**Tema 6 – LlamaC++**

**Plugins configurados**

plugins {

alias(libs.plugins.android.application)

alias(libs.plugins.kotlin.android)

alias(libs.plugins.kotlin.compose)

}

* android.application: Plugin base para aplicaciones Android.
* kotlin.android: Soporte para Kotlin en Android.
* kotlin.compose: Plugin específico para Jetpack Compose (la moderna UI toolkit de Android).

**Configuración principal de Android**

android {

namespace = "com.example.myllamaapp"

compileSdk = 35

* Define el namespace de la aplicación y utiliza el SDK más reciente (API 35)

**Configuración de la aplicación**

defaultConfig {

applicationId = "com.example.myllamaapp"

minSdk = 35

targetSdk = 35

*// ...*

}

* minSdk = 35: Requiere Android API 35+ (muy restrictivo, limita compatibilidad).
* targetSdk = 35: Optimizado para las últimas características de Android.

**Configuración nativa C++**

externalNativeBuild {

cmake {

cppFlags("-std=c++17", "-fexceptions")

arguments(

"-DANDROID\_STL=c++\_shared",

"-DLLAMA\_NATIVE=OFF",

"-DLLAMA\_ARM\_NEON=ON",

"-DLLAMA\_K\_QUANTS=ON"

)

}

}

Esta sección es crucial para la integración con LLaMA.cpp:

* C++17: Estándar moderno de C++.
* LLAMA\_ARM\_NEON=ON: Optimizaciones SIMD para procesadores ARM.
* LLAMA\_K\_QUANTS=ON: Habilita cuantización K-quants para modelos más eficientes.
* c++\_shared: Biblioteca estándar compartida de C++.

**Arquitecturas soportadas**

ndk {

abiFilters += listOf("arm64-v8a","armeabi-v7a")

}

* arm64-v8a: Procesadores ARM de 64 bits (modernos).
* armeabi-v7a: Procesadores ARM de 32 bits (compatibilidad).

**Integración CMake**

externalNativeBuild {

cmake {

path = file("CMakeLists.txt")

version = "3.22.1"

}

}

**Dependencias principales**

dependencies {

// Core Android con Compose

implementation(libs.androidx.core.ktx)

implementation(libs.androidx.activity.compose)

implementation(platform(libs.androidx.compose.bom))

// Arquitectura moderna

implementation("androidx.lifecycle:lifecycle-viewmodel-compose:2.8.7")

implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.7.3")

// Gestión de permisos

implementation("com.google.accompanist:accompanist-permissions:0.34.0")

}

**Puntos destacados:**

* ViewModel-Compose: Gestión de estado reactiva
* Coroutines: Procesamiento asíncrono (esencial para LLMs)
* Accompanist-permissions: Manejo moderno de permisos de Android

Código **build.gradle.kts(Module:app)**:

plugins **{** alias(*libs*.*plugins*.*android*.*application*)  
 alias(*libs*.*plugins*.*kotlin*.*android*)  
 alias(*libs*.*plugins*.*kotlin*.*compose*)  
**}***android* **{** namespace = "com.example.myllamaapp"  
 compileSdk = 35  
  
 defaultConfig **{** applicationId = "com.example.myllamaapp"  
 minSdk = 35  
 targetSdk = 35  
 versionCode = 1  
 versionName = "1.0"  
  
 testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"  
  
 *// Configuración de opciones de compilación Java* javaCompileOptions **{** annotationProcessorOptions **{** arguments += *mapOf*("room.incremental" *to* "true")  
 **}  
 }** *// Configuración de compilación nativa* externalNativeBuild **{** cmake **{** cppFlags("-std=c++17", "-fexceptions")  
 arguments(  
 "-DANDROID\_STL=c++\_shared",  
 "-DLLAMA\_NATIVE=OFF",  
 "-DLLAMA\_ARM\_NEON=ON",  
 "-DLLAMA\_K\_QUANTS=ON"  
 )  
 **}  
 }** *// Definir filtro ABI (Application Binary Interface)* ndk **{** abiFilters += *listOf*("arm64-v8a","armeabi-v7a")  
 **}  
 }** buildTypes **{** *release* **{** isMinifyEnabled = false  
 proguardFiles(  
 getDefaultProguardFile("proguard-android-optimize.txt"),  
 "proguard-rules.pro"  
 )  
 **}  
 }** compileOptions **{** sourceCompatibility = JavaVersion.*VERSION\_11* targetCompatibility = JavaVersion.*VERSION\_11* **}** *kotlinOptions* **{** jvmTarget = "11"  
 **}** buildFeatures **{** compose = true  
 **}** *// Enlace al script de compilación CMake* externalNativeBuild **{** cmake **{** path = file("CMakeLists.txt")  
 version = "3.22.1"  
 **}  
 }  
  
