Folder Structure->  
OpenCVEdgeDetection/

├── app/

│ ├── src/

│ │ ├── main/

│ │ │ ├── cpp/

│ │ │ ├── java/

│ │ │ │ └── com/example/opencvedgedetection/

│ │ │ │ ├── MainActivity.kt

│ │ │ │ └── ui/ ← tiny Compose helpers

│ │ │ │ ├── CameraPreviewView.kt

│ │ │ │ └── EdgeGLView.kt

│ │ │ ├── res/

│ │ │ └── AndroidManifest.xml

│ │ └── jniLibs/

│ ├── build.gradle.kts

│ └── CMakeLists.txt

├── gradle/ & gradle.\*

└── README.md

Features->

* Live camera preview using Jetpack CameraX PreviewView (back camera)
* Real-time frame processing in C++ with three modes: Canny edges, GaussianBlur + Threshold mask, Invert gray image
* OpenGL ES overlay with ALPHA\_8 texture blended over preview, synced with JNI toggles
* Jetpack Compose UI with transparent system bars, Material 3 FABs, and runtime permission handling
* FPS counter rendered using Compose with rolling average
* Two FABs for toggling overlay visibility and cycling effects
* Multi-ABI build support for armeabi-v7a, arm64-v8a, x86, x86\_64 with c++\_shared
* Modular and minimal structure with separate /cpp, /gl, and /app (Compose) layers

Setup instructions->

Sure! Here's the setup guide in concise, single-line, point-wise format for easy copying:

* Clone the repo: git clone https://github.com/your-handle/OpenCVEdgeDetection.git && cd OpenCVEdgeDetection
* Install Android NDK (version r27+): via SDK Manager → NDK
* Install CMake (version ≥ 3.22): via SDK Manager → CMake
* Set $ANDROID\_NDK\_HOME if building from CLI to point to installed NDK
* Unzip and copy native libs: cp -R OpenCV-android-sdk/sdk/native/libs/\* app/src/main/jniLibs/
* This places .so files like libopencv\_java4.so for all ABIs in app/src/main/jniLibs/
* No .jar needed since System.loadLibrary("opencv\_java4") is used
* Open Android Studio → Sync Gradle → select device → Run
* App will request camera permission on first launch

Explanation of Architecture->

1. CameraX ImageAnalysis provides frames using only the Y-plane (grayscale).
2. Frame is passed to native-lib.cpp via JNI with an effectMode integer.
3. In C++ (OpenCV CPU), one of the following effects is applied:
   * Canny edge detection
   * Gaussian blur + threshold
   * Invert grayscale
4. The processed result is returned as a ByteArray (mask or edges).
5. EdgeRenderer.kt uploads this ByteArray to OpenGL ES 2.0 as an ALPHA\_8 texture.
6. Jetpack Compose UI displays:
   * PreviewView for the camera
   * GLSurfaceView overlay for the effect
   * FABs to toggle visibility and switch modes
   * Text showing FPS and current mode
7. composeView.requestRender() triggers a re-render with the updated texture.

A couch and a coffee table in a room

AI-generated content may be incorrect.A blurry image of a window

AI-generated content may be incorrect.A close-up of a painting

AI-generated content may be incorrect.