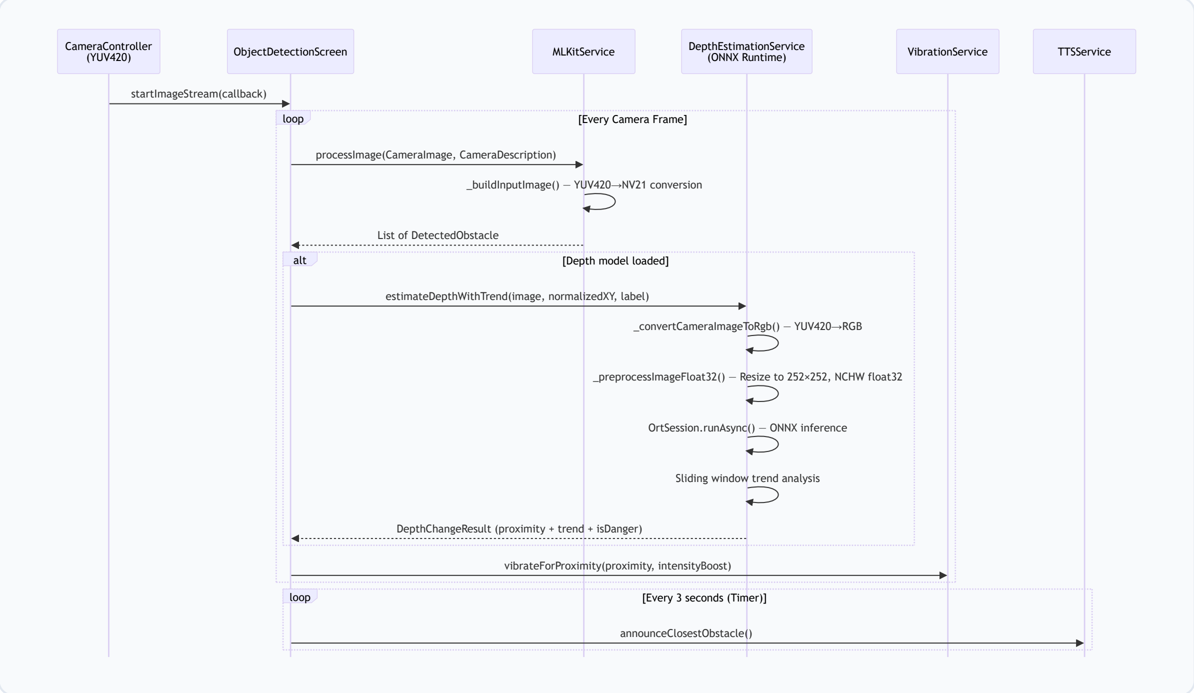
System Flow and Implementation Details

Project: SEE (Visual Assistant) — Flutter Mobile Application
Scope: Code-level execution paths, data flows, and implementation mechanics
Codebase analysed: `lib/` (35 Dart files), `android/.../MainActivity.kt`

