

AI-Enabled iOS Apps

On-device vision, language models and reviewed cloud integration

FA-0573 · 2 days

Software Development · Agentic & Generative AI

Build selected iOS AI prototype features using camera inputs, supported on-device runtimes, language-model interfaces and a bounded hybrid architecture.

Introduction

This two-day workshop is for experienced Swift/iOS developers. It connects on-device vision, small language-model options and remote model calls with app architecture, profiling and responsible data handling.

The core exercises use a prepared camera/vision example and a small language or hybrid prototype. Runtime comparisons, quantization and advanced deployment are selected demonstrations. Model/device support, memory, OS versions and framework availability must be confirmed for the lab; not every feature can run on every iPhone.

Local execution and cloud fallback have different data-handling implications. Apps should obtain appropriate user permission before transmitting content and should not embed a reusable developer API secret in the distributed client.

Learning outcomes

The workshop teaches participants to:

- Integrate a suitable on-device inference runtime and inspect model input/output requirements.
- Connect camera frames to a selected vision feature and review performance and overlays.
- Compare local language-model runtimes and available Apple Foundation Models capabilities.
- Connect to a remote model through an appropriate protected service interface.
- Design explicit local/cloud/degraded-mode boundaries and user-visible fallback behaviour.
- Profile memory, latency, threading and resource use in selected prototypes.

- Test AI outputs, data handling and failure cases rather than assume model correctness.

Prerequisites

- Proficiency in Swift (or Swift + some Objective-C)
- Solid experience in UIKit / SwiftUI, app lifecycle, concurrency (GCD, async/await)
- Comfort with Xcode, debugging tools, and asset pipelines
- Basic familiarity with ML/AI concepts: neural networks, model training, inference, quantization (helpful but not strictly required)
- API integration experience (REST / GraphQL)

A Mac with compatible Xcode, a suitable test device and prepared models/libraries. Confirm hardware, OS/framework support, authorised backend access and any usage costs before the workshop.

Training outline

1. Day 1 — On-device vision and camera integration

1. Architecture and constraints

- Edge-AI trade-offs: latency, resources, data handling and offline conditions; local execution alone does not ensure privacy.
- Hybrid AI architectures: on-device vs API vs fallback
- Memory, battery, concurrency trade-offs
- Illustrative app architectures: local versus remote components.
- Use prepared scaffolds for selected vision and language prototype features.

2. LiteRT and runtime choices

Compare LiteRT (formerly TensorFlow Lite), Core ML and available Apple-native options for the selected models.

- Use the supported LiteRT iOS integration for the chosen version; confirm library/delegate compatibility.
- Model conversion and supported quantization: selected demonstration, not a universal 4-bit/8-bit recipe.
- Load the model, allocate tensors, run inference in Swift
- Image preprocessing, normalization, buffer layout
- Supported Metal/Core ML delegates and their model/device limitations.

- Handling multiple inputs/outputs, dynamic shapes
- Debugging and profiling inference time
- Discuss training versus inference and runtime-specific support; on-device training is not a required lab or assumed iOS capability.

3. Camera and vision pipeline

- Capturing frames (AVCaptureSession, sample buffers) and converting to ML input
- Real-time inference vs batch
- Overlay UI: bounding boxes, heat maps, segmentation masks
- Optimizing frame rate and dropping frames gracefully
- Image preprocessing strategies: cropping, resizing, letterboxing

Choose one core vision task; other examples are demonstrations

- Object detection / bounding boxes
- Pose estimation
- Style transfer / filter effects
- Using intermediate output (e.g. landmarks) to drive UI / behavior

2. Day 2 — Language features, hybrid apps and profiling

4. Local language-model choices

- Choose a model that fits the tested device/runtime; compare formats and libraries such as GGUF/llama.cpp where appropriate, without a fixed parameter-count guarantee.
- Converting / quantizing LLM models to mobile formats
- Model loading and memory; CPU/GPU/Neural Engine execution depends on the runtime, exported model and hardware.
- Prompting, context windows, tokenization on device
- Performance tuning, threading, streaming responses
- Integrating with your app UI (chat view, partial streaming)
- Explicit hybrid policy: local failure may trigger a reviewed/consented cloud path or a degraded local response, not silent upload.
- Current LiteRT-LM Swift/iOS route for a supported local language-model example; confirm selected model/backend support.
- Apple Foundation Models and current native model interfaces: availability, selected capabilities and device/OS/language limitations.

5. Protected remote model integration

- API design considerations (rate limits, tokens, cost, privacy)
- Keep service-owned API secrets on a protected backend; Keychain may protect appropriate user credentials but does not make an embedded developer secret safe.
- Streaming responses in Swift async through the selected authenticated service interface.
- Prompt engineering for mobile apps: chunking, context window sliding
- Managing network latency, retries, fallback to local or degraded mode
- Combining local state + remote model: caching, memory summarization

6. A scoped hybrid prototype

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

Sample app idea 1: "Camera Chat"

- Illustrative camera workflow: a supported vision model produces checked output, then an optional reviewed cloud request adds language assistance; captioning requires a suitable model.

Sample app idea 2: "On-device Assistant + Cloud Boost"

- Illustrative assistant workflow: local inference with an explicit, user-visible/authorised cloud option or degraded mode.
- Manage memory: summarization, trimming context
- UI considerations: latency feedback, partial results, "thinking" states
- Logging, metrics, fallbacks, and versioning
- Testing / QA: how to test your AI features (unit, integration, performance)

7. Optimisation and evaluation

- Instruments, Time Profiler, GPU counters
- Memory footprint, model loading/unloading, resource cleanup
- Threading strategies, off-main-thread batching
- Model version/download management, integrity and compatibility checks.
- Quantization and pruning strategies
- Fallback heuristics: when to switch to remote, when to degrade features
- Edge cases: OOMs, overheating, battery drain mitigation
- Privacy & data handling: user data, prompt sanitization, on-device vs cloud tradeoffs

8. Demonstration and next steps

- Participants demo their mini AI apps

- Group code review, common pitfalls discussion
- Further study: larger-model constraints and advanced edge/federated architectures, not a two-day implementation outcome.
- Discuss appropriate model/runtime documentation and next-step learning.
- App distribution considerations: current platform review, privacy disclosures and model/asset licences; approval is not guaranteed.