Interview Task: Real-Time Video Anonymization Pipeline

Welcome to your technical interview task. This project is designed to assess your skills in video processing, and computer vision, by having you build a real-time face anonymization pipeline.

The unique feature of this task is that **you choose your own difficulty**. Read the three levels below and choose the one that best showcases your skills and experience. Higher levels are more complex but offer a greater opportunity to demonstrate deep technical expertise.

The Core Problem

All three levels address the same core problem:

- **Input:** A standard video file https://www.pexels.com/video/people-wearing-face-mask-in-public-area-6960541/ .
- **Process:** For each frame of the video, detect all human faces.
- **Output:** Anonymize the detected faces by applying a blur effect.
- **Display:** Show the resulting video stream with the blurred faces in real-time.

Choose Your Level

Select **one** of the following three levels to complete.

Level 1: The analyst

This level focuses on core computer vision and Python scripting skills.

- **Objective:** Create a Python script that uses the OpenCV library exclusively to handle all video operations.
- **Implementation:**
  - Use `cv2.VideoCapture()` to read the video file frame by frame.
  - Use a pre-trained OpenCV face detection model (e.g., Haar Cascade or a DNN model) to find face coordinates.
  - For each face found, use `cv2.GaussianBlur()` to apply the blur.
  - Use `cv2.imshow()` to display the processed frames in a window.

- **Skills Demonstrated:** Python proficiency, practical OpenCV usage, basic computer vision algorithms.

Level 2: The Pipeline Integrator

This level demonstrates your ability to integrate custom processing logic into a more robust, high-performance pipeline using GStreamer.

- **Objective:** Build a GStreamer pipeline in Python that hands off frames to OpenCV for processing.
- **Implementation:**
  - Construct a GStreamer pipeline that uses an **`appsink`** element to pull video frames into your Python script.
  - In the `appsink` callback, convert the GStreamer buffer into a NumPy array.
  - Perform the face detection and blurring on the NumPy array using OpenCV, just like in Level 1.
  - Output the result. You can use the simple `cv2.imshow()` for display or, for full credit, push the modified NumPy array back into a second GStreamer pipeline using an **`appsrc`** element for display.
- **Skills Demonstrated:** All skills from Level 1, plus GStreamer framework knowledge, buffer/data manipulation between libraries, and more advanced pipeline design.

Level 3: The Video Systems Engineer

This is a high-risk, high-reward challenge that demonstrates deep, low-level system engineering and performance optimization skills.

- **Objective:** Build a custom, reusable GStreamer plugin in **C/C++** that performs the OpenCV processing natively.
- **Implementation:**
  - Write a new GStreamer element (e.g., `cvfaceblur`) in C or C++. You can use the `gst-plugin-bad cvfilter` or the `gst-plugins-base identity` element as a starting template.
  - In the element's `transform_ip` function, wrap the incoming `GstBuffer` data into a `cv::Mat` object without copying if possible.
  - Use the OpenCV C++ API to perform the face detection and blurring directly on the `cv::Mat`.
  - Integrate your new plugin into the GStreamer build system using **Meson**, linking against the required OpenCV libraries.

- Your final demonstration can be a simple Python script or a `gst-launch-1.0` command that uses your new element (`... ! cvfaceblur ! ...`).
- **Skills Demonstrated:** All concepts from Level 2, plus C/C++ proficiency, GStreamer C API and plugin architecture, native library integration, Meson build system, and performance optimization.

Deliverable (Required for All Levels)

Your submission must be a fully self-contained Docker environment. The goal is for us to clone your repository and run your project with just two commands, regardless of the level you choose.

1. **Git Repository:** Provide a link to a public Git repository (GitHub, GitLab, etc.).
2. **Dockerfile:** Your repository **must** contain a `Dockerfile` that builds an image with all necessary dependencies (Python, OpenCV, GStreamer, C++ compilers, etc.).
   - *For Level 3*, this should ideally be a **multi-stage Dockerfile** where the plugin is compiled in a "builder" stage and only the final runtime libraries and the compiled `.so` file are copied to the final, smaller image.
3. **README.md:** Your repository's `README.md` must contain:
   - A clear statement of which level you chose.
   - The exact `docker build` command to build your image (e.g., `docker build -t face-anonymizer .`).
   - The exact `docker run` command to execute your project.

Evaluation Criteria

- **Functionality:** Does the project run as described for the chosen level via the Docker commands?
- **Code Quality:** Is your code clean, well-structured, and easy to understand?
- **Design Choices:** How effectively did you implement the solution for your chosen level?
- **Containerization:** How well-crafted and efficient is your `Dockerfile`? Did you successfully create a portable environment?