}***dependencies* **{** *implementation*(*libs*.*androidx*.*core*.*ktx*)  
 *implementation*(*libs*.*androidx*.*lifecycle*.*runtime*.*ktx*)  
 *implementation*(*libs*.*androidx*.*activity*.*compose*)  
 *implementation*(platform(*libs*.*androidx*.*compose*.*bom*))  
 *implementation*(*libs*.*androidx*.*ui*)  
 *implementation*(*libs*.*androidx*.*ui*.*graphics*)  
 *implementation*(*libs*.*androidx*.*ui*.*tooling*.*preview*)  
 *implementation*(*libs*.*androidx*.*material3*)  
 *testImplementation*(*libs*.*junit*)  
 *androidTestImplementation*(*libs*.*androidx*.*junit*)  
 *androidTestImplementation*(*libs*.*androidx*.*espresso*.*core*)  
 *androidTestImplementation*(platform(*libs*.*androidx*.*compose*.*bom*))  
 *androidTestImplementation*(*libs*.*androidx*.*ui*.*test*.*junit4*)  
 *debugImplementation*(*libs*.*androidx*.*ui*.*tooling*)  
 *debugImplementation*(*libs*.*androidx*.*ui*.*test*.*manifest*)  
  
 *// Añadir dependencias para ViewModel y LiveData  
 implementation*("androidx.lifecycle:lifecycle-viewmodel-compose:2.8.7")  
 *implementation*("androidx.lifecycle:lifecycle-runtime-compose:2.8.7")  
  
 *// Coroutines para trabajo en segundo plano  
 implementation*("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.7.3")  
  
 *// Añadir Google Accompanist para gestión de permisos  
 implementation*("com.google.accompanist:accompanist-permissions:0.34.0")  
  
**Declaración XML y namespaces**

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools">

* Define la versión XML y los namespaces necesarios.
* tools: Namespace para herramientas de desarrollo y directivas de compilación.

<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE"

android:maxSdkVersion="32" />

* Permiso tradicional para leer almacenamiento externo.
* maxSdkVersion="32": Solo se aplica hasta Android 12.

**Android 10 y anterior (API ≤ 29)**

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"

android:maxSdkVersion="29" />

**Android 13+ (API 33+) - Permisos granulares**

<uses-permission android:name="android.permission.READ\_MEDIA\_IMAGES" />

<uses-permission android:name="android.permission.READ\_MEDIA\_VIDEO" />

<uses-permission android:name="android.permission.READ\_MEDIA\_AUDIO" />

<uses-permission android:name="android.permission.MANAGE\_EXTERNAL\_STORAGE"

tools:ignore="ScopedStorage" />

* MANAGE\_EXTERNAL\_STORAGE: Acceso completo al almacenamiento (similar al comportamiento pre-Android 11).
* tools:ignore="ScopedStorage": Suprime advertencias sobre Scoped Storage
* **Importante**: Google Play Store requiere justificación especial para este permiso.

