

# Offline LLM with llama.cpp — Capacitor Plugin Integration & App Configuration (SproutChat)

**Goal:** Integrate and configure the **capacitor-offline-llm** plugin that embeds **llama.cpp** so SproutChat runs **fully offline** on Android and iOS.

**Order of operations:** 1) Plugin integration → 2) App configuration → 3) Platform linking → 4) On-device validation.

---

## 0) Repo map (what we rely on)

From your repo root: [GitHub](#)

- `android/` — Capacitor Android project
- `ios/` — Capacitor iOS workspace
- `scripts/` — helper scripts (we'll use `fetch-llama.sh`) [GitHub](#)
- `src/` — app code (we'll call the headless runtime from here)
  - `src/offlineRuntime.ts` — headless wrapper that talks to the plugin [GitHub](#)
- `capacitor.config.ts` — Capacitor config (used when syncing) [GitHub](#)
- `package.json` — dependency + scripts (we'll wire postinstall + cap sync) [GitHub](#)

If your plugin folder already exists at `plugins/capacitor-offline-llm/`, great. If not, create it (or vendor it via a local path dependency as shown below).

---

## 1) Integrate the plugin (TypeScript surface + native stubs)

## 1.1 Add the plugin as a local dependency

In `package.json`, ensure the dependency (local file path) and scripts exist:

```
{
  "dependencies": {
    // ...
    "capacitor-offline-llm": "file:./plugins/capacitor-offline-llm"
  },
  "scripts": {
    "postinstall": "bash ./scripts/fetch-llama.sh",
    "cap:sync:android": "npx cap sync android",
    "cap:sync:ios": "npx cap sync ios"
  }
}
```

- The local file path ensures your app consumes your plugin source directly.
- `postinstall` pulls native llama binaries into the plugin during `npm install`.

Confirm `fetch-llama.sh` exists at `scripts/fetch-llama.sh`. It should download:

- Android:  
`plugins/capacitor-offline-llm/android/src/main/jniLibs/arm64-v8a/libllama.so`
- iOS:  
`plugins/capacitor-offline-llm/ios/llama/lib/arm64/libllama.a` [GitHub](#)

## 1.2 Verify the app's headless wrapper exists

Check `src/offlineRuntime.ts` is present and exports an API that calls the plugin: `ensureModel`, `loadModel`, `generate`, `stop/stopInference`, `unload`, `clearContext`, etc. [GitHub](#)

If missing, create it. Example minimal wrapper:

```

// src/offlineRuntime.ts
import { Capacitor } from '@capacitor/core';
import { OfflineLLMRuntime } from 'capacitor-offline-llm'; // plugin
TS typings

const notNative = () => { throw new Error('Offline LLM not available
on web'); };

export async function ensureModel(url: string, filename: string,
sha256?: string) {
  if (!Capacitor.isNativePlatform()) return notNative();
  return OfflineLLMRuntime.ensureModel({ url, filename, sha256 });
}
export async function loadModel(opts: { path: string; nCtx?: number;
nThreads?: number; useMetal?: boolean }) {
  if (!Capacitor.isNativePlatform()) return notNative();
  return OfflineLLMRuntime.loadModel(opts);
}
export async function generate(
  opts: { prompt: string; maxTokens?: number; temperature?: number;
topP?: number; topK?: number; repeatPenalty?: number },
  onToken: (t: string) => void,
  onDone: (stats: any) => void
) {
  if (!Capacitor.isNativePlatform()) return notNative();
  const sub = OfflineLLMRuntime.addListener('generationProgress', (e:
any) => {
    if (e?.token) onToken(e.token);
    if (e?.done) { onDone(e.stats); sub.remove(); }
  });
  return OfflineLLMRuntime.generate(opts);
}
export async function stop() { return
OfflineLLMRuntime.stopInference(); }
export async function unload() { return
OfflineLLMRuntime.unloadModel(); }
export async function listDownloadedModels() { return
OfflineLLMRuntime.listDownloadedModels(); }

```

This file is your **SDK** for the app. No UI required.

