On-Device LLM Inference Project - Technical Documentation

Your approach and what's unique

The solution focuses on fully on-device LLM inference using a quantized GGUF model, avoiding cloud latency, data egress, and privacy concerns by running everything locally on Android hardware. The approach is intentionally toolchain-driven: start with a known-good model format (GGUF), push it to the device, and integrate a native llama.cpp backend. The plan adapts when third-party Kotlin artifacts prove unstable by switching to two robust routes: a working JNI-based reference app or a direct native (.so) build. This pragmatic pivot ensures delivery speed while preserving maintainability. The final direction leverages Flutter to streamline native bundling through a plugin model, reducing JNI complexity while keeping the core advantages of llama.cpp on-device inference.
What makes it unique:
- Privacy-first, offline-by-default LLM on mobile using open tooling.
- Flexible backend strategy: JNI reference app, direct native build, or Flutter plugin wrapper, chosen dynamically based on artifact availability and time.
- Production-minded file strategy: quick start from shared storage (/sdcard/Download), with a path to one-time copy into app-internal storage for stable mmap and better performance.
- Prompt-format correctness for Phi-3, enabling strong instruction-following without fine-tuning.

Technical stack

- Base inference runtime: llama.cpp (C/C++), for efficient LLM inference and GGUF support.
- Model: Microsoft Phi-3 Mini (pre-quantized GGUF), stored on-device.
- Android build system: Gradle with Kotlin DSL and NDK support for native libraries.
- Optional integration stacks:
- Kotlin Android app with JNI bindings or a community binding when reliably available.
- Flutter app with a llama.cpp plugin to simplify native integration across platforms.
- Android tooling: ADB, Platform-Tools, device debugging, and storage access best practices.
- UI:
- Kotlin path: Jetpack Compose minimal UI for chat.
- Flutter path: Standard Flutter widgets for a simple prompt/response chat experience.

Architecture

- Mobile device layers:
- Storage: GGUF model file (initially /sdcard/Download, optionally copied to app-internal storage).
- Native inference core: llama.cpp compiled for arm64-v8a.
- Bridge:
- Kotlin/Android path: JNI or community binding exposing load/generate parameters (context size, threads, sampling).
- Flutter path: Dart plugin calling native methods (create/load/generate/close).
- App layer: UI for prompt input and streaming/collected output, with optional local persistence (Room/SQLite or Flutter local storage) for conversation context or personalization.
- Data flow:

- User prompt → Prompt template formatter (Phi-3: <|user|>…<|end|><|assistant|>) → llama.cpp generation → token stream/complete text → UI display → optional conversation store.
- Performance controls:
- Threads capped based on big.LITTLE core availability (e.g., 4–8).
- Context length tuned to device RAM (e.g., 2k for Q4_K_M).
- Quantization profile chosen for size/perf balance (Q4_K_M initial).

Implementation details

- Model preparation:
- Download pre-converted Phi-3 Mini GGUF.
- Push to device via adb; path referenced directly by the app.
- Android project setup (Kotlin path):
- repositories in settings.gradle(.kts), not module files.
- ndk.abiFilters set to arm64-v8a.
- Manifest with MAIN/LAUNCHER and exported activity for Android 12+.
- Minimal LLM manager encapsulating init/generate/close.
- JNI/native fallback:
- Build llama.cpp as .so for arm64-v8a with CMake/NDK.
- Package into jniLibs or via externalNativeBuild.
- Expose JNI methods: create/load, generate, release.
- Flutter pivot:
- Install Flutter SDK; flutter doctor to validate Android toolchain.
- Create Flutter app; add llama.cpp plugin (or wrap llama.cpp as a custom plugin).
- Dart API: create(context), generate(prompt, params), close.
- Pass absolute model path; use mmap on native side.

Installation instructions

- Android prerequisites:
- Android Studio and SDK Platform-Tools on PATH.
- Device with USB debugging enabled and authorized.
- Model:
- Download Phi-3 Mini GGUF locally.
- adb push "path\to\Phi-3-mini-4k-instruct-q4.gguf" /sdcard/Download/phi3-mini-q4.gguf
- Kotlin project (if using native route):
- Create Android project (Empty Activity), minSdk 24+, Kotlin DSL.
- In settings.gradle(.kts), define repositories (google(), mavenCentral()).
- In app’s Gradle, set ndk { abiFilters += listOf("arm64-v8a") }.
- Build/run “Hello World” to device.
- Integrate JNI/cmake or a working reference JNI app structure.
- Flutter project (recommended for faster native bundling):
- Install Flutter SDK; set PATH.
- flutter doctor; flutter doctor --android-licenses.
- flutter create ondevice_flutter; flutter run to device.
- Add llama.cpp plugin; configure Android ABI if required; rebuild.
- Reference model path in Dart code; run generation.

User guide

- Launch the app on the device.
- Ensure the model file exists at the configured path (e.g., /sdcard/Download/phi3-mini-q4.gguf).
- On the first screen, tap “Generate” (or open chat screen) to submit a prompt.
- Input guidance:
- The app formats prompts for Phi-3 automatically (<|user|>…<|end|><|assistant|>).
- Start with short prompts to verify performance; increase context gradually.
- Advanced:
- Adjust threads (start 4–6) and n_ctx (e.g., 2048) for the device.
- For better performance and stability, relocate the model to app-internal storage on first run (one-time copy), then reference that internal path.
- Troubleshooting:
- If “file not found”: verify the exact path and filename, including case.
- If native load error: confirm arm64-v8a ABI and the .so packaging; rebuild if needed.
- If slow or OOM: lower n_ctx, reduce max tokens, or use a tighter quantization.

Salient features

- Fully offline, on-device LLM inference with a popular GGUF model.
- Flexible backend integration: JNI/native or Flutter plugin, depending on team preference and time constraints.
- Production-minded file strategy: easy initial path from shared storage, with a path to internal storage for stable mmap.
- Configurable inference controls: threads, context size, sampling parameters.
- Clean prompt formatting for Phi-3 instruction following.
- Clear, stepwise build and deployment flow validated on real hardware.

If desired, this can be extended with:
- Streaming token output to UI.
- Conversation storage (Room/SQLite or Flutter local DB).
- Background worker for one-time model copy into internal storage.
- Device capability probing to auto-tune threads and context length.