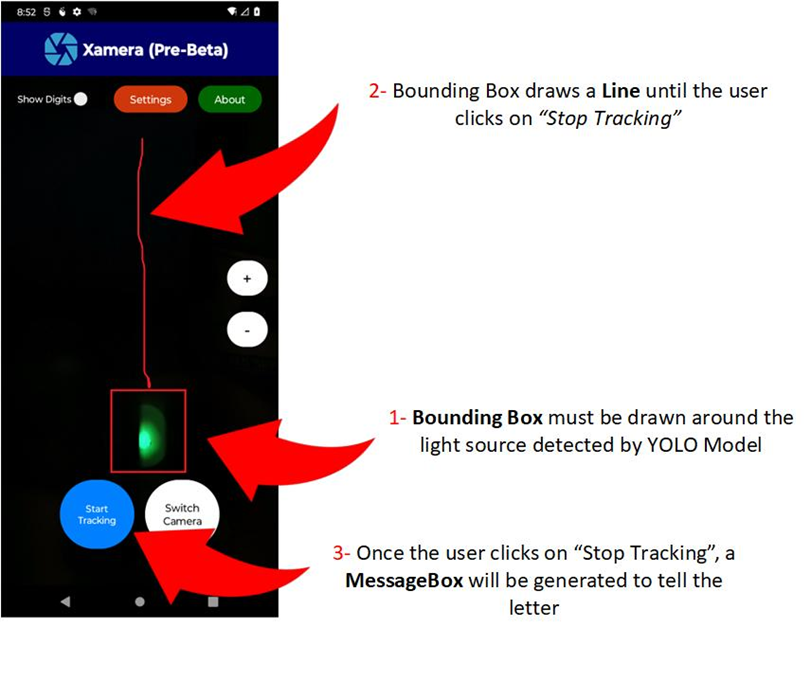
**Inference Instructions for Xamera**

This document explains how Xamera (Pre-Beta) uses a YOLO-based model to track a light source (e.g., a fingertip with a small LED) and infer a letter from the path drawn on-screen. Key roles:

* **MainActivity.kt** – Manages the UI and triggers the VideoProcessor when the user taps **Start Tracking** or **Stop Tracking**.
* **VideoProcessor.kt** – Once triggered, it processes the camera feed, draws the bounding box around the fingertip light, tracks the line, and converts the line data into a 640×640 image.
* **InferenceYolo.kt** – Uses the YOLO-based model (possibly with LSTM) to interpret the 640×640 image and produce the final letter.



**Figure 3:** YOLO detects the bounding box and VideoProcessor.kt keeps track of the lines until the user presses on “Stop Tracking”

**1. Step-by-Step Workflow**

**Step 1 – User Initiates Tracking**

1. **User taps “Start Tracking”** in MainActivity.
2. **VideoProcessor is triggered immediately**, starting to process the live camera frames.

**Step 2 – Detecting the Light Source**

1. VideoProcessor runs the YOLO detection on each incoming camera frame to **find a bright source** (the fingertip light).
2. A **bounding box** is drawn around the detected light source in real time.

**Step 3 – Drawing the Line**

1. As the bounding box moves (following the fingertip), VideoProcessor accumulates the bounding-box coordinates.
2. **A line** (e.g., a red stroke) is drawn to connect each sequential position of the bounding box—this visually represents the user’s gesture.

**Step 4 – Stopping the Tracking**

1. **User taps “Stop Tracking”** in MainActivity when done drawing.
2. VideoProcessor finalizes the coordinate list for the user’s drawn line.

**Step 5 – Creating the 640×640 Image**

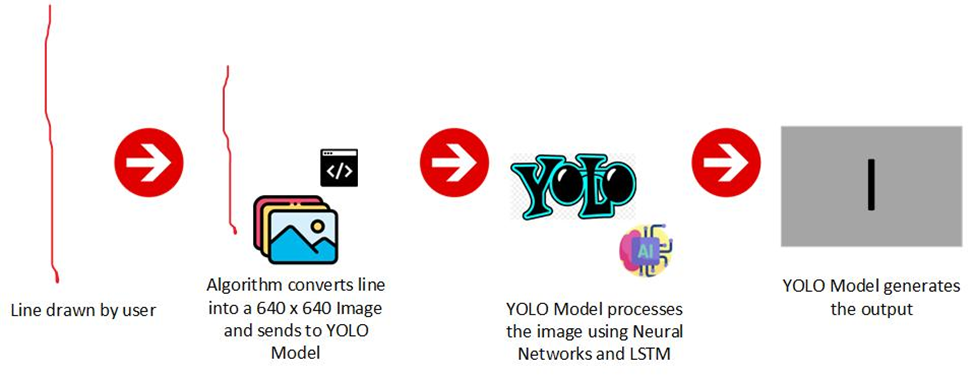
1. Still within VideoProcessor, the list of coordinates is used to **render a 640×640 bitmap** (or other image format).
2. The line is drawn on a blank (black or transparent) 640×640 canvas so the model can process a standardized input.

