (El "Radar")

has_motion, flow = self._detect_motion(frame)

Aquí llamas al método que utiliza el algoritmo de **Farneback** que analizamos al principio.

- **flow**: Es la matriz que ya conoces, con sus dos capas (X e Y) que actúan como "etiquetas" de hacia dónde se movió cada píxel.
- **has_motion**: Es un valor Booleano (True o False). El programa calcula si el movimiento promedio es lo suficientemente grande como para tomarlo en cuenta. Si solo hay ruido digital y no te mueves, has_motion será False.

Segmentación Actual (La "Opinión de la IA")

mp_img = mp.Image(image_format=mp.ImageFormat.SRGB, data=frame)

raw = segmenter.segment(mp_img).confidence_masks[0].numpy_view()

raw = cv2.resize(raw, (w, h))

¿Qué es mp.Image()?

mp_img = mp.Image(image_format=mp.ImageFormat.SRGB, data=frame)

Es un **contenedor**.

- **El problema:** OpenCV lee las imágenes en un formato (matrices NumPy en orden de color BGR). MediaPipe, que es la librería de Google, no entiende ese formato directamente.
- **La solución:** mp.Image toma los datos crudos de tu frame y los "empaqueta" en un objeto que MediaPipe puede procesar. Es como traducir un documento al idioma nativo de la IA antes de entregárselo. El formato SRGB le indica que los colores vienen en el estándar normal de video.

¿Se le pide la opinión a la IA otra vez?

raw = segmenter.segment(mp_img).confidence_masks[0].numpy_view()

Sí, absolutamente. En cada cuadro de esta segunda pasada, el programa vuelve a llamar a la IA (segmenter.segment).

- **¿Por qué?**: Porque aunque en la primera pasada encontramos la "mejor máscara", la gente se mueve. Necesitamos que la IA nos diga en **este milisegundo exacto** dónde cree que estás tú,como sabemos que la IA puede fallar o ser lenta, no usaremos este resultado a ciegas. Lo guardamos en raw para compararlo luego con la máscara que "empujamos" con el flujo óptico..
- **El resultado (raw)**: Como ya aprendiste, no es un blanco y negro puro, sino un mapa de **probabilidades** (0.0 a 1.0) que indica qué tan segura está la IA de que cada píxel es parte de la persona.

¿Qué hace cv2.resize(raw, (w, h))?

Esta línea es un paso de **sincronización de dimensiones**.

- **El Problema:** Para ahorrar batería y procesar más rápido, la IA de MediaPipe no analiza la imagen a máxima resolución (por ejemplo, si tu video es 4K, la IA quizás solo analiza una versión pequeña de 256x256 píxeles).
- **La Solución:** cv2.resize toma esa respuesta pequeña y borrosa de la IA y la **estira** (escala) hasta que tenga exactamente el mismo ancho (w) y alto (h) que tu video original.
- **Por qué es vital:** Si no hicieras esto, intentarías aplicar una máscara de juguete sobre un video gigante. Los píxeles no coincidirían y el programa daría un error matemático al intentar multiplicar matrices de diferentes tamaños.

El Punto de Partida: La Máscara Maestra

current = self.best_mask.copy()

El programa empieza cada cuadro con una base sólida: la best_mask (aquella que encontramos en la primera pasada porque era de excelente calidad). Hacemos una copia para no alterar la original.

La Condición de Activación

if has_motion and flow is not None and self.prev_mask is not None:

El "empuje" del movimiento solo se activa si se cumplen tres cosas:

- **has_motion**: El radar detectó que algo se movió.

- **flow is not None**: Tenemos el mapa de flechas listo.
- **self.prev_mask**: Existe una máscara del cuadro anterior para poder moverla.

El "Warping" (La Deformación)

warped = self._warp_mask(self.prev_mask, flow)

Aquí llamas al método que analizamos al principio (_warp_mask). Toma la silueta del cuadro anterior y la **empuja** siguiendo las flechas del flujo óptico. Si tu cabeza se movió a la derecha, la máscara se estira a la derecha.

La Fusión de Seguridad

current = np.maximum(current, warped * 0.9)

Esta es la línea clave. Combina dos fuentes de información:

1. **current**: La máscara maestra (lo que sabemos que es una persona).
2. **warped * 0.9**: La máscara que acabamos de mover por flujo óptico, pero ligeramente "atenuada" (al 90%).