**Configuración de la aplicación**

<application

android:allowBackup="true"

android:dataExtractionRules="@xml/data\_extraction\_rules"

android:fullBackupContent="@xml/backup\_rules"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:roundIcon="@mipmap/ic\_launcher\_round"

android:supportsRtl="true"

android:theme="@style/Theme.MyLlamaApp"

tools:targetApi="31">

**Actividad principal**

<activity

android:name=".MainActivity"

android:exported="true"

android:label="@string/app\_name"

android:theme="@style/Theme.MyLlamaApp">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

Código **AndroidManifest.xml**:

*<?*xml version="1.0" encoding="utf-8"*?>*<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools">  
  
 *<!-- Permisos básicos para almacenamiento -->* <uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE"  
 android:maxSdkVersion="32" />  
  
 *<!-- Para Android 10 (API 29) y anterior -->* <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"  
 android:maxSdkVersion="29" />  
  
 *<!-- Para Android 13 (API 33) y superior -->* <uses-permission android:name="android.permission.READ\_MEDIA\_IMAGES" />  
 <uses-permission android:name="android.permission.READ\_MEDIA\_VIDEO" />  
 <uses-permission android:name="android.permission.READ\_MEDIA\_AUDIO" />  
  
 *<!-- Para Android 11 (API 30) - Solo si es absolutamente necesario -->  
 <!-- Nota: Esto requiere justificación para Play Store -->* <uses-permission android:name="android.permission.MANAGE\_EXTERNAL\_STORAGE"  
 tools:ignore="ScopedStorage" />  
  
  
  
 <application  
 android:allowBackup="true"  
 android:dataExtractionRules="@xml/data\_extraction\_rules"  
 android:fullBackupContent="@xml/backup\_rules"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true"  
 android:theme="@style/Theme.MyLlamaApp"  
 tools:targetApi="31">  
 <activity  
 android:name=".MainActivity"  
 android:exported="true"  
 android:label="@string/app\_name"  
 android:theme="@style/Theme.MyLlamaApp">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
 </activity>  
 </application>  
  
</manifest>

Código **CmakeList.txt**:

cmake\_minimum\_required(VERSION 3.22.1)  
project(llama\_wrapper\_app)  
  
*# Mensajes para depuración*message(STATUS "Configurando proyecto para Android ABI: ${ANDROID\_ABI}")  
message(STATUS "Android SDK: ${ANDROID\_SDK}")  
message(STATUS "Android NDK: ${ANDROID\_NDK}")  
message(STATUS "Toolchain file: ${CMAKE\_TOOLCHAIN\_FILE}")  
  
*# Definir la ruta a llama.cpp*set(LLAMACPP\_SOURCE\_DIR ${CMAKE\_CURRENT\_SOURCE\_DIR}/src/main/cpp/llama.cpp)  
  
*# Verificar o clonar llama.cpp*if(NOT EXISTS ${LLAMACPP\_SOURCE\_DIR})  
 message(STATUS "Clonando llama.cpp desde GitHub...")  
 execute\_process(  
 COMMAND git clone --depth 1 https://github.com/ggerganov/llama.cpp.git ${LLAMACPP\_SOURCE\_DIR}  
 RESULT\_VARIABLE GIT\_RESULT  
 )  
 if(NOT GIT\_RESULT EQUAL "0")  
 message(FATAL\_ERROR "Error al clonar llama.cpp: ${GIT\_RESULT}")  
 endif()  
endif()  
  
*# Configurar opciones para llama.cpp antes de incluirlo*set(LLAMA\_NATIVE OFF CACHE BOOL "Disable native optimizations for cross-compilation" FORCE)  
set(LLAMA\_AVX OFF CACHE BOOL "Disable AVX for ARM" FORCE)  
set(LLAMA\_AVX2 OFF CACHE BOOL "Disable AVX2 for ARM" FORCE)  
set(LLAMA\_AVX512 OFF CACHE BOOL "Disable AVX512 for ARM" FORCE)  
set(LLAMA\_FMA OFF CACHE BOOL "Disable FMA for ARM" FORCE)  
set(LLAMA\_ARM\_NEON ON CACHE BOOL "Enable NEON for ARM" FORCE)  
set(LLAMA\_F16C ON CACHE BOOL "Enable F16C if available" FORCE)  
set(LLAMA\_K\_QUANTS ON CACHE BOOL "Enable k-quants support" FORCE)  
set(GGML\_OPENMP OFF CACHE BOOL "Disable OpenMP for Android" FORCE)  
set(BUILD\_SHARED\_LIBS OFF CACHE BOOL "Build static libraries" FORCE)  
  
*# Deshabilitar los componentes innecesarios de llama.cpp*set(LLAMA\_BUILD\_EXAMPLES OFF CACHE BOOL "Disable examples" FORCE)  
set(LLAMA\_BUILD\_SERVER OFF CACHE BOOL "Disable server" FORCE)  
  