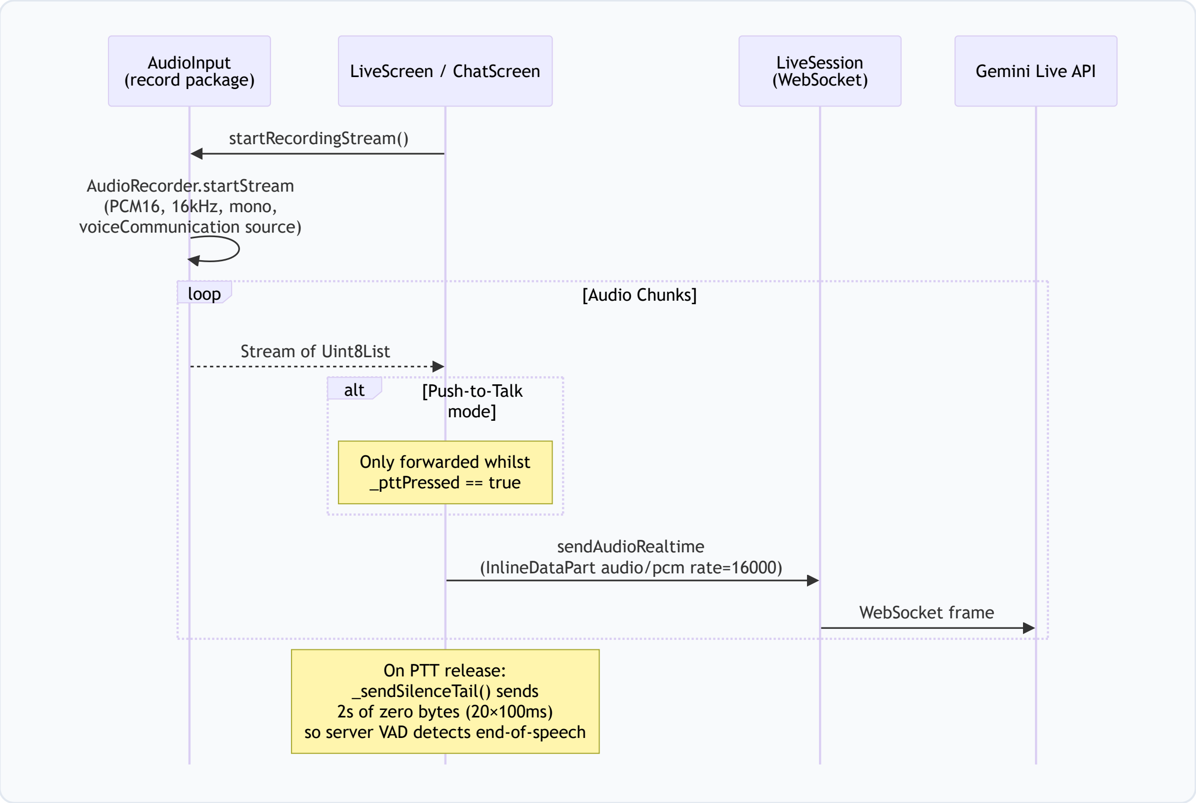
1. System Flowcharts

1.1 Sensor & Hardware Integration — Camera Frame Pipeline

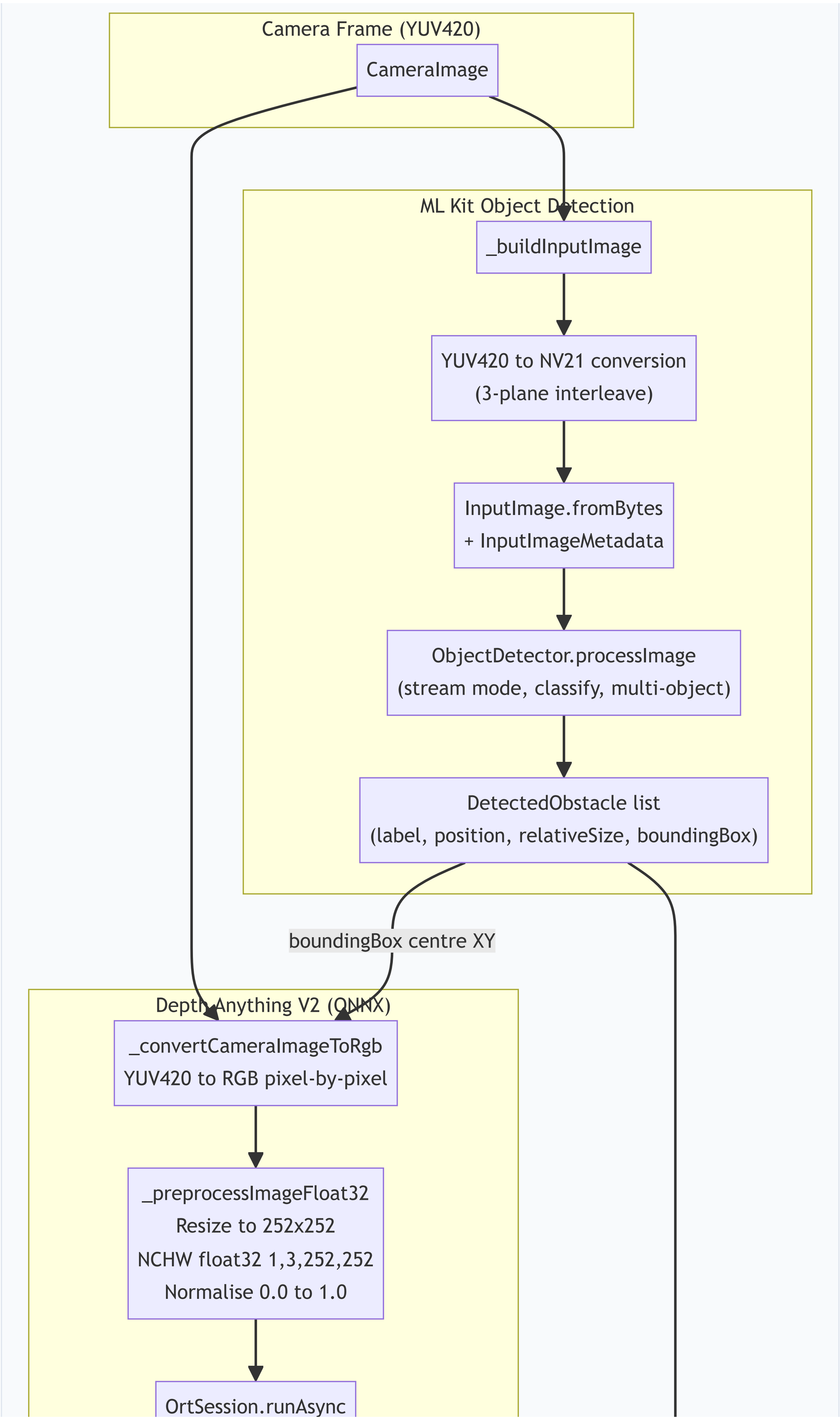
The camera pipeline has two distinct consumers: **ML Kit Object Detection** (processed on every frame) and the **Gemini Live API** (sampled at 1 fps and converted to JPEG via a native platform channel).