¿Por qué el 0.9? Porque el flujo óptico no es perfecto. Al multiplicarlo por 0.9, le damos un poco menos de importancia que a la máscara actual, evitando que si hay un error en el movimiento, la máscara se "vaya volando" fuera de tu cuerpo.

Esta sección del código es el **"Filtro de Suavizado Temporal"**. Su objetivo es eliminar el "parpadeo" (flickering) que a veces ocurre cuando la máscara vibra en los bordes.

El Historial (self.mask_history.append)

self.mask_history.append(current)

El programa no solo mira el cuadro actual, sino que guarda las últimas máscaras en una lista llamada mask_history. Es como una **memoria a corto plazo**. Generalmente, esta lista está configurada para guardar los últimos 3 o 5 cuadros (según el *preset*).

La Comparación: if len(self.mask_history) >= 2

Aquí el programa pregunta: **"¿Ya tengo suficiente memoria guardada?"**.

- **Si es menor a 2:** Significa que es el primer frame del video. No hay nada con qué comparar, así que simplemente usa la máscara actual (current).
- **Si es 2 o más:** Significa que ya tenemos una máscara anterior y la de ahora. Entonces puede empezar a aplicar el "suavizado".

El Suavizado (np.maximum.reduce)

stable = np.maximum.reduce(list(self.mask_history))

Esta es la parte más inteligente. np.maximum.reduce toma todas las máscaras que hay en el historial y las "apila".

- **¿Qué hace?:** Para cada píxel, busca el valor máximo entre todas las máscaras de la memoria.
- **El resultado:** Si en un cuadro la IA "olvidó" pintar un pedacito de tu oreja, pero en el cuadro anterior sí estaba pintado, esta función **rescata** ese pedacito.

La Actualización de la Referencia

self.prev_mask = stable.copy()

Finalmente, guarda esta máscara ya suavizada y estable como la prev_mask. Esta será la que se use en el siguiente cuadro para el "Warping" (el empuje del movimiento).

ALPHA Y COMPOSITE

Esta es la etapa final de la "fábrica": el **ensamblaje**. Aquí es donde el programa toma todas las piezas (tu imagen nítida, el fondo borroso y la máscara) y las une para crear el fotograma final del video.

alpha = self._create_alpha(stable, raw, frame)

bg = self._create_background(frame, h, w)

- **alpha**: Es una máscara de suavizado. No es blanco y negro puro, sino que tiene tonos de gris en los bordes. El blanco (1.0) significa "Persona", el negro (0.0) significa "Fondo", y el gris (0.5) es la zona de transición para que el recorte no se vea como una calcomanía pegada.

- **bg (Background)**: Es una copia de tu fotograma original, pero a la que se le ha aplicado un desenfoque (Blur). Es lo que se verá "detrás" de ti.

El truco del "Eje Extra" (np.newaxis)

alpha_3 = alpha[:, :, np.newaxis]

Este es un paso matemático crítico.

- Tu máscara alpha es 2D (Alto x Ancho).
- Tu imagen y el fondo son 3D (Alto x Ancho x 3 canales RGB).
- **¿Qué hace?**: Añade una dimensión extra a la máscara para que "encaje" con los 3 canales de color. Así, puedes multiplicar la máscara por el Rojo, el Verde y el Azul al mismo tiempo.

La Ecuación Maestra (Composición Alpha)

final = (frame.astype(np.float32) * alpha_3 +

bg.astype(np.float32) * (1.0 - alpha_3))

Esta es la fórmula estándar de la industria del cine y edición:

1. **frame * alpha_3**: Multiplica la imagen original por la máscara. Donde hay blanco, te quedas tú; donde hay negro, te borras.
2. **bg * (1.0 - alpha_3)**: Multiplica el fondo borroso por la máscara invertida. Donde estabas tú, ahora hay un "hueco" negro; donde estaba el fondo, ahora aparece el desenfoque.
3. **La Suma (+)**: Al sumar ambos resultados, tu silueta nítida rellena el "hueco" del fondo borroso.

Medición del Rendimiento

self.frame_times.append(time.perf_counter() - start)

Como pusimos un cronómetro al principio (start), ahora calculamos la diferencia.

- **¿Qué hace?**: Guarda cuánto tiempo exacto tomó procesar este cuadro (en milisegundos). Al final del video, el programa usará esta lista para decirte: *"Procesé tu video a una media de 24.5 FPS"*.

