Guide: Compiling whisper.cpp for Flutter FFI on Android

This document provides the definitive, step-by-step process for compiling the whisper.cpp C++ library into a standalone shared library (.so) that is compatible with Dart's Foreign Function Interface (FFI) for use in a Flutter application on Android.

1. The Core Concept: FFI vs. JNI

The most common mistake is to use the official whisper.android example project. That project is designed to build a **Java Native Interface (JNI)** library, which is meant to be called from Kotlin/Java code.
For Flutter, we need a generic, standalone **C-style shared library**. Dart FFI calls C functions directly, not Java/Kotlin methods. Therefore, we must compile the library from the command line using the Android NDK toolchain, bypassing Android Studio's JNI-focused build process entirely.

2. Prerequisites

Ensure you have the following tools installed and accessible from your terminal.

- **Git:** For cloning and updating the whisper.cpp repository.
- **Android NDK:** The toolkit for compiling C++ for Android. Make sure you know the path to its installation directory (e.g., C:\Users\YourUser\AppData\Local\Android\Sdk\ndk\<version>).
- **CMake:** The build system generator used by whisper.cpp.
- **Ninja:** A small, fast build system that works with CMake and the NDK. The easiest way to install it on Windows is with winget.
  # Run this in PowerShell
  winget install -e --id Ninja-build.Ninja

3. The Compilation Process

Follow these steps from a terminal (like PowerShell) in the root directory of your whisper.cpp project.

Step 1: Update the Repository

Always start by getting the latest stable code from the developers.
git pull origin master

Step 2: Create a Clean Build Directory

This ensures there are no old or conflicting files from previous build attempts.
If a build-android folder already exists, delete it first
rm -r build-android -Force
mkdir build-android
cd build-android

Step 3: Configure the Build with CMake

This is the most critical step. Run the following command, making sure to **replace the NDK path** with the correct path for your system.
cmake .. -G "Ninja"
-DCMAKE_TOOLCHAIN_FILE="C:\Users\gombi\AppData\Local\Android\Sdk\ndk\27.0.12077973\build\cmake\android.toolchain.cmake" -DANDROID_ABI="arm64-v8a"
-DANDROID_PLATFORM=android-24 -DBUILD_SHARED_LIBS=ON
-DWHISPER_BUILD_EXAMPLES=OFF -DWHISPER_BUILD_TESTS=OFF

Command Breakdown:

- -G "Ninja": **Crucial.** Forces CMake to use the Ninja build system instead of the default (Visual Studio).
- -DCMAKE_TOOLCHAIN_FILE: **Crucial.** Points to the NDK's configuration file for cross-compiling to Android.
- -DANDROID_ABI="arm64-v8a": Specifies the target processor architecture for modern 64-bit Android phones.
- -DANDROID_PLATFORM=android-24: Sets a compatible Android API level.
- -DBUILD_SHARED_LIBS=ON: **Crucial.** This is the master switch that tells CMake to build .so files instead of static libraries.
- -DWHISPER_BUILD_EXAMPLES/TESTS=OFF: Speeds up the build by not compiling the extra example and test programs.

Step 4: Run the Build

Once configuration is successful, compile the code.
cmake --build .

Step 5: Verify the Output

After the build finishes, navigate to the output directory (build-android\src\) and verify that the C functions have been correctly exported.
Replace with your NDK path
C:\Users\gombi\AppData\Local\Android\Sdk\ndk\27.0.12077973\toolchains\llvm\prebuilt\windows-x86_64\bin\llvm-objdump.exe -T libwhisper.so | findstr whisper_init_from_file_with_params

A **successful output** will show the function name, confirming it's a valid FFI library:

000000000006709c g   DF .text 0000000000000d0
whisper_init_from_file_with_params

4. Integrating with Your Flutter Project

Step 1: Copy the Core Libraries

Copy the newly compiled libraries into your Flutter project.

- From build-android\src\, copy libwhisper.so.
- From build-android\ggml\src\, copy libggml.so.

Paste them into: your_flutter_project\android\app\src\main\jniLibs\arm64-v8a\

Step 2: Copy the libomp Dependency

whisper.cpp requires the OpenMP library for multi-threading. You must copy this from the NDK.

- **Find it at:** C:\Users\gombi\AppData\Local\Android\Sdk\ndk\<version>\toolchains\llvm\prebuilt\windows-x86_64\lib64\clang\<version>\lib\linux\aarch64\libomp.so
- **Paste it into:** your_flutter_project\android\app\src\main\jniLibs\arm64-v8a\

Your jniLibs/arm64-v8a folder should now contain at least libwhisper.so, libggml.so, and libomp.so.

Step 3: Clean and Run

In your Flutter project terminal, run flutter clean and then flutter run.

5. How to Update the Library in the Future

The process is simple:

1. In your whisper.cpp directory, run git pull origin master to get the latest C++ code.
2. Delete your old build-android folder.
3. Follow the compilation process from **Section 3** again.
4. Copy the new .so files into your Flutter project.