*# IMPORTANTE: Incluir llama.cpp como un subproyecto*add\_subdirectory(${LLAMACPP\_SOURCE\_DIR} ${CMAKE\_BINARY\_DIR}/llama.cpp)  
  
*# Verificar qué objetivos/bibliotecas se crearon*get\_directory\_property(LLAMA\_TARGETS DIRECTORY ${LLAMACPP\_SOURCE\_DIR} BUILDSYSTEM\_TARGETS)  
message(STATUS "Objetivos disponibles en llama.cpp: ${LLAMA\_TARGETS}")  
  
*# Crear nuestra biblioteca wrapper*add\_library(  
 llama\_wrapper  
 SHARED  
 src/main/cpp/llama\_wrapper.cpp  
)  
  
*# Añadir directorios de inclusión*target\_include\_directories(  
 llama\_wrapper  
 PRIVATE  
 ${LLAMACPP\_SOURCE\_DIR}  
)  
  
*# Enlaces con llama (esta es la biblioteca principal de llama.cpp)*target\_link\_libraries(  
 llama\_wrapper  
 PRIVATE  
 llama *# Este es el nombre del objetivo creado por llama.cpp* android  
 log  
)  
  
*# Verificación adicional post-build*add\_custom\_command(  
 TARGET llama\_wrapper  
 POST\_BUILD  
 COMMAND ${CMAKE\_COMMAND} -E echo "Biblioteca llama\_wrapper.so construida con éxito"  
)

**Guardas de inclusión y dependencias**

#ifndef LLAMA\_WRAPPER\_H

#define LLAMA\_WRAPPER\_H

#include <jni.h>

#include <string>

#include "llama.h"

#include "src/llama-model.h"

#include "src/llama-context.h"

* Guards de inclusión: Previenen inclusiones múltiples del mismo archivo.
* jni.h: Biblioteca principal para la interoperabilidad Java-C++.
* llama.h y componentes: Bibliotecas principales de LLaMA.cpp para manejo de modelos y contexto.

**Clase LlamaWrapper - Encapsulación del modelo**

class LlamaWrapper {

public:

LlamaWrapper();

~LlamaWrapper();

Patrón RAII: Constructor y destructor garantizan gestión automática de recursos.

**Métodos principales**

bool loadModel(const std::string& model\_path);

* Carga un modelo desde el sistema de archivos.
* Retorna bool para indicar éxito/fallo.
* Recibe la ruta completa al archivo **.gguf**

std::string generateResponse(const std::string& prompt, int max\_tokens);

* Generación síncrona: Método tradicional que retorna la respuesta completa.
* Útil para casos donde no se necesita feedback en tiempo real.

std::string generateResponseStreaming(JNIEnv\* env, jobject callback\_obj,

const std::string& prompt, int max\_tokens);

* Generación asíncrona: Método avanzado con callback.
* *JNIEnv env*\*: Entorno JNI para comunicarse con Java.
* **jobject callback\_obj**: Objeto Java que recibe tokens incrementales.
* Permite mostrar texto conforme se genera (mejor UX).

**Métodos utilitarios**

bool isModelLoaded() const;

void unloadModel();

* isModelLoaded(): Verificación de estado del modelo.
* unloadModel(): Liberación explícita de memoria (importante para modelos grandes).

**Variables privadas**

private:

llama\_model\* model\_;

llama\_context\* context\_;

bool model\_loaded\_;

int context\_size\_;

int num\_threads\_;

* model\_: Puntero al modelo cargado en memoria.
* context\_: Contexto de conversación para mantener coherencia.
* model\_loaded\_: Flag de estado para evitar operaciones inválidas.
* context\_size\_ y num\_threads\_: Parámetros de configuración de rendimiento.