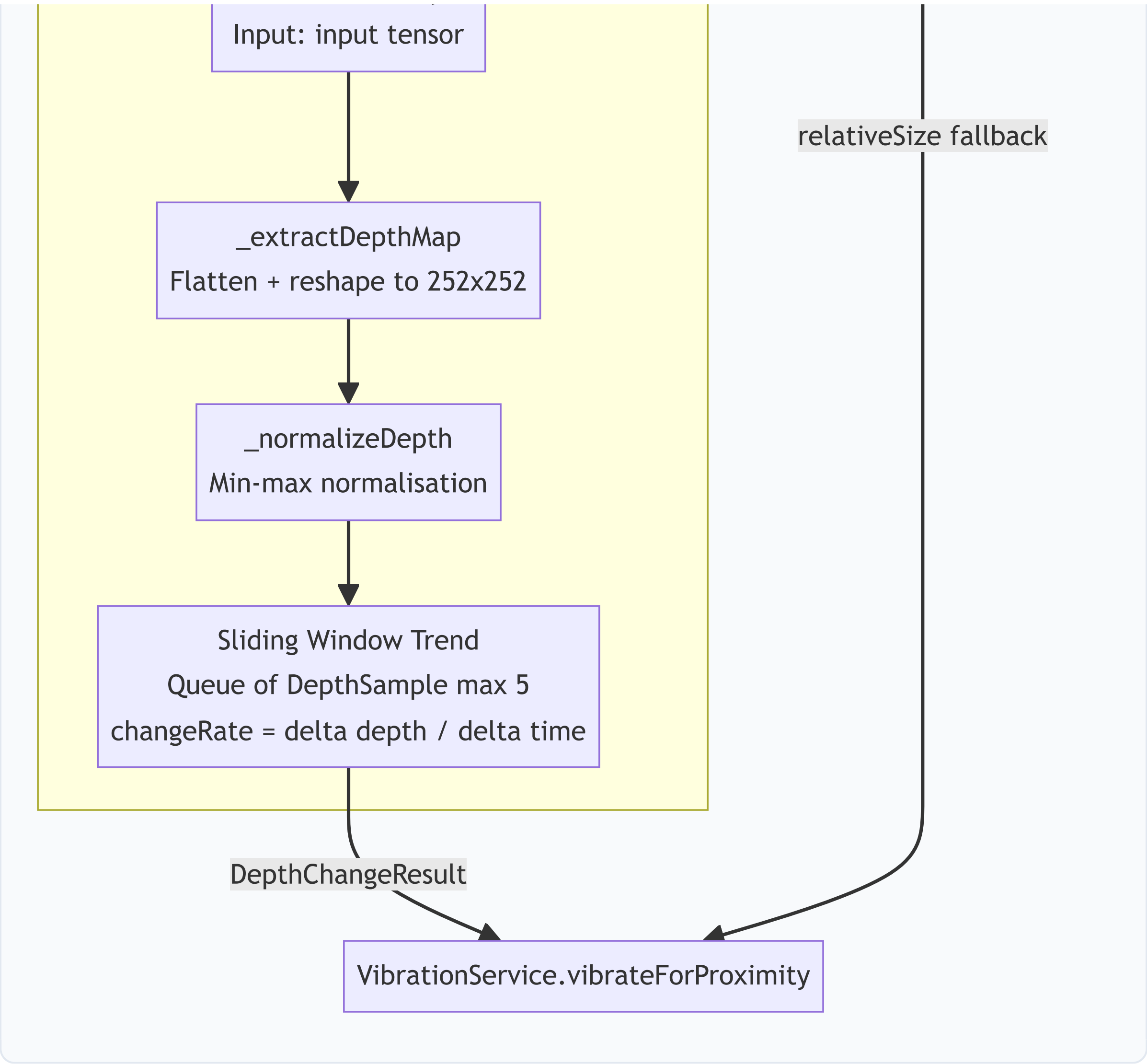

CameraController (YUV420)
ObjectDetectionScreen
MLKitService
DepthEstimationService (ONNX Runtime)
VibrationService
TTSService
startImageStream(callback)
loop
[Every Camera Frame]
processImage(CameraImage, CameraDescription)
_buildInputImage() — YUV420→NV21 conversion
List of DetectedObstacle
alt
[Depth model loaded]
estimateDepthWithTrend(image, normalizedXY, label)
_convertCameraImageToRgb() — YUV420→RGB
_preprocessImageFloat32() — Resize to 252×252, NCHW float32
OrtSession.runAsync() — ONNX inference
Sliding window trend analysis
DepthChangeResult (proximity + trend + isDanger)
vibrateForProximity(proximity, intensityBoost)
loop
[Every 3 seconds (Timer)]
announceClosestObstacle()

1.2 Microphone Audio Capture Pipeline (16 kHz PCM with Hardware AEC)

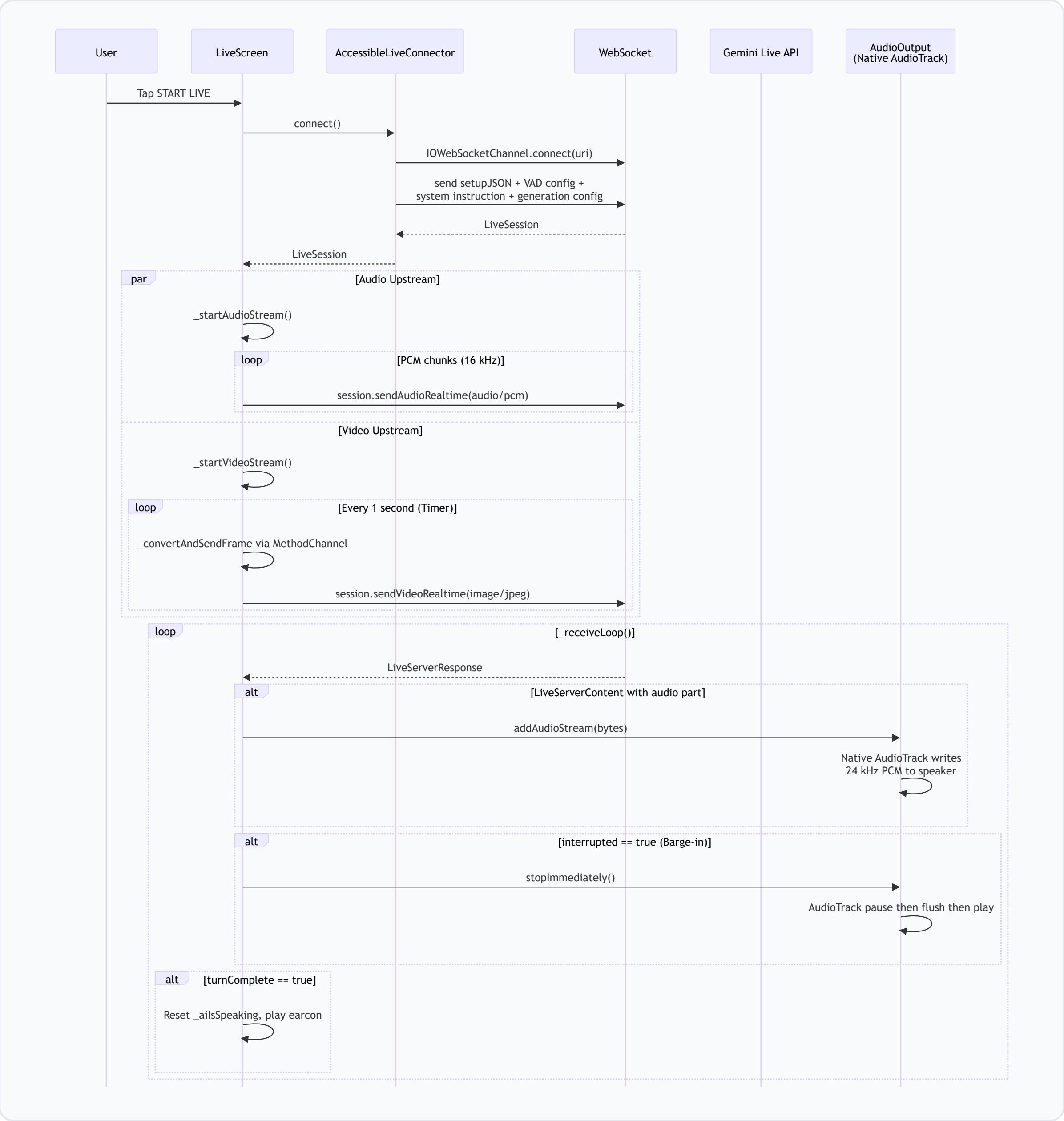


1.3 On-Device ML Inference — Depth Estimation & Object Detection


Camera Frame (YUV420)
CameraImage
ML Kit Object Detection
_buildInputImage
YUV420 to NV21 conversion
(3-plane interleave)
InputImage.fromBytes
+ InputImageMetadata
ObjectDetector.processImage
(stream mode, classify, multi-object)
DetectedObstacle list
(label, position, relativeSize, boundingBox)
boundingBox centre XY
Depth Anything V2 (ONNX)
_convertCameraImageToRgb
YUV420 to RGB pixel-by-pixel
_preprocessImageFloat32
Resize to 252x252
NCHW float32 1,3,252,252
Normalise 0.0 to 1.0
OrtSession.runAsync