El Sistema de Diagnóstico

```
if diagnose and frame_idx < 3:
    self._save_diagnostics(frame_idx, frame, raw, stable, alpha, final)
```

- **¿Qué hace?**: Si activaste el modo diagnose, el programa guardará imágenes individuales de los **primeros 3 cuadros**.
- **¿Para qué sirve?**: Te permite ver "las tripas" del proceso. Guarda la imagen original, la máscara de la IA, la máscara estabilizada y el resultado final por separado. Es como hacer una auditoría para asegurarte de que todo está funcionando bien antes de que el programa siga procesando el resto del video.

Grabación del Fotograma (out.write)

```
out.write(final.astype(np.uint8))
```

Esta es la línea donde el video se "materializa":

- **astype(np.uint8)**: Durante los cálculos anteriores usamos decimales (float32) para tener precisión. Pero los archivos de video estándar no entienden decimales, necesitan números enteros de 8 bits (0 a 255). Esta conversión transforma los datos al formato de imagen estándar.
- **out.write(...)**: Toma esa matriz de píxeles ya procesada con el efecto Bokeh y la envía al archivo que definimos al principio con VideoWriter. Es como "pegar" el fotograma en el rollo de película final.

El Contador Final

```
self.frame_count += 1
```

- **¿Qué hace?**: Incrementa el contador global. Aunque estamos usando frame_idx en el bucle, self.frame_count mantiene la cuenta total de fotogramas procesados con éxito. **Una vez que esta línea se ejecuta, el bucle for vuelve arriba para empezar con el siguiente cuadro**.

¿Qué está pasando "bajo el capó"?

```
# 1. Libera el archivo de video original (lectura)
cap.release()

# 2. Finaliza y cierra el archivo de video procesado (escritura)
out.release()
```

1. cap.release()

Al igual que en la primera pasada, esta línea le dice al sistema operativo: *"Ya terminé de leer el video original, puedes desbloquear el archivo"*.

- **Importancia:** Libera la memoria RAM que se usaba para almacenar los buffers de video que estaban pendientes de lectura.

2. out.release() (La más crítica)

Esta es, probablemente, la línea más importante de toda la segunda pasada.

- **El "Sellado" del archivo:** Los archivos de video (como el MP4) tienen una estructura compleja. No son solo una secuencia de imágenes; al final del archivo debe escribirse un "índice" o cabecera final que le dice a los reproductores de video cuánto dura el clip y cómo debe leerse.
- **¿Qué pasa si no se pone?:** Si el programa se cierra sin ejecutar out.release(), el archivo de video se queda "abierto". Al intentar abrirlo en VLC o QuickTime, te dirá que el **archivo está dañado o es ilegible**, porque le falta la firma de cierre.
- **Finalización de la compresión:** El códec (mp4v) termina de comprimir los últimos cuadros que podrían estar todavía en la memoria temporal.

 MÉTODO _SAVE_DIAGNOSTICS

Este método es una herramienta de depuración. Su función es "congelar" un instante del video y guardar en tu computadora imágenes individuales de cómo se ve cada etapa del algoritmo.

 Las 6 Entradas (Parámetros)

1. **idx (int):** El número del frame (ejemplo: 0, 1 o 2). Sirve para que los nombres de los archivos no se repitan.
2. **frame (np.ndarray):** La imagen original tal cual sale de la cámara.
3. **raw (np.ndarray):** La máscara de confianza "sucia" que entrega la IA de MediaPipe.
4. **mask (np.ndarray):** La máscara estabilizada y procesada (después del flujo óptico y la acumulación).
5. **alpha (np.ndarray):** La máscara final con bordes suavizados lista para la mezcla.
6. **final (np.ndarray):** El resultado final con el fondo ya desenfocado.

 Paso a Paso: ¿Qué está pasando aquí?

El Revelado de las Imágenes

```
cv2.imwrite(f'diagnostics_pics/diag_f{idx}_input.png', frame)
cv2.imwrite(f'diagnostics_pics/diag_f{idx}_raw.png', (raw * 255).astype(np.uint8))
```

- **cv2.imwrite**: Es el comando para guardar un archivo de imagen en el disco duro en la carpeta "diagnostics_pics/".
- **raw * 255**: Como la IA entrega valores entre 0.0 y 1.0, los multiplicamos por 255 para que se conviertan en una imagen visible (donde 255 es blanco puro).