**Interfaz JNI - Puente Java-C++**

extern "C" {

JNIEXPORT jlong JNICALL Java\_com\_example\_myllamaapp\_LlamaInterface\_createLlamaWrapper(JNIEnv\* env, jobject thiz);

**Convención de nombres JNI**: Java\_[paquete]\_[clase]\_[método]

**Gestión de ciclo de vida**

JNIEXPORT jlong JNICALL Java\_com\_example\_myllamaapp\_LlamaInterface\_createLlamaWrapper(...);

JNIEXPORT void JNICALL Java\_com\_example\_myllamaapp\_LlamaInterface\_deleteLlamaWrapper(..., jlong handle);

* createLlamaWrapper: Retorna jlong (puntero a la instancia C++).
* deleteLlamaWrapper: Recibe handle para destruir la instancia.
* Patrón Handle: Java no maneja punteros directamente, usa handles numéricos.

**Operaciones del modelo**

JNIEXPORT jboolean JNICALL Java\_com\_example\_myllamaapp\_LlamaInterface\_loadModel(JNIEnv\* env, jobject thiz, jlong handle, jstring model\_path);

* jstring model\_path: Cadena Java convertida a C++ internamente
* jboolean: Tipo booleano de JNI (compatible con Java)

**Generación de texto - Dual approach**

*// Método síncrono tradicional*

JNIEXPORT jstring JNICALL Java\_com\_example\_myllamaapp\_LlamaInterface\_generateResponse(...);

*// Método asíncrono con streaming*

JNIEXPORT jstring JNICALL Java\_com\_example\_myllamaapp\_LlamaInterface\_generateResponseStreaming(..., jobject callback, ...);

Código **llama\_wrapper.h**:

#ifndef **LLAMA\_WRAPPER\_H**#define **LLAMA\_WRAPPER\_H**#include <jni.h>  
#include <string>  
#include "llama.h"  
#include "src/llama-model.h"  
#include "src/llama-context.h"  
  
*// Forward declarations*class LlamaWrapper {  
public:  
 LlamaWrapper();  
 ~LlamaWrapper();  
  
 *// Inicializar el modelo desde la ruta del archivo* bool loadModel(const std::string& model\_path);  
  
 *// Método original para generar texto con un prompt dado* std::string generateResponse(const std::string& prompt, int max\_tokens);  
  
 *// Nuevo método con callback para generación incremental* std::string generateResponseStreaming(JNIEnv\* env, jobject callback\_obj,  
 const std::string& prompt, int max\_tokens);  
  
 *// Verificar si el modelo está cargado* bool isModelLoaded() const;  
  
 *// Descargar el modelo y liberar recursos* void unloadModel();  
  
private:  
 llama\_model\* model\_;  
 llama\_context\* context\_;  
 bool model\_loaded\_;  
 int context\_size\_;  
 int num\_threads\_;  
};  
  
*// Declaraciones de funciones JNI*extern "C" {  
**JNIEXPORT** jlong **JNICALL** Java\_com\_example\_myllamaapp\_LlamaInterface\_createLlamaWrapper(JNIEnv\* env, jobject thiz);  
**JNIEXPORT** void **JNICALL** Java\_com\_example\_myllamaapp\_LlamaInterface\_deleteLlamaWrapper(JNIEnv\* env, jobject thiz, jlong handle);  
**JNIEXPORT** jboolean **JNICALL** Java\_com\_example\_myllamaapp\_LlamaInterface\_loadModel(JNIEnv\* env, jobject thiz, jlong handle, jstring model\_path);  
  
*// Método original***JNIEXPORT** jstring **JNICALL** Java\_com\_example\_myllamaapp\_LlamaInterface\_generateResponse(JNIEnv\* env, jobject thiz, jlong handle, jstring prompt, jint max\_tokens);  
  
*// Nuevo método para streaming***JNIEXPORT** jstring **JNICALL** Java\_com\_example\_myllamaapp\_LlamaInterface\_generateResponseStreaming(JNIEnv\* env, jobject thiz, jlong handle, jobject callback, jstring prompt, jint max\_tokens);  
  
**JNIEXPORT** jboolean **JNICALL** Java\_com\_example\_myllamaapp\_LlamaInterface\_isModelLoaded(JNIEnv\* env, jobject thiz, jlong handle);  
**JNIEXPORT** void **JNICALL** Java\_com\_example\_myllamaapp\_LlamaInterface\_unloadModel(JNIEnv\* env, jobject thiz, jlong handle);  
}  
  