1.4 Real-Time Bidirectional Streaming — Gemini Live API

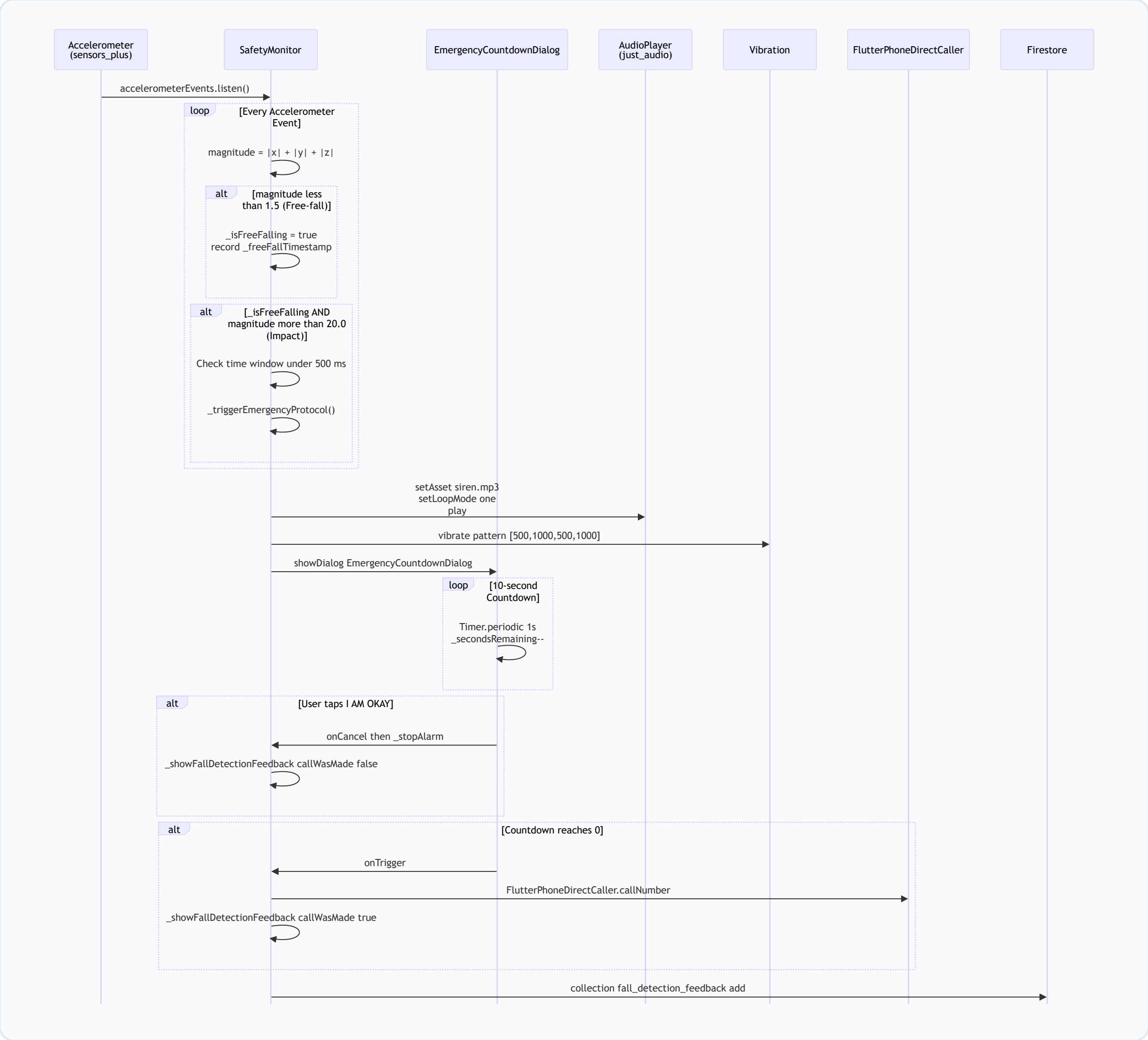

User
LiveScreen
AccessibleLiveConnector
WebSocket
Gemini Live API
AudioOutput (Native AudioTrack)
Tap START LIVE
connect()
IOWebSocketChannel.connect(uri)
send setupJSON + VAD config + system instruction + generation config
LiveSession
LiveSession
par
[Audio Upstream]
_startAudioStream()
loop
[PCM chunks (16 kHz)]
session.sendAudioRealtime(audio/pcm)
[Video Upstream]
_startVideoStream()
loop
[Every 1 second (Timer)]
_convertAndSendFrame via MethodChannel
session.sendVideoRealtime(image/jpeg)
loop
[_receiveLoop()]
LiveServerResponse
alt
[LiveServerContent with audio part]
addAudioStream(bytes)
Native AudioTrack writes 24 kHz PCM to speaker
alt
[interrupted == true (Barge-in)]
stopImmediately()
AudioTrack pause then flush then play
alt
[turnComplete == true]
Reset _aiIsSpeaking, play earcon

Barge-In Handling Detail

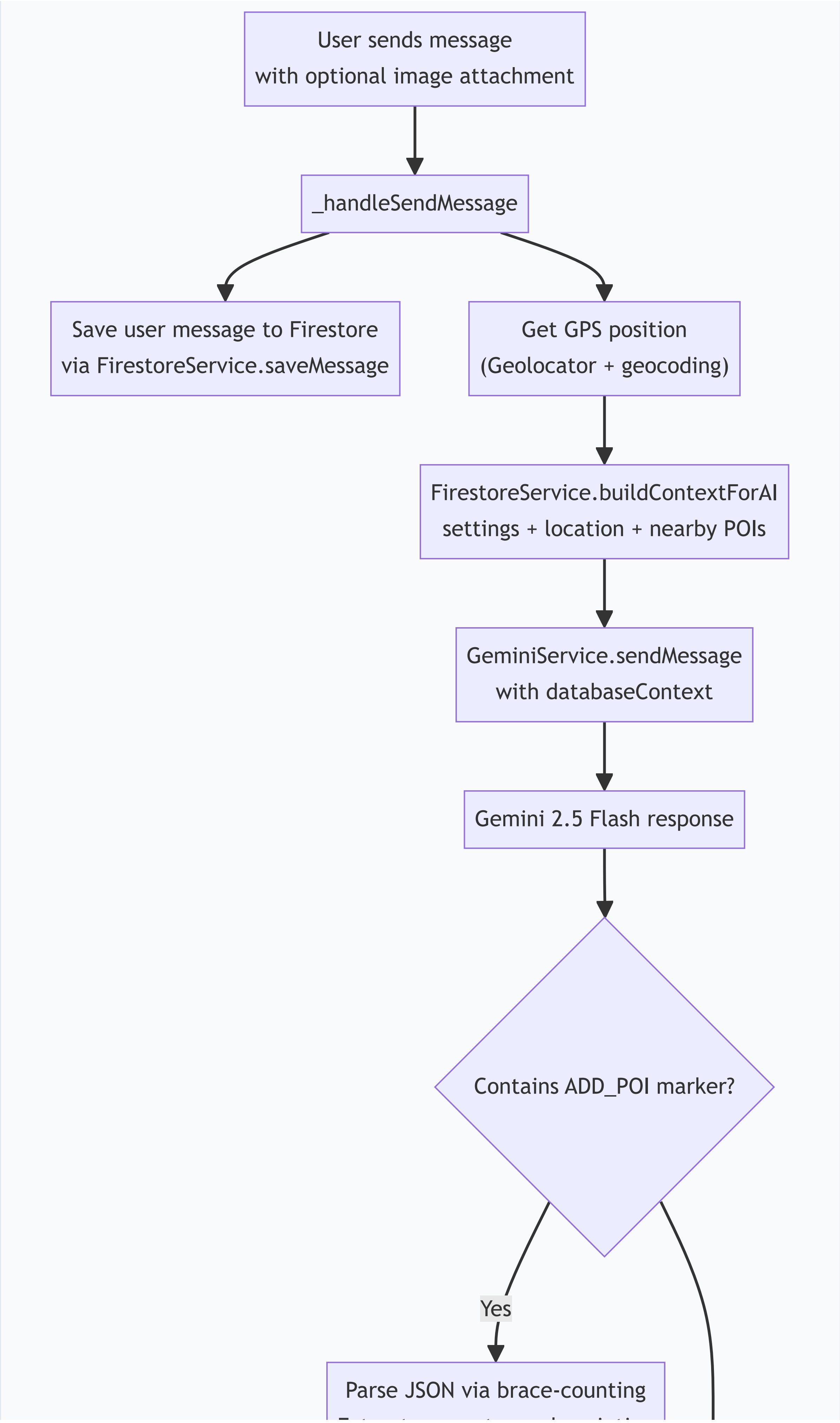
When the server signals `msg.interrupted == true`:

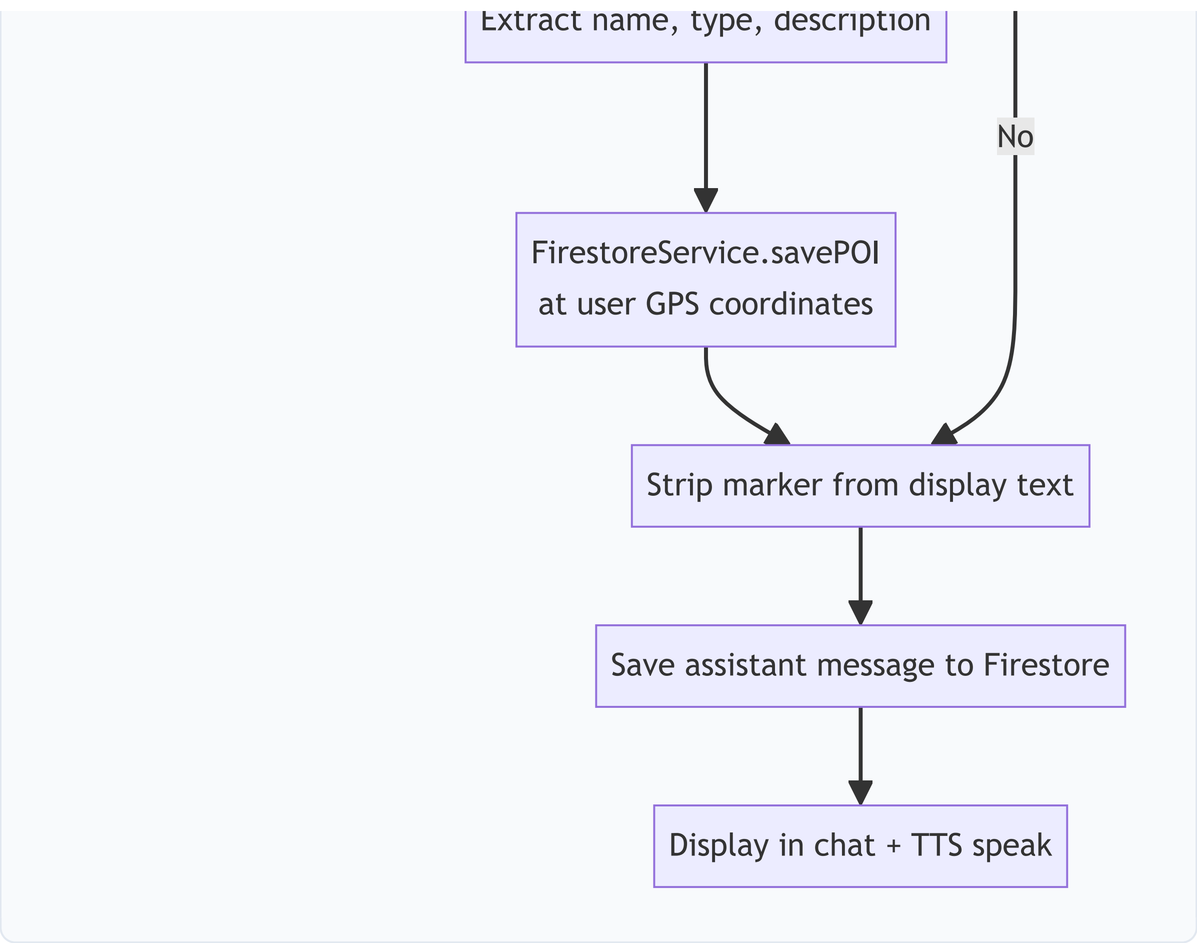
1. `AudioOutput.stopImmediately()` is called, which invokes the native `flush` method.
2. The native Kotlin code executes `audioTrack.pause()` → `audioTrack.flush()` → `audioTrack.play()`, instantly clearing the playback buffer.
3. The `_aiIsSpeaking` flag is reset and `_isFirstAudioChunk` is set to `true` so the next response triggers a fresh earcon.