**Step 6 – YOLO Inference**

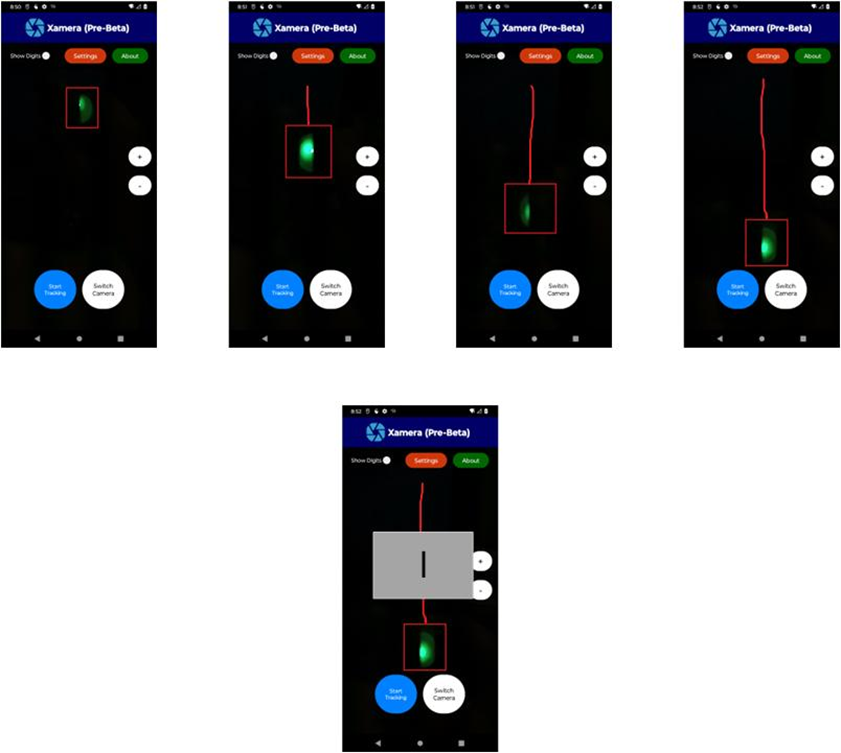
1. The 640×640 image is **passed from VideoProcessor to InferenceYolo.kt**.
2. YOLO (optionally with an LSTM) **analyzes the line shape** and predicts a letter.
3. The result is returned to MainActivity.

**Step 7 – Displaying the Inference**

1. In MainActivity, a **MessageBox** (or dialog) **shows the predicted letter** to the user.
2. The user may confirm, dismiss, or proceed to track another letter if desired.



**Figure 2:** Shows the inference process within Xamera



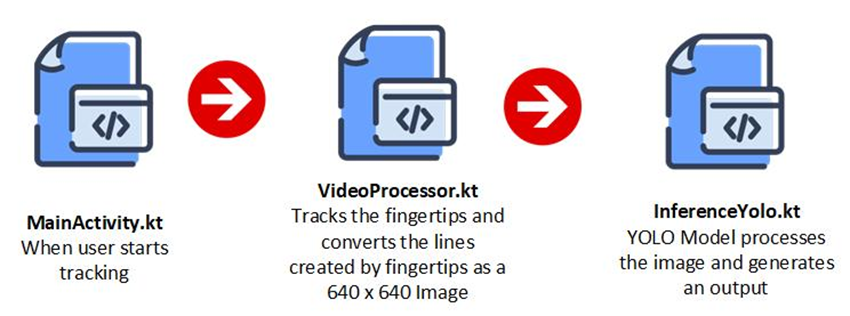
**Figure 3:** YOLO detects the bounding box and VideoProcessor.kt keeps track of the lines until the user presses on “Stop Tracking”

**2. Visual Breakdown**

1. **Start Tracking** → triggers VideoProcessor.
2. **Bounding Box Appears** → Light source is detected.
3. **Line is Drawn** → Coordinates are recorded and visualized.
4. **Stop Tracking** → Final line is converted into 640×640.
5. **Model Inference** → YOLO + LSTM processes the image.
6. **Result** → A letter is displayed on-screen.

**3. Responsibilities by File**

* **MainActivity.kt**
  + Handles UI buttons: **Start Tracking**, **Stop Tracking**, **Switch Camera**, etc.
  + Triggers VideoProcessor when **Start Tracking** is tapped.
  + Receives the YOLO inference result from InferenceYolo.kt and displays it to the user.
* **VideoProcessor.kt**
  + **Starts** when the user clicks **Start Tracking**.
  + Continuously monitors the camera feed to detect the bounding box using YOLO.
  + Draws the line based on bounding-box coordinates.
  + On **Stop Tracking**, finalizes the line drawing and **creates a 640×640 image**.
  + Passes this image to InferenceYolo.kt.
* **InferenceYolo.kt**
  + Loads and runs the YOLO-based model on the 640×640 image.
  + Optionally uses an LSTM for sequence or shape analysis.
  + Returns the predicted letter (e.g., A, B, C, etc.) to MainActivity for display.



**Figure 4:** Shows the flow of class activities within Xamera

**4. Technical Notes & Recommendations**

* **YOLO Confidence Threshold**: Ensure it’s configured to reliably detect only the intended fingertip light source and ignore noise.
* **Line Drawing Implementation**:
  + Store bounding-box centers in a list.
  + Use an Android Canvas or similar approach to render the path onto a bitmap.
* **Performance**:
  + Consider using background threads or coroutines to avoid blocking the UI when performing YOLO inference.
  + Optimize camera frame capture for minimal latency.
* **Edge Cases**:
  + Multiple bright objects could cause bounding-box confusion; handle gracefully by picking the most confident detection.
  + If the user’s line is too short or incomplete, YOLO may not infer a valid letter, so display an “Unrecognized Gesture” message.

**5. Conclusion**

By tapping **Start Tracking**, the **VideoProcessor** is activated. It detects the fingertip, draws the gesture, and, upon stopping, converts the line into a 640×640 image. The YOLO model in **InferenceYolo.kt** then analyzes this image, returning a letter for MainActivity to display. This streamlined workflow ensures that the user’s midair-drawn letters are translated into recognized symbols with minimal user intervention.