#endif *// LLAMA\_WRAPPER\_H*

**Sistema de logging para Android**

#define LOG\_TAG "LlamaWrapper"

#define LOGD(...) \_\_android\_log\_print(ANDROID\_LOG\_DEBUG, LOG\_TAG, \_\_VA\_ARGS\_\_)

#define LOGE(...) \_\_android\_log\_print(ANDROID\_LOG\_ERROR, LOG\_TAG, \_\_VA\_ARGS\_\_)

* Macros de logging: Permiten depuración en tiempo real usando adb logcat.
* LOG\_TAG: Identifica los mensajes de esta clase en el logcat.
* Niveles: DEBUG, ERROR, INFO, WARN para diferentes tipos de información.

**Constructor y destructor - Gestión RAII**

LlamaWrapper::LlamaWrapper() :

model\_(nullptr),

context\_(nullptr),

model\_loaded\_(false),

context\_size\_(2048), *// Tamaño de contexto predeterminado*

num\_threads\_(8) { *// Hilos para procesamiento paralelo*

LOGD("Creando LlamaWrapper");

}

* Inicialización segura: Todos los punteros en nullptr.
* context\_size\_(2048): Ventana de contexto para mantener coherencia en conversaciones.
* num\_threads\_(8): Aprovecha múltiples núcleos del procesador móvil.

**Carga del modelo - Proceso crítico**

bool LlamaWrapper::loadModel(const std::string& model\_path) {

*// Si ya hay un modelo cargado, descargarlo primero*

if (model\_loaded\_) {

LOGW("Ya hay un modelo cargado. Descargando primero...");

unloadModel();

}

**Configuración de parámetros**

llama\_model\_params model\_params = llama\_model\_default\_params();

model\_ = llama\_model\_load\_from\_file(model\_path.c\_str(), model\_params);

llama\_context\_params ctx\_params = llama\_context\_default\_params();

ctx\_params.n\_ctx = context\_size\_;

ctx\_params.n\_threads = num\_threads\_;

context\_ = llama\_init\_from\_model(model\_, ctx\_params);

**Generación de texto - Implementación compleja**

**Preparación y tokenización**

*// Limpiar el contexto KV*

llama\_kv\_self\_clear(context\_);

*// Tokenización segura*

std::vector<llama\_token> tokens;

tokens.resize(std::min(context\_size\_, 1024)); *// Limitar para mayor seguridad*

int n\_tokens = llama\_tokenize(

vocab,

prompt.c\_str(),

prompt.length(),

tokens.data(),

tokens.size(),

true, *// add\_special*

false *// parse\_special*

);

**Configuración de batch processing**

struct llama\_batch batch;

memset(&batch, 0, sizeof(batch));

batch.n\_tokens = n\_tokens;

batch.token = tokens.data();

*// Configurar estructuras auxiliares*

std::vector<llama\_pos> positions(n\_tokens);

std::vector<int32\_t> n\_seq\_id(n\_tokens, 1);

std::vector<llama\_seq\_id\*> seq\_ids(n\_tokens);

**Generación iterativa de tokens**

while (num\_tokens\_generated < max\_tokens) {

*// Obtener logits (probabilidades) para el siguiente token*

float\* logits = llama\_get\_logits(context\_);

*// Selección greedy del token más probable*

float max\_logit = -INFINITY;

int max\_token\_id = -1;

for (int token\_id = 0; token\_id < n\_vocab; token\_id++) {

if (logits[token\_id] > max\_logit) {

max\_logit = logits[token\_id];

max\_token\_id = token\_id;

}

}

**Generación con streaming - Funcionalidad avanzada**

std::string LlamaWrapper::generateResponseStreaming(JNIEnv\* env, jobject callback\_obj,

const std::string& prompt, int max\_tokens) {

*// Obtener el método onToken del callback*

jclass callback\_class = env->GetObjectClass(callback\_obj);

jmethodID onToken\_method = env->GetMethodID(callback\_class, "onToken", "(Ljava/lang/String;)V");

**Callback en tiempo real**

*// Convertir token a texto y enviarlo inmediatamente*

char token\_text[128];

int len = llama\_token\_to\_piece(vocab, new\_token, token\_text, sizeof(token\_text) - 1, 0, false);

if (len > 0 && len < (int)sizeof(token\_text)) {

token\_text[len] = '\0';

response += token\_text;