1.5 Background Fall Detection and SOS Sequence


Accelerometer (sensors_plus)
SafetyMonitor
EmergencyCountdownDialog
AudioPlayer (just_audio)
Vibration
FlutterPhoneDirectCaller
Firestore
accelerometerEvents.listen()
loop
[Every Accelerometer Event]
magnitude = |x| + |y| + |z|
alt
[magnitude less than 1.5 (Free-fall)]
_isFreeFalling = true
record _freeFallTimestamp
alt
[_isFreeFalling AND magnitude more than 20.0 (Impact)]
Check time window under 500 ms
_triggerEmergencyProtocol()
setAsset siren.mp3
setLoopMode one
play
vibrate pattern [500,1000,500,1000]
showDialog EmergencyCountdownDialog
loop
[10-second Countdown]
Timer.periodic 1s
_secondsRemaining--
alt
[User taps I AM OKAY]
onCancel then _stopAlarm
_showFallDetectionFeedback callWasMade false
alt
[Countdown reaches 0]
onTrigger
FlutterPhoneDirectCaller.callNumber
_showFallDetectionFeedback callWasMade true
collection fall_detection_feedback add

1.6 Chat Message Flow with POI Extraction


User sends message
with optional image attachment
_handleSendMessage
Save user message to Firestore
via FirestoreService.saveMessage
Get GPS position
(Geolocator + geocoding)
FirestoreService.buildContextForAI
settings + location + nearby POIs
GeminiService.sendMessage
with databaseContext
Gemini 2.5 Flash response
Contains ADD_POI marker?
Yes
Parse JSON via brace-counting


Extract name, type, description
No
FirestoreService.savePOI
at user GPS coordinates
Strip marker from display text
Save assistant message to Firestore
Display in chat + TTS speak

2. Implementation Details

2.1 State Management

The application uses **Flutter's built-in `StatefulWidget` / `setState` pattern** exclusively. There is no provider, bloc, or riverpod dependency. State is managed as follows:

Domain	State Container	Key Variables
Theme	`ThemeNotifier` (singleton `ChangeNotifier`)	`_themeMode` (light/dark)
Fall Detection	`_SafetyMonitorState`	`_isFreeFalling` , `_freeFallTimestamp` , `_isAlertActive` , `_emergencyPhone`
Object Detection	`_ObjectDetectionScreenState`	`_obstacles` , `_isDetecting` , `_depthModelLoaded` , `_streamErrorCount`
Live Session	`_LiveScreenState`	`_isConnected` , `_isStreamingAudio/Video` , `_aiIsSpeaking` , `_pushToTalkMode` , `_pttPressed`
Chat	`_ChatScreenState`	`_messages` (list), `_isLiveMode` , `_isRecording` , `_isAiSpeaking` , `_pendingImageBytes`
Depth Estimation	`DepthEstimationService`	`_depthHistory` (Map of Queue), `_cachedResult` , `_isProcessing`
ML Kit	`MLKitService`	`_isProcessing` (frame-level mutex), `_frameCount`
TTS	`TTSService` (singleton)	`_isReady` , `_isSpeaking` , `_lastSpoken` , `_lastSpokenTime`

High-frequency sensor streams (camera, accelerometer) are throttled explicitly:

- **Depth estimation:** Minimum 333 ms between inferences (max ~3 FPS) via `_minInterval` .
- **Vibration:** 300 ms throttle via `_lastVibration` timestamp comparison.

- **TTS obstacle announcements:** 3-second `Timer.periodic` in `ObjectDetectionScreen`.
- **Video frames to Gemini:** 1-second `Timer.periodic` in `LiveScreen`.
- **Location awareness POI announcements:** 5-second cooldown in `LocationAwarenessService`.

2.2 Data Structures & Transformation

YUV420 → NV21 Conversion (Dart, for ML Kit)

Located in `ml_kit_service.dart`:

```
Uint8List _convertYUV420ToNV21(CameraImage image) {
  final int ySize = width * height;
  final int uvSize = (width * height) ~/ 2;
  final Uint8List nv21 = Uint8List(ySize + uvSize);

  // Copy Y plane row-by-row (respecting bytesPerRow stride)
  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      nv21[y * width + x] = yPlane.bytes[y * yPlane.bytesPerRow + x];
    }
  }

  // Interleave V and U planes (NV21 = VUVU ordering)
  int uvIndex = ySize;
  for (int y = 0; y < uvHeight; y++) {
    for (int x = 0; x < uvWidth; x++) {
      nv21[uvIndex++] = vPlane.bytes[vIdx]; // V first
      nv21[uvIndex++] = uPlane.bytes[uIdx]; // then U
    }
  }
  return nv21;
}
```

YUV420 → JPEG Conversion (Native Kotlin, for Gemini video stream)

Located in `MainActivity.kt`. Uses Android's `YuvImage.compressToJpeg()` after constructing an NV21 byte array from the three YUV planes received via `MethodChannel`. Quality is set to 40 for bandwidth efficiency.

YUV420 → RGB → Float32 NCHW (Dart, for ONNX Depth)

Located in `depth_estimation_service.dart`:

```
Float32List _preprocessImageFloat32(Uint8List rgbBytes, int w, int h) {
  final inputData = Float32List(1 * 3 * 252 * 252);

  for (int c = 0; c < 3; c++) {        // Channel: R, G, B
    for (int y = 0; y < 252; y++) {
      for (int x = 0; x < 252; x++) {
        // Nearest-neighbour resize
        final srcX = (x * w / 252).round().clamp(0, w - 1);
        final srcY = (y * h / 252).round().clamp(0, h - 1);
        // NCHW layout: all channel-c values contiguous
        inputData[(c*252*252) + (y*252) + x] =
            rgbBytes[(srcY*w + srcX)*3 + c] / 255.0;
      }
    }
  }
  return inputData;
}
```

Temporal Depth Trend — Sliding Window

Located in `depth_estimation_service.dart`:

- Per-object label `Queue<DepthSample>` with a maximum of **5 samples**.
- Trend is calculated as `changeRate = (currentDepth - oldestSample.normalizedDepth) / timeDelta`.
- Thresholds: `>0.15/s` → `approaching_fast`; `>0.05/s` → `approaching`; `<-0.15/s` → `moving_away_fast`.
- `isDanger` flag: `isApproaching && currentDepth > 0.5`.
- Stale entries (>3 seconds since last sample) are cleaned up via `_cleanupOldHistory()`.

Firestore Data Models

```
conversations/{userId}/topics/{topicId}
    ├── createdAt: Timestamp
    ├── lastUpdated: Timestamp
    ├── firstMessage: String (truncated to 100 chars)
    ├── lastMessage: String
    └── messages/{messageId}
            ├── role: 'user' | 'assistant' | 'system'
            ├── content: String
            ├── timestamp: Timestamp
            ├── imageUrl: String?
            └── hasImage: bool

users/{fullName}
    ├── profile: { fullName, email, phone, emergencyContactName,
    │              emergencyContactPhone, uid, createdAt, lastActive }
    └── settings: { visualImpairment, hearingImpairment,
                    mobilityImpairment, preferredVoice,
                    speechRate, highContrastMode }

pois/{poiName}
    ├── name, type, description, safetyNotes
    ├── location: { latitude, longitude, address }
    ├── addedBy: String
    └── createdAt: Timestamp

fall_detection_feedback/{docId}
    ├── userPhone, triggerCorrect, callCorrect
    ├── emergencyNumberDialed, timestamp, deviceTime, userId

traffic_feedback/{docId}
    ├── wasSuccessful: bool
    ├── timestamp, deviceTime
```

POI Extraction from Gemini Response

The chat screen parses Gemini responses for `ADD_POI` markers using **brace-counting** (not regex) to reliably extract nested JSON:

```
int depth = 0;
for (int i = jsonStart; i < displayResponse.length; i++) {
  if (displayResponse[i] == '{') depth++;
  if (displayResponse[i] == '}') depth--;
  if (depth == 0) { jsonEnd = i + 1; break; }
}
```

2.3 Error Handling & Edge Cases

Camera & Sensor Permissions

Scenario	Handling
Camera permission denied	Full-screen card with "Open Settings" button via `permission_handler.openAppSettings()` .
Location permission denied	`Geolocator.requestPermission()` ; if permanently denied, TTS speaks a message and UI shows retry/settings dialog.
Location permission permanently denied	`AlertDialog` with "Open Settings" option.
Microphone permission denied	`AudioInput` throws `Exception('Microphone permission not granted')` .