---

## 2) Configure the app to use the plugin

### 2.1 Feature flag

Create/append `.env` at the project root:

```
VITE_OFFLINE_LLM=1
```

Use it to guard offline code paths if needed:

```
// e.g., src/somewhere.ts
export const OFFLINE_ENABLED = import.meta.env.VITE_OFFLINE_LLM ===
'1';
```

### 2.2 Native binary URLs & integrity

Export before `npm install` so the postinstall script can fetch natives:

```
export LLAMA_SO_URL="https://<your-cdn>/libllama.so"    # Android
arm64-v8a
export LLAMA_A_URL="https://<your-cdn>/libllama.a"      # iOS arm64
# Optional (recommended)
export LLAMA_SO_SHA256="<sha256-of-so>"
export LLAMA_A_SHA256="<sha256-of-a>"
```

Run install + sync:

```
npm install
npm run cap:sync:android
npm run cap:sync:ios
```

✅ **Verify** the binaries exist under the plugin after `npm install`:

- `plugins/capacitor-offline-llm/android/src/main/jniLibs/arm64-v8a/libllama.so`
  - `plugins/capacitor-offline-llm/ios/llama/lib/arm64/libllama.a`
- 

## 3) Platform linking details

### 3.1 Android (JNI/CMake/Gradle)

Confirm these bits inside your plugin's Android subtree:

**CMake:** (e.g.,

`plugins/capacitor-offline-llm/android/src/main/cpp/CMakeLists.txt`)

```
add_library(llama SHARED IMPORTED)
set_target_properties(llama PROPERTIES IMPORTED_LOCATION
    ${CMAKE_CURRENT_SOURCE_DIR}/../jniLibs/${ANDROID_ABI}/libllama.so)

target_link_libraries( # your plugin target
    offline_llm_plugin
    llama
    log # etc.
)
```

**Gradle ABI filter** (module-level `build.gradle`):

```
android {
    defaultConfig { /* ... */ }
    ndk { abiFilters "arm64-v8a" }
}
```

Open Android Studio from the repo root:

```
npm run android
```

Run on a **physical ARM64 device** (or ARM64 emulator image).

**Logcat clues on success:**

```
LlamaBridge: llama.cpp runtime initialized  
LlamaBridge: Model loaded (ctx=2048)  
LlamaBridge: Streaming tokens...
```

### 3.2 iOS (Swift/ObjC++/Podspec)

Check your plugin Podspec includes:

```
s.vendored_libraries = 'ios/llama/lib/arm64/libllama.a'  
s.frameworks = 'Accelerate'  
s.weak_frameworks = 'Metal', 'MetalKit'
```

Open Xcode:

```
npm run ios
```

Then **Product** → **Clean Build Folder** → **Build** → **Run** (physical device).

**Xcode Debug output on success:**

```
[OfflineLLMRuntimePlugin] llama.cpp context initialized  
[OfflineLLMRuntimePlugin] token: "Hello"  
[OfflineLLMRuntimePlugin] done: { tokens: 64, tps: 26.1 }
```

**If you see “Undefined symbols: llama\_\*”** — your `libllama.a` wasn’t built with the C API. Rebuild it properly, re-host, re-run `npm install`.

---

## 4) First run (headless, no UI changes)

Create a tiny dev script to validate everything:

```
// scripts/dev-offline.ts  
import { ensureModel, loadModel, generate, unload } from  
'../src/offlineRuntime';
```

```

(async () => {
  const url = 'https://<your-model-cdn>/LLaMA-2-7B-Q4_K_M.gguf';
  const name = 'LLaMA-2-7B-Q4_K_M.gguf';

  const path = await ensureModel(url, name);          // 1) download &
cache
  await loadModel({ path, nCtx: 2048, nThreads: 4, useMetal: true });
  // 2) init llama.cpp

  await generate(                                     // 3) stream
tokens
    { prompt: "Explain gravity in one sentence.", maxTokens: 64,
temperature: 0.7 },
    t => console.log("token:", t),
    stats => console.log("done:", stats)
  );

  await unload();                                     // 4) cleanup
})();

```

Run the app on device, then run the script from your dev console/workflow (or wire it into your app's logic for a quick smoke test).