*// Llamar al callback con el token actual*

jstring jtoken = env->NewStringUTF(token\_text);

env->CallVoidMethod(callback\_obj, onToken\_method, jtoken);

env->DeleteLocalRef(jtoken);

}

**Interfaz JNI - Funciones de exportación**

extern "C" {

JNIEXPORT jlong JNICALL Java\_com\_example\_myllamaapp\_LlamaInterface\_createLlamaWrapper(JNIEnv\* env, jobject thiz) {

LlamaWrapper\* wrapper = new LlamaWrapper();

return reinterpret\_cast<jlong>(wrapper);

}

**Gestión de handles**

JNIEXPORT void JNICALL Java\_com\_example\_myllamaapp\_LlamaInterface\_deleteLlamaWrapper(JNIEnv\* env, jobject thiz, jlong handle) {

if (handle != 0) {

LlamaWrapper\* wrapper = reinterpret\_cast<LlamaWrapper\*>(handle);

delete wrapper;

}

}

Código **llama\_wrapper.cpp**:

#include "llama\_wrapper.h"  
#include <android/log.h>  
#include <vector>  
#include <string>  
#include <stdexcept>  
  
*// Definiciones para logging en Android*#define **LOG\_TAG** "LlamaWrapper"  
#define **LOGD**(...) \_\_android\_log\_print(ANDROID\_LOG\_DEBUG, **LOG\_TAG**, \_\_VA\_ARGS\_\_)  
#define **LOGE**(...) \_\_android\_log\_print(ANDROID\_LOG\_ERROR, **LOG\_TAG**, \_\_VA\_ARGS\_\_)  
#define **LOGI**(...) \_\_android\_log\_print(ANDROID\_LOG\_INFO, **LOG\_TAG**, \_\_VA\_ARGS\_\_)  
#define **LOGW**(...) \_\_android\_log\_print(ANDROID\_LOG\_WARN, **LOG\_TAG**, \_\_VA\_ARGS\_\_)  
  
*// Constructor*LlamaWrapper::LlamaWrapper() :  
 model\_(nullptr),  
 context\_(nullptr),  
 model\_loaded\_(false),  
 context\_size\_(2048), *// Tamaño de contexto predeterminado 128* num\_threads\_(8) { *// Número de hilos predeterminado 4 - 8* **LOGD**("Creando LlamaWrapper");  
}  
  
*// Destructor*LlamaWrapper::~LlamaWrapper() {  
 unloadModel();  
}  
  
*// Cargar el modelo desde la ruta de archivo*bool LlamaWrapper::loadModel(const std::string& model\_path) {  
 **LOGD**("Cargando modelo desde: %s", model\_path.c\_str());  
  
 *// Si ya hay un modelo cargado, descargarlo primero* if (model\_loaded\_) {  
 **LOGW**("Ya hay un modelo cargado. Descargando primero...");  
 unloadModel();  
 }  
  
 try {  
 *// Configurar los parámetros del modelo* llama\_model\_params model\_params = llama\_model\_default\_params();  
  
 *// Cargar el modelo* model\_ = llama\_model\_load\_from\_file(model\_path.c\_str(), model\_params);  
 if (!model\_) {  
 **LOGE**("Error al cargar el modelo desde %s", model\_path.c\_str());  
 return false;  
 }  
  
 *// Configurar los parámetros del contexto* llama\_context\_params ctx\_params = llama\_context\_default\_params();  
 ctx\_params.n\_ctx = context\_size\_;  
 ctx\_params.n\_threads = num\_threads\_;  
  
 *// Crear el contexto usando la API más nueva* context\_ = llama\_init\_from\_model(model\_, ctx\_params);  
 if (!context\_) {  
 **LOGE**("Error al crear el contexto");  
 llama\_model\_free(model\_);  
 model\_ = nullptr;  
 return false;  
 }  
  
 model\_loaded\_ = true;  
 **LOGD**("Modelo cargado exitosamente");  
 return true;  
 } catch (const std::exception& e) {  
 **LOGE**("Excepción durante la carga del modelo: %s", e.what());  
 if (context\_) {  
 llama\_free(context\_);  
 context\_ = nullptr;  
 }  
 if (model\_) {  
 llama\_model\_free(model\_);  
 model\_ = nullptr;  
 }  
 model\_loaded\_ = false;  
 return false;  
 }  
}  
  