Network & API Failures

Scenario	Handling
Gemini API error (chat mode)	Returns `"Error communicating with AI service"` — displayed as model message.
Live API WebSocket closed	`_receiveLoop()` detects 'Closed' and calls `_stopAll()` to tear down streams.
Custom VAD connect fails	Falls back to `_sdkFallbackConnect()` using standard Firebase AI SDK.
Places API (New) HTTP error	Falls back to `_searchPlacesOldApi()` using legacy Nearby Search endpoint.
Location timeout	15-second timeout; falls back to `Geolocator.getLastKnownPosition()` .

ML Inference Failures

Scenario	Handling
Stream mode repeated failures	After **100** consecutive empty results, switches to file-based detection with 2-second Timer.
ONNX model load failure	Returns `false` ; object detection continues without depth data.
Depth estimation busy	Returns `_cachedResult` when within 333 ms throttle window.

TTS Engine Binding (Xiaomi/MIUI)

`TTSService.initialize()` retries configuration up to **5 times** with increasing delays (1s, 2s, 3s, 4s, 5s). If all attempts fail, `_isReady` is still set to `true` so that `speak()` can retry at call time.

App Lifecycle

Both `LiveScreen` and `ObjectDetectionScreen` implement `WidgetsBindingObserver`:

- **`inactive`**: Calls `_stopAll()` / cancels timers and disposes camera.
- **`resumed`**: Re-initialises the camera.

2.4 Package Implementation

`record` (Audio Capture)

Configured in `audio_input.dart`:

```
final stream = await _recorder.startStream(
  const RecordConfig(
    encoder: AudioEncoder.pcm16bits,
    numChannels: 1,
    sampleRate: 16000,
    androidConfig: AndroidRecordConfig(
      audioSource: AndroidAudioSource.voiceCommunication,
      audioManagerMode: AudioManagerMode.modeInCommunication,
      muteAudio: false,
    ),
  ),
);
```

- `voiceCommunication` enables Android's built-in `AcousticEchoCanceler`, `NoiseSuppressor`, and `AutomaticGainControl`.
- `modeInCommunication` optimises the audio pipeline for two-way communication.
- `muteAudio: false` ensures AI speech continues to play through the speaker.

`just_audio` (Siren Playback)

Used in `SafetyMonitor` for the emergency alarm:

```
await _audioPlayer.setAsset('assets/audio/siren.mp3');
await _audioPlayer.setLoopMode(LoopMode.one);
_audioPlayer.play();
```

The siren loops until `_stopAlarm()` calls `_audioPlayer.stop()`, and `Vibration.cancel()` stops the vibration pattern.

Native `AudioTrack` (AI Speech Playback)

The `AudioOutput` class communicates with native Kotlin via `MethodChannel('audio_output_channel')`. The native implementation in

`MainActivity.kt`:

1. Sets `AudioManager.MODE_IN_COMMUNICATION` globally.
2. Creates an `AudioTrack` with `USAGE_VOICE_COMMUNICATION` and `CONTENT_TYPE_SPEECH` at **24 kHz** (Gemini's output sample rate).
3. Attaches `AcousticEchoCanceler` and `NoiseSuppressor` to the audio session.
4. Forces routing to the **built-in loudspeaker** (not earpiece) using `setCommunicationDevice()` on Android 12+ or `isSpeakerphoneOn` on older devices.
5. Maximises `STREAM_VOICE_CALL` volume for blind users.

The `flush` method (for barge-in) executes `audioTrack.pause()` → `audioTrack.flush()` → `audioTrack.play()`.

Firebase AI SDK (`firebase_ai`)

- **Chat mode:** `FirebaseAI.vertexAI().generativeModel(model: 'gemini-2.5-flash')` — standard `generateContent()` with `Content.multi([TextPart, InlineDataPart])`.
- **Live mode:** `FirebaseAI.vertexAI(location: 'us-central1').liveGenerativeModel(model: 'gemini-live-2.5-flash-native-audio')` with `ResponseModalities.audio` and `SpeechConfig(voiceName: 'Kore')`.
- **Custom VAD:** `AccessibleLiveConnector` bypasses the SDK's `connect()` by constructing the WebSocket URI manually and sending a custom setup JSON with `realtime_input_config.automatic_activity_detection`.

`flutter_tts` (Text-to-Speech)

Implemented as a **singleton** (`TTSService._instance`) with:

- Language: `en-US`, speech rate: `0.5`, volume: `1.0`, pitch: `1.0`.
- Duplicate suppression: same message within 2 seconds is skipped (unless `force: true`).
- `awaitSpeakCompletion(true)` enables synchronous speech in `speak()`.

`onnxruntime` (Depth Estimation)

- Model: `assets/models/depth_anything_v2.onnx` (Depth Anything V2, ViT-S variant).
- Input: `Float32List` tensor shaped `[1, 3, 252, 252]` (NCHW). 252 is a multiple of the ViT patch size (14).
- Inference: `OrtSession.fromBuffer(bytes, options)` → `session.runAsync(runOptions, {'input': tensor})`.
- Output: Flattened depth map reshaped to `252×252`, normalised via min-max to `[0.0, 1.0]`.

`google_mlkit_object_detection`

- Configured in **stream mode** with `classifyObjects: true` and `multipleObjects: true`.
- Object position is categorised into `left` / `center` / `right` based on `centerX` relative to image width thirds.
- Relative size = `objectArea / imageArea`, used for proximity estimation when depth is unavailable.
- `isClose` threshold: `relativeSize > 0.10`; `isVeryClose`: `> 0.25`.