La Detección de Bordes (El Overlay)

```
overlay = frame.copy()
edge = cv2.Canny((alpha * 255).astype(np.uint8), 100, 200)
```

Aquí el programa hace algo muy profesional:

- Usa un algoritmo llamado **Canny** (Detección de bordes).
- Analiza la máscara alpha para encontrar exactamente por dónde pasa el contorno de tu silueta.

El Dibujo del Contorno Verde

```
overlay[edge > 0] = [0, 255, 0]
cv2.imwrite(f'diag_f{idx}_overlay.png', overlay)
```

- Toma la imagen original (frame) y pinta de **verde brillante** [0, 255, 0] los píxeles donde el detector Canny encontró el borde.
- Esto te permite ver una imagen (el "overlay") donde sales tú con un contorno verde neón. Es la mejor forma de comprobar si la IA te está recortando bien o si está cortando parte de tu brazo o cabeza.

EL MÉTODO PROCESS (Punto de Entrada Principal)

Las 4 Entradas (Parámetros)

1. **self**: Referencia a la instancia de la clase (estándar en Python).
2. **input_path (str)**: La ruta del video original que quieres procesar.
3. **output_path (str)**: El nombre o ruta del archivo de video final que se creará.
4. **diagnose (bool)**: Un interruptor. Si lo pones en True, activará el guardado de las imágenes de diagnóstico que vimos anteriormente. Por defecto es False.

¿Qué parámetros imprime?

Al ejecutarse, el método imprime un bloque de texto que sirve como **resumen de configuración**:

1. **Título**: ZOOM-STYLE BOKEH PROCESSOR (Para confirmar que el motor está encendido).
2. **Preset**: Muestra el nombre (self.preset.name) y la descripción (self.preset.description) de la calidad elegida (Low, Medium o High). Esto le dice al usuario, por ejemplo: *"High - Best quality, stronger blur"*.
3. **Mode**: Indica si se está usando BLUR (desenfocar el fondo original) o REPLACE (poner una imagen de fondo nueva).

El Contexto de la IA (with ... as segmenter)

with vision.ImageSegmenter.create_from_options(self.seg_options) as segmenter:

- **¿Qué hace?**: Abre la sesión de la IA de MediaPipe.
- **El uso de with**: En Python, esto se llama un "Context Manager". Es muy importante porque garantiza que, una vez que el programa termine de procesar el video (o si ocurre un error), la memoria de la tarjeta gráfica o el procesador que usa la IA se **limpie automáticamente**. Es como encender la maquinaria de una fábrica y asegurar que se apague sola al final.
- **segmenter**: Es el nombre del "cerebro" que le pasamos al _first_pass y al _second_pass.

El Oráculo: _first_pass

w, h = self._first_pass(input_path, segmenter)

Aquí se activa la **Fase 1**. El programa recorre todo el video solo para medir y observar. Al terminar, devuelve el ancho (w) y el alto (h) del video para que el resto del código sepa qué tamaño de lienzo usar.

El Filtro de Error (Control de Calidad)

```
if self.best_mask is None:
    print("ERROR: No valid segmentation found!")
    return
```

- **¿Por qué ocurre?**: Si procesas un video donde no hay personas (por ejemplo, un paisaje o una habitación vacía), el _first_pass nunca encontrará una máscara de buena calidad.
- **return**: Si best_mask es None, el programa se detiene ahí mismo. Esto evita que el código intente aplicar un efecto de desenfoque sobre "nada", lo que causaría que el programa se cerrara con un error crítico (crash).

El Guardado de la "Máscara Maestra"

```
if diagnose:
    cv2.imwrite('diag_best_mask.png', (self.best_mask * 255).astype(np.uint8))
```

- **¿Qué hace?**: Si el modo diagnóstico está activado, el programa guarda un archivo de imagen llamado diag_best_mask.png.
- **¿Por qué es importante?**: Esta es la máscara que el programa consideró como la de **mejor calidad** en todo el video (la que tenía mejor confianza y cobertura).
- **Conversión**: Multiplica por 255 y convierte a uint8 para que los datos matemáticos se vuelvan una imagen en blanco y negro que tú puedas abrir y ver en tu computadora. Es la "referencia de oro" que se usará para estabilizar todo el video.

El Disparo de la Fase de Renderizado

```
self._second_pass(input_path, output_path, segmenter, w, h, diagnose)
```

Esta es la llamada al método que realiza el trabajo pesado. Aquí se le entregan todos los "ingredientes" recolectados:

1. **input_path**: El video original para volver a leerlo.
2. **output_path**: El archivo donde se escribirá el resultado.
3. **segmenter**: El motor de IA ya encendido.
4. **w, h**: Las dimensiones exactas para que el video no salga deformado.
5. **diagnose**: Para saber si debe seguir guardando imágenes de prueba.