---

## 5) Definitive offline proof (airplane mode)

1. Ensure the model is downloaded (first run).
2. **Enable Airplane Mode** on the device.
3. Call `loadModel()` then `generate()` again.

✅ You should still get tokens streaming.  
No network calls in logs. That proves **pure on-device inference**.

---

## 6) Performance tips (mobile defaults)

- **Threads:** start with `nThreads = big_cores` (often 4–6).
- **Context:** `nCtx = 2048` (raise only if needed; memory grows fast).
- **Models:** use **GGUF quantized** (Q4\_K\_M/Q5\_K\_M) to balance speed & quality.
- **Metal (iOS):** `{ useMetal: true }` usually yields 2–3× higher TPS than CPU.

Measure tokens/sec:

```
const t0 = performance.now();
await generate({ prompt: "...", () => {}, s => {
  const tps = s.tokens / ((performance.now() - t0) / 1000);
  console.log("Tokens/sec:", tps);
}});
```

---

## 7) Troubleshooting (fast mapping)

Symptom	Likely cause	Fix
<code>UnsatisfiedLinkError</code> (Android)	<code>.so</code> missing/wrong path or x86 emulator	Ensure <code>jniLibs/arm64-v8a/libllama.so</code> , run on ARM64
<code>Undefined symbols: llama_*</code> (iOS)	<code>.a</code> built without C API	Rebuild <code>libllama.a</code> exporting llama C API
No token stream	Not listening to event, or generation not started	Ensure <code>generationProgress</code> listener; check <code>generate()</code> call
Very slow on iOS	CPU path active	Pass <code>{ useMetal: true }</code>
OOM / crashes on load	Model too big / ctx too high	Use smaller GGUF / lower <code>nCtx</code>
Network calls during gen	Model not cached	Use <code>ensureModel()</code> first; then go offline

---



## 8) CI/CD & auto-push

- Do not commit `.so/.a` into Git.

Host binaries, set env vars in CI, and rely on `postinstall`:

```
export LLAMA_SO_URL="https://cdn/libllama.so"
export LLAMA_A_URL="https://cdn/libllama.a"
npm ci
npm run cap:sync:android
npm run cap:sync:ios
```

- - Then build with Gradle/Xcode as normal.
- 

## 9) Final verification checklist (copy/paste)

- `capacitor-offline-llm` listed in `package.json` as local dependency
  - `.env` has `VITE_OFFLINE_LLM=1`
  - `LLAMA_SO_URL` / `LLAMA_A_URL` set before `npm install`
  - `npm install` downloaded natives into the plugin
  - `npm run cap:sync:android` and `npm run cap:sync:ios` succeeded
  - Android Studio build on **ARM64** device succeeds; Logcat prints llama init + tokens
  - Xcode build on **ARM64** device succeeds; Debug console prints llama init + tokens
  - Airplane mode → tokens still stream (no network)
  - Final stats show tokens/sec
-

## 10) Notes that are specific to this repo

- The headless entry-point is **already** in `src/offlineRuntime.ts` (you call this from anywhere; no UI change required). [GitHub](#)
  - The fetch script lives in `scripts/fetch-llama.sh`; we rely on it to place binaries into the plugin subtrees on install. [GitHub](#)
  - Capacitor workspace files (`android/`, `ios/`) are present in the repo; syncing registers the plugin with those native projects. [GitHub](#)
  - Capacitor config is at `capacitor.config.ts` (used by sync). [GitHub](#)
  - Your `package.json` exists at repo root; ensure the local plugin dependency + scripts shown above. [GitHub](#)
- 

### That's it

Follow this guide **in order** and you'll end up with a **fully offline, on-device LLM** inside SproutChat, powered by **llama.cpp**, verifiably running on Android and iOS devices (including in airplane mode). If anything fails, use the troubleshooting table and the platform-specific logs to locate the exact gap.