`vibration` + `HapticFeedback` (Haptic Feedback)

`VibrationService` attempts the `vibration` plugin first; if it fails, it sets `_useFlutterHaptics = true` and falls back to Flutter's `HapticFeedback` API. Three intensity levels:

Effective Proximity	Method	Pattern
> 0.10 (very close)	`_vibrateHeavy()`	`heavyImpact` × 2 (100 ms gap)
> 0.05 (close)	`_vibrateMedium()`	`mediumImpact` × 1
> 0.01 (detected)	`_vibrateLight()`	`lightImpact` × 1

Approaching objects receive an `intensityBoost` of `0.2` added to the raw proximity value, clamped to `[0.0, 1.0]`.

2.5 Earcon System (Accessibility Cues)

The `LiveScreen` implements non-verbal audio cues for blind users:

Event	Earcon	Implementation
AI starts listening	`_playListeningEarcon()`	`HapticFeedback.lightImpact()` + `SystemSound.play(SystemSoundType.click)`
User releases PTT	`_playProcessingEarcon()`	`HapticFeedback.mediumImpact()` × 2 (100 ms apart)
AI starts responding	`_playResponseEarcon()`	`HapticFeedback.heavyImpact()` + `SystemSound.play(SystemSoundType.click)`

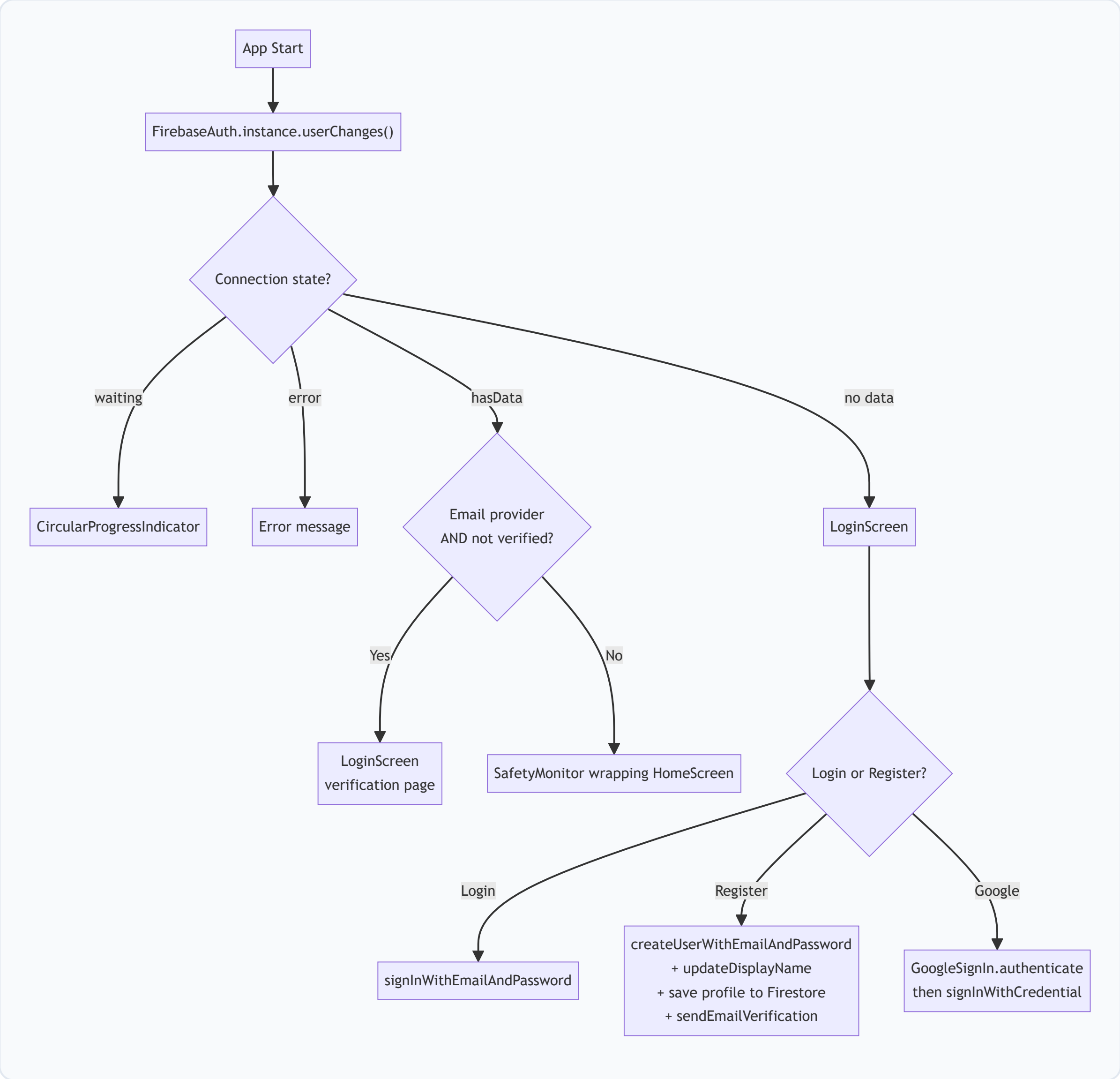
2.6 Traffic Light Monitoring

The `LiveScreen` includes a traffic light monitoring mode that:

1. Sends an initial text prompt asking Gemini to identify the pedestrian crossing signal colour.

2. Re-prompts every **5 seconds** via `Timer.periodic`, instructing Gemini to only speak if the colour has changed.
3. On stop, collects user feedback (was it successful?) and uploads to `Firestore.collection('traffic_feedback')`.

2.7 Authentication Flow



App Start
FirebaseAuth.instance.userChanges()
Connection state?
waiting
error
hasData
no data
CircularProgressIndicator
Error message
Email provider AND not verified?
LoginScreen
Yes
No
LoginScreen verification page
SafetyMonitor wrapping HomeScreen
Login or Register?
Login
Register
Google
signInWithEmailAndPassword
createUserWithEmailAndPassword + updateDisplayName + save profile to Firestore + sendEmailVerification
GoogleSignIn.authenticate then signInWithCredential

2.8 Navigation & Location Awareness

- **Route calculation:** `NavigationService.getRoute()` calls the Google Directions API (walking mode) and parses steps into `NavigationStep` objects.
- **Live tracking:** `Geolocator.getPositionStream(distanceFilter: 5)` updates every 5 metres. `_checkArrivalAtStep()` advances to the next step when within **10 metres** of the current step's end point.
- **Explore mode:** `LocationAwarenessService.startExploring()` tracks position with a 10-metre distance filter. On each update (throttled to 5-second intervals), it calls the Places API (New) `searchNearby` endpoint within a 100-metre radius. New POIs are

announced via TTS, and previously announced names are stored in a `Set<String>` (capped at 50 entries).

Report generated from codebase analysis on 26 February 2026.