Tiempo promedio por cuadro (avg_ms)

avg_ms = np.mean(self.frame_times) * 1000

- **self.frame_times**: Es la lista donde guardamos la duración de cada cuadro (calculada con time.perf_counter() - start).
- **np.mean(...)**: Calcula el promedio (la media aritmética) de todos esos tiempos.
- *** 1000**: Como Python mide el tiempo en segundos, multiplicamos por 1000 para convertirlo a **milisegundos (ms)**. Es la unidad estándar en la industria de los videojuegos y el video para medir la latencia.
- **Significado**: Te dice cuánto tiempo, en promedio, le tomó a tu CPU y GPU procesar un solo fotograma (incluyendo la IA, el flujo óptico y el desenfoque).

Cuadros por segundo (fps)

fps = 1.0 / np.mean(self.frame_times)

- **La Fórmula**: La frecuencia f es el inverso del periodo T, es decir, **f = 1/T.**
- **Significado**: Indica la velocidad de procesamiento real. Si el promedio de tiempo por cuadro fue de 0.033 segundos, entonces 1 / 0.033 aproximadamente 30 FPS.
- **Utilidad**: Te permite saber si el programa es capaz de funcionar en **tiempo real**. Para que un video se vea fluido en vivo, necesitamos al menos 24 o 30 FPS. Si este número es bajo (ej. 5 FPS), significa que el algoritmo es muy pesado para tu hardware actual.

RESUMEN DE FINALIZACIÓN (TELEMETRÍA)

```
print(f"\n{'='*60}")
print("COMPLETE")
print(f"{'='*60}")

# Muestra el total de cuadros procesados
print(f"Frames: {self.frame_count}")

# Reporta la latencia media y la fluidez (FPS)
print(f"Speed: {avg_ms:.1f}ms/frame ({fps:.1f} FPS)")

# Confirma la ubicación del archivo resultante
print(f"Output: {output_path}")
print(f"{'='*60}\n")
```

 ¿Qué hace este bloque en detalle?

Este fragmento no realiza cálculos matemáticos, sino que actúa como una **interfaz de comunicación** con el usuario. Es fundamental por tres razones:

1. **Confirmación de Integridad**: Al imprimir el número total de Frames, el usuario puede verificar si coincide con la duración de su video original, asegurándose de que no hubo cortes ni errores durante la renderización.
2. **Métricas de Rendimiento**: Al mostrar los **FPS (Frames Per Second)**, el usuario puede evaluar la potencia de su hardware.
    - *Ejemplo:* Si el resultado es 30 FPS, el procesamiento fue a velocidad de tiempo real.
    - *Ejemplo:* Si es 120 FPS, el software es extremadamente eficiente en esa máquina.
3. **Localización del Activo**: Imprime el output_path, lo cual es una buena práctica de UX (Experiencia de Usuario) para que el cliente sepa exactamente en qué carpeta se guardó su video editado.

Implementación del Patrón de Diseño Facade (Fachada): Abstracción de la API de Usuario (método blur_background)

```
def blur_background(input_video: str,
                    output_video: str,
                    intensity: str = "high",
                    diagnose: bool = False):
```

Tiene **4 parámetros de entrada**:

1. **input_video** (str): La ruta del archivo de video original que quieres procesar (ej: "mi_video.mp4").
2. **output_video** (str): El nombre o ruta donde se guardará el video con el fondo desenfocado (ej: "resultado_bokeh.mp4").
3. **intensity** (str, *opcional*): Define qué tan fuerte será el desenfoque y qué tanta potencia de cómputo se usará. Por defecto es "high". Acepta los valores: "low", "medium" o "high".
4. **diagnose** (bool, *opcional*): Un interruptor (True/False) para decidir si el programa debe generar imágenes de diagnóstico para revisión técnica. Por defecto es False.

processor = ZoomStyleBokeh(preset=intensity)

Acción: Instanciación y Configuración del Cerebro.