Du sa:

That's only 9 steps????

ChatGPT sa:

Totally fair — here's a **full end-to-end runbook with 24 concrete steps** (integration → configuration → platform linking → validation → performance → CI → rebuilds). Copy it straight into your repo as `docs/OFFLINE_LLM_PLUGIN_INTEGRATION.md`.

---

## SproutChat + llama.cpp (Capacitor) — Master Integration & Configuration Runbook

**Goal:** Make SproutChat run **fully offline** using your **capacitor-offline-llm** plugin (llama.cpp inside Android/iOS).

**Order:** Integration → App config → Platform linking → Validation → Performance  
→ CI/CD → Rebuilds.

---

## Integration (Steps 1–7)

### 1) Verify repo layout (must exist)

```
/android
/ios
/src
/scripts/fetch-llama.sh
/plugins/capacitor-offline-llm    (TS bridge + Android + iOS)
package.json
capacitor.config.ts
```

### 2) Add local plugin dependency

In **package.json**:

```
{
  "dependencies": {
    "capacitor-offline-llm": "file:./plugins/capacitor-offline-llm"
  },
  "scripts": {
    "postinstall": "bash ./scripts/fetch-llama.sh",
    "cap:sync:android": "npx cap sync android",
    "cap:sync:ios": "npx cap sync ios",
    "android": "npx cap open android",
    "ios": "npx cap open ios"
  }
}
```

### 3) Create/confirm headless SDK wrapper

**src/offlineRuntime.ts**:

```

import { Capacitor } from '@capacitor/core';
import { OfflineLLMRuntime } from 'capacitor-offline-llm';

const notNative = () => { throw new Error('Offline LLM not available
on web'); };

export async function ensureModel(url:string, filename:string,
sha256?:string){
  if(!Capacitor.isNativePlatform()) return notNative();
  return OfflineLLMRuntime.ensureModel({ url, filename, sha256 });
}
export async function
loadModel(opts:{path:string;nCtx?:number;nThreads?:number;useMetal?:bo
olean}){
  if(!Capacitor.isNativePlatform()) return notNative();
  return OfflineLLMRuntime.loadModel(opts);
}
export async function generate(

opts:{prompt:string;maxTokens?:number;temperature?:number;topP?:number
;topK?:number;repeatPenalty?:number},
  onToken:(t:string)=>void,
  onDone:(stats:any)=>void
){
  if(!Capacitor.isNativePlatform()) return notNative();
  const sub = OfflineLLMRuntime.addListener('generationProgress',
(e:any)=>{
    if(e?.token) onToken(e.token);
    if(e?.done){ onDone(e.stats); sub.remove(); }
  });
  return OfflineLLMRuntime.generate(opts);
}
export const stop    = () => OfflineLLMRuntime.stopInference();
export const unload  = () => OfflineLLMRuntime.unloadModel();
export const clearContext = () => OfflineLLMRuntime.clearContext?();
export const listDownloadedModels = () =>
OfflineLLMRuntime.listDownloadedModels?();

```

#### 4) Add runtime flag (optional)

`src/runtimeMode.ts:`

```
export const OFFLINE_ENABLED = import.meta.env.VITE_OFFLINE_LLM ===  
'1';
```

#### 5) Prepare fetch script (binaries downloader)

`scripts/fetch-llama.sh` must:

- Read `LLAMA_SO_URL/LLAMA_A_URL` (+ optional SHA256)
- `mkdir -p` target dirs
- Download to:
  - `plugins/capacitor-offline-llm/android/src/main/jniLibs/arm64-v8a/libllama.so`
  - `plugins/capacitor-offline-llm/ios/llama/lib/arm64/libllama.a`
- Exit non-zero on checksum mismatch

