Faysal Aziz

Edge AI on Android

On-device vision, local language models and controlled cloud integration

FA-0607 · 2 days

Software Development · Agentic & Generative AI · Data & Analytics

Integrate selected vision and language-model features into an Android learning prototype, then profile resource use and review local/cloud boundaries.

Introduction

This two-day lab is for developers already comfortable with Android and camera/network APIs. It uses prepared project components to integrate a selected camera/vision pipeline and a small supported local language model, with a controlled remote-model path where the learning environment permits.

One small prototype is the main practical deliverable; alternate vision tasks, model conversion and fine-tuning are demonstrations or follow-up topics. Runtimes, quantisation, acceleration and multimodal features depend on the selected model, SDK and physical device. Local execution can reduce data transfer but does not automatically guarantee privacy, low cost or offline operation of every feature.

Learning outcomes

The course teaches participants to:

- Design an Android inference layer with explicit device and cloud boundaries.
- Run a compatible LiteRT vision model and connect it to a selected CameraX pipeline.
- Load and interact with a small supported LiteRT-LM model on compatible hardware.
- Handle asynchronous generation, cancellation, resource cleanup and bounded fallback.
- Use an authenticated backend or suitable provider-supported short-lived mechanism rather than embed a long-lived provider key in the app.
- Profile a learning prototype and explain latency, memory, battery and quality trade-offs.

Prerequisites

- Solid Android app dev (Kotlin / Java, Android SDK, lifecycle, concurrency)
- Experience with Android camera APIs (Camera2, CameraX)
- Familiarity with REST APIs, JSON, networking, threading
- Basic understanding of ML/Neural Networks, inference, quantization (helpful, not mandatory)
- Experience using Android Studio, profiling tools

A compatible physical Android device, adequate storage/memory and approved model files/licences; prepared projects reduce setup work.

Training outline

1. Day 1 — Architecture, LiteRT and Camera Inference

Edge architecture

- Local latency, offline capability, data-transfer, privacy and cost trade-offs; benefits depend on the design.
- Hybrid AI patterns: local-first, fallback, caching
- Resource tradeoffs: CPU / GPU / NPU / memory / battery
- Example app architectures: modular inference layers, model management
- Prepared vision/chat components for one selected prototype and comparison demonstrations.

LiteRT inference

- LiteRT (formerly TensorFlow Lite) and alternative runtime compatibility; select a supported API for the lab.
- Converting models (TensorFlow / Keras → .tflite)
- Post-training/dynamic/integer quantisation concepts; precision and operator support depend on the model/runtime, including any selected low-bit format.
- Loading models, allocating tensors, buffer layout
- Current supported CPU/GPU/NPU acceleration; LiteRT CompiledModel versus compatibility Interpreter paths. Treat deprecated NNAPI as migration context.
- Multiple inputs / outputs, dynamic shapes
- Profiling inference time, memory usage, warm-up strategies
- Handling errors, model fallback

Camera and vision pipeline

- Capturing frames: CameraX, ImageAnalysis, preview, YUV → RGB conversion
- Real-time vs batched inference strategies
- Preprocessing: resizing, normalization, cropping, batching
- Overlay UI: bounding boxes, masks, segment overlays

  Example mini features:

  - Object detection (bounding boxes)
  - Pose / keypoints
  - Segmentation / background removal / filters
- Balancing frame rate and inference cost

Implement one selected task, such as object detection; pose/segmentation examples are comparisons or demonstrations rather than three complete apps.

2. Day 2 — Local LLMs, Hybrid Integration and Profiling

Local language-model integration

- What is feasible on Android (model size, quantization)

  LiteRT-LM Android/Kotlin for the selected model; MediaPipe LLM Inference API is maintenance-only legacy/migration context.

  - Add a compatible selected LiteRT-LM Android dependency and inspect build/runtime requirements.
  - Load a supported model format such as .litertlm and manage its download/storage lifecycle.
  - Create an engine/conversation, handle asynchronous message generation and release resources using the selected SDK.
  - Multimodal input only for explicitly compatible models/configurations; not every small text model accepts images.
- Choose a compatible small model and supported quantisation; Gemma-family examples are version/device-dependent, not a universal recommendation.
- Tokenization, context window, streaming partial results
- Model conversion and LoRA concepts as guided overviews; adapter/model/backend compatibility must be checked and training is not a full two-day objective.
- Fallback for context, speed or capability limits only after checking permitted data sharing; local failure does not automatically authorise cloud transfer.

Controlled remote API path

- API design: prompt format, streaming, chunking, rate limits
- Keep long-lived provider API keys on an authenticated backend, not in an APK/Keystore. Use Android Keystore for appropriate app-local keys/tokens; encryption does not make a shipped provider secret confidential.
- Asynchronous calls, cancellations, partial streaming UI
- Prompt engineering for mobile: template, trimming, summarization
- Explicit local/cloud routing, privacy-aware caching, response handling and bounded fallback.

Combined learning prototype

- App architecture: model manager, inference layer, fallback logic

  Demo app ideas:

  - Illustrative photo/vision plus text-generation prototype; captioning or multimodal generation requires the selected model's support.
  - Local-first assistant with a separately authorised remote path and clear data-transfer expectations.
- UI flows: progress feedback, graceful fallback, streaming UI elements
- Non-sensitive logging, metrics, model versions and verified/authorised model download.

Optimisation and review

- Android Profiler, Memory Profiler, CPU / GPU traces
- Reducing memory footprint: load / unload models, pooling, reuse buffers
- Threading and batching strategies
- Tune only supported accelerators; record CPU fallback and explain deprecated NNAPI compatibility.
- Quantisation/pruning/model-splitting trade-offs as selected demonstrations with quality re-evaluation.
- Heuristics for fallback, caches, summarization
- Edge cases: OOM, overheating, battery mitigation

8. Final Demos, Review & Roadmap

- Participants demo their apps
- Group code review, critique, lessons learned
- Future directions: federated learning, model updates, cross-device inference
- Review approved model sources and conversion tools, licences and runtime compatibility rather than assume all hub models are usable.

- Discuss current Play distribution requirements, asset delivery, model licensing and appropriate user disclosures; no publishing or approval guarantee.

Review one prototype's test evidence, data flows and unresolved limitations; public app distribution remains a planning topic.

© 2026 Faysal Aziz · Kuala Lumpur