- **¿Qué hace?**: Crea un "objeto" (una instancia de la clase) llamado processor. En este momento, se cargan en la memoria de tu computadora todos los parámetros que definiste anteriormente.
- **El rol de preset=intensity**: Aquí es donde el programa elige su "personalidad". Si intensity es "high", el objeto processor se configura internamente con un radio de desenfoque de 27 píxeles y 5 cuadros de suavizado temporal.
- **Carga del Modelo**: Durante esta línea, el sistema localiza el archivo selfie_segmenter.tflite y prepara la Inteligencia Artificial para empezar a recibir imágenes.

processor.process(input_video, output_video, diagnose)

Acción: Disparo del Pipeline de Producción.

- **¿Qué hace?**: Es la orden de inicio. Esta única línea activa toda la maquinaria compleja que analizamos antes.
- **El flujo que dispara**:

1. **Abre el video** original (input_video).
2. **Ejecuta la Fase 1**: Analiza todo el video para encontrar la mejor máscara.
3. **Ejecuta la Fase 2**: Lee el video de nuevo, aplica el flujo óptico, crea el fondo borroso (o reemplazo) y mezcla las imágenes píxel por píxel.
4. **Escribe el archivo**: Guarda el resultado final en la ruta de output_video.
5. **Auditoría**: Si diagnose es verdadero, guarda las fotos de prueba en cada paso crítico.

Método REPLACE_BACKGROUND: Implementación de Escenarios Virtuales

```
def replace_background(input_video: str,
                       output_video: str,
                       background_image: str,
                       diagnose: bool = False):
```

 Análisis Detallado del Funcionamiento

Este método realiza tres acciones críticas que lo diferencian del desenfoque:

1. **Cambio de Modo Operativo (mode=BlurMode.REPLACE)**: Al cambiar esta bandera, el programa ya no ejecutará el filtro de desenfoque (Gaussian Blur) sobre el fondo original. En su lugar, activará la lógica de **redimensionamiento**, donde la imagen de fondo proporcionada se ajusta exactamente al ancho y alto del video original.
2. **Carga de Activos Externos**: El parámetro background_image obliga al sistema a realizar una operación de lectura de disco adicional (cv2.imread). El procesador verifica que la imagen exista y la mantiene en memoria para "pegarla" detrás del usuario en cada fotograma.
3. **Composición mediante Máscara Alpha**: Durante el proceso, el sistema toma tres elementos por cada cuadro:
    - **Primer plano**: El usuario extraído por la IA.
    - **Fondo**: La imagen nueva cargada.
    - **Máscara**: El mapa de transparencia que define qué píxel pertenece a quién. El resultado es una mezcla perfecta donde el usuario parece estar en un lugar totalmente distinto.

El punto de entrada de ejecución del script. Es el bloque que le dice a Python: "Si este archivo se ejecuta directamente (y no se importa como una librería), haz lo siguiente".

El Guardián de Ejecución (if __name__ == "__main__":)

Esta línea es un estándar de oro en la programación profesional con Python.

- **¿Qué hace?**: Asegura que el código de prueba (el procesamiento del video) solo se ejecute si tú lanzas el archivo manualmente. Si otro programa intentara "importar" tus funciones para usarlas en otro lugar, esta parte se ignoraría, evitando que el procesamiento empiece sin querer.

La Llamada a la Función Principal

blur_background('3_seconds_video.mp4', 'ZOOM_BLUR.mp4', intensity="high", diagnose=True)

Aquí es donde se da la orden de "Fuego". El programa realiza las siguientes acciones en cadena:

1. **Lectura del Origen**: Busca el archivo físico llamado '3_seconds_video.mp4' en la carpeta de tu proyecto.
2. **Destino**: Prepara el contenedor para el nuevo archivo 'ZOOM_BLUR.mp4'.
3. **Configuración de Intensidad**: Al elegir "high", el sistema activa el perfil de máxima calidad (27 píxeles de radio de desenfoque y 5 cuadros de estabilidad temporal).
4. **Activación de Rayos X (diagnose=True)**: Esta es la instrucción más importante para un desarrollador. Al estar en True, el programa no solo te entregará el video final, sino que "escupirá" en tu carpeta una serie de imágenes (como diag_f0_mask.png) para que puedas ver exactamente cómo la IA está recortando tu silueta.

RESUMEN DESEMPEÑO DEL MODELO

El sistema opera actualmente en modo **asíncrono (offline)**. Debido a la alta carga computacional del preset 'High' (que incluye filtrado guiado y estabilidad temporal), el tiempo de inferencia y renderizado por cuadro es de **259.0ms** . Esto indica que el procesamiento es aproximadamente **8 veces más lento que el tiempo real**, priorizando la precisión visual y la eliminación de artefactos sobre la velocidad de entrega.