#### 6) Set environment & flag

```
echo "VITE_OFFLINE_LLM=1" >> .env
```

```
export LLAMA_SO_URL="https://<your-cdn>/libllama.so"  
export LLAMA_A_URL="https://<your-cdn>/libllama.a"  
# (Optional, recommended)  
export LLAMA_SO_SHA256="<sha256-so>"  
export LLAMA_A_SHA256="<sha256-a>"
```

#### 7) Install & sync

```
npm install  
npm run cap:sync:android  
npm run cap:sync:ios
```

**Verify** after install:

```
plugins/capacitor-offline-llm/android/src/main/jniLibs/arm64-v8a/libllama.so
plugins/capacitor-offline-llm/ios/llama/lib/arm64/libllama.a
```

---

## App Configuration (Steps 8–10)

### 8) (Optional) Gate usage by flag

Use `OFFLINE_ENABLED` when wiring any app logic to offline runtime.

### 9) Prepare a dev smoke test

`scripts/dev-offline.ts`:

```
import { ensureModel, loadModel, generate, unload } from
'../src/offlineRuntime';

(async()=>{
  const modelUrl = 'https://<your-model-cdn>/LLaMA-2-7B-Q4_K_M.gguf';
  const modelName = 'LLaMA-2-7B-Q4_K_M.gguf';

  const path = await ensureModel(modelUrl, modelName);
  await loadModel({ path, nCtx: 2048, nThreads: 4, useMetal: true });
  await generate(
    { prompt: "Explain gravity in one sentence.", maxTokens: 64,
    temperature: 0.7, topP: 0.9, topK: 40, repeatPenalty: 1.1 },
    t => console.log("token:", t),
    s => console.log("done:", s)
  );
  await unload();
})();
```

### 10) Confirm no UI dependency

No UI components need to import the plugin. You can keep all usage headless or wire it where you need.

---

## Android Linking (Steps 11–13)

### 11) CMake linkage (JNI)

plugins/capacitor-offline-llm/android/src/main/cpp/CMakeLists.txt:

```
add_library(llama SHARED IMPORTED)
set_target_properties(llama PROPERTIES IMPORTED_LOCATION
    ${CMAKE_CURRENT_SOURCE_DIR}/../jniLibs/${ANDROID_ABI}/libllama.so)

target_link_libraries(offline_llm_plugin PRIVATE llama log)
set(CMAKE_CXX_STANDARD 17)
```

### 12) Gradle ABI filter

Module build.gradle:

```
android {
    defaultConfig { /* ... */ }
    ndk { abiFilters "arm64-v8a" }
}
```

### 13) Build & run in Android Studio

```
npm run android
```

Run on **ARM64** physical device (or ARM64 emulator).

**Logcat success signatures:**

```
LlamaBridge: llama.cpp runtime initialized
LlamaBridge: Model loaded (ctx=2048)
LlamaBridge: Streaming tokens...
```

---

## iOS Linking (Steps 14–16)

## 14) Podspec & frameworks

`plugins/capacitor-offline-llm/ios/CapacitorOfflineLlm.podspec:`

```
s.vendored_libraries = 'ios/llama/lib/arm64/libllama.a'
s.frameworks = 'Accelerate'
s.weak_frameworks = 'Metal', 'MetalKit'
```

## 15) Build & run in Xcode

```
npm run ios
```

**Xcode** → Product → Clean Build Folder → Build → Run (physical device).

## 16) Debug console success lines

```
[OfflineLLMRuntimePlugin] llama.cpp context initialized
[generationProgress] token: "Hello"
[generationProgress] done: { tokens: 64, tps: 26.1 }
```

If `Undefined symbols: llama_*` → rebuild `libllama.a` with C API exports.

---

## Validation (Steps 17–18)

### 17) Headless smoke test

With app running on device, execute `node scripts/dev-offline.ts` (or trigger code path).

**Expect:** streamed tokens + final stats.

### 18) Airplane mode proof

1. Ensure model cached (first run online).
2. Turn **Airplane Mode ON**.
3. Call `loadModel()` and `generate()` again.  
**Expect:** tokens still stream; no network calls in logs.



---

## ⚡ Performance & Tuning (Steps 19–20)

### 19) Measure tokens/sec

```
const t0 = performance.now();
await generate({ prompt:"...", maxTokens:128 }, ()=>{}, stats=>{
  const tps = stats.tokens / ((performance.now()-t0)/1000);
  console.log("Tokens/sec:", tps);
});
```

### Recommendations

- `nThreads = big_cores` (4–6 typical)
- `nCtx = 2048` to start; raise cautiously
- Prefer **GGUF Q4\_K\_M / Q5\_K\_M**
- iOS: `{ useMetal: true }`  $\approx$  2–3× faster than CPU

### 20) Model suggestions (choose one to start)

- **LLaMA 2 7B Q4\_K\_M** (~4GB) — good mobile baseline
- **LLaMA 3 8B Q4\_K\_M** (~5GB) — newer arch, heavier
- **LLaMA 1–3B Q4\_K\_M** (<1–2GB) — quick validation

---

## Troubleshooting (Step 21)

Problem	Likely Cause	Fix
<code>UnsatisfiedLinkError</code> (Android)	Wrong ABI or missing <code>.so</code>	Ensure <code>jniLibs/arm64-v8a/libllama. so</code> , ARM64 device

Undefined symbols: llama_* (iOS)	.a lacks C API	Rebuild libllama.a exporting llama C symbols
No tokens stream	Listener missing / generation not started	Ensure generationProgress listener is attached
Slow iOS	CPU fallback	Use { useMetal:true } in loadModel()
OOM or crash on load	Model too large / ctx too high	Use smaller GGUF; lower nCtx
Network during gen	Model not cached	Call ensureModel() first; then offline test

---

## CI/CD & Auto-Push (Step 22)

- **Never commit .so/.a.** Fetch at install time.
- In CI:

```
export LLAMA_SO_URL="https://cdn/libllama.so"
export LLAMA_A_URL="https://cdn/libllama.a"
export LLAMA_SO_SHA256="..."
export LLAMA_A_SHA256="..."
```

```
npm ci
npm run cap:sync:android
npm run cap:sync:ios
```

```
# your build commands here (gradle/xcodebuild/fastlane)
```

---

## Rebuild Binaries (Step 23)

Android (**libllama.so**)

- Build llama.cpp for **arm64-v8a**, export C API, **-O3**.
- Place at:

```
plugins/capacitor-offline-llm/android/src/main/jniLibs/arm64-v8a/  
libllama.so
```

- Host and update **LLAMA\_SO\_URL**, reinstall.

## iOS (**libllama.a**)

- Build static library for **arm64** with C API; link **Accelerate**.
- (Optional) enable **Metal** (**-DGGML\_METAL**, include **.metal** sources).
- Place at:  

```
plugins/capacitor-offline-llm/ios/llama/lib/arm64/libllama.a
```
- Host and update **LLAMA\_A\_URL**, reinstall.

---

## Final Production Gate (Step 24)

Ship when all are true:

- ☒ **npm install** fetched natives successfully
- ☒ **cap:sync:\*** completed
- ☒ Android/iOS builds pass on physical devices
- ☒ Tokens stream in logs with **airplane mode ON**
- ☒ Final stats show TPS > 10 for your chosen model
- ☒ No network requests during generation
- ☒ Memory stable across **loadModel** → **generate** → **unload** cycles

---

## Appendix — Log snippets to look for

### Android (Logcat)

```
LlamaBridge: llama.cpp runtime initialized  
LlamaBridge: context=2048, threads=4  
LlamaBridge: token="The"  
LlamaBridge: token=" quick"  
LlamaBridge: done { tokens: 64, ms: 3100, tps: 20.6 }
```

### iOS (Xcode Debug)

```
[OfflineLLMRuntimePlugin] llama.cpp context ready (ctx=2048,  
threads=4)  
[generationProgress] token: "Hello"  
[generationProgress] done: { tokens: 64, ms: 2500, tps: 25.6 }
```