*// Método original de generación (sin streaming)*std::string LlamaWrapper::generateResponse(const std::string& prompt, int max\_tokens) {  
 **LOGD**("Generando respuesta para prompt: %s", prompt.c\_str());  
 if (!model\_loaded\_ || !context\_ || !model\_) {  
 **LOGE**("Modelo no cargado o contexto no disponible");  
 return "Error: Modelo no cargado correctamente";  
 }  
  
 try {  
 *// Limpiar el contexto KV* llama\_kv\_self\_clear(context\_);  
  
 *// Verificación segura del vocabulario* const struct llama\_vocab\* vocab = llama\_model\_get\_vocab(model\_);  
 if (vocab == nullptr) {  
 **LOGE**("No se pudo obtener el vocabulario del modelo");  
 return "Error: No se pudo acceder al vocabulario del modelo";  
 }  
  
 *// Tokenización segura* std::vector<llama\_token> tokens;  
 tokens.resize(std::min(context\_size\_, 1024)); *// Limitar para mayor seguridad* **LOGD**("Tokenizando texto, longitud: %zu", prompt.length());  
 int n\_tokens = llama\_tokenize(  
 vocab,  
 prompt.c\_str(),  
 prompt.length(),  
 tokens.data(),  
 tokens.size(),  
 true, *// add\_special* false *// parse\_special* );  
  
 if (n\_tokens <= 0) {  
 **LOGE**("Error al tokenizar texto o prompt vacío");  
 return "Error: No se pudo tokenizar el texto de entrada";  
 }  
  
 *// Ajustar el tamaño del vector al número real de tokens* tokens.resize(n\_tokens);  
 **LOGD**("Texto tokenizado en %d tokens", n\_tokens);  
  
 *// Enfoque simplificado usando la API de nivel más bajo  
 // Vamos a crear y configurar manualmente el batch para mayor control* struct llama\_batch batch;  
 memset(&batch, 0, sizeof(batch));  
  
 batch.n\_tokens = n\_tokens;  
 batch.token = tokens.data();  
  
 *// Alojar memoria para los otros campos del batch* std::vector<llama\_pos> positions(n\_tokens);  
 std::vector<int32\_t> n\_seq\_id(n\_tokens, 1); *// Cada token pertenece a 1 secuencia* std::vector<llama\_seq\_id\*> seq\_ids(n\_tokens);  
 std::vector<llama\_seq\_id> seq\_id\_data(n\_tokens);  
 std::vector<int8\_t> logits(n\_tokens, 0);  
  
 *// Configurar posiciones* for (int i = 0; i < n\_tokens; i++) {  
 positions[i] = i;  
 seq\_id\_data[i] = 0; *// Todos los tokens pertenecen a la secuencia 0* seq\_ids[i] = &seq\_id\_data[i];  
 }  
  
 *// Solicitar logits solo para el último token* logits[n\_tokens - 1] = 1;  
  
 *// Asignar los vectores a los campos del batch* batch.pos = positions.data();  
 batch.n\_seq\_id = n\_seq\_id.data();  
 batch.seq\_id = seq\_ids.data();  
 batch.logits = logits.data();  
  
 *// Procesar el batch con mejor manejo de errores* **LOGD**("Procesando batch de entrada con %d tokens", n\_tokens);  
 int decode\_result = llama\_decode(context\_, batch);  
 if (decode\_result != 0) {  
 **LOGE**("Error al procesar el batch de entrada: código %d", decode\_result);  
 return "Error: Fallo al procesar el prompt (código: " + std::to\_string(decode\_result) + ")";  
 }  
  
 *// Generar tokens de respuesta uno por uno* std::string response;  
 llama\_token new\_token = 0;  
 int num\_tokens\_generated = 0;  
  
 *// Para la generación, creamos un batch sencillo para un solo token* struct llama\_batch gen\_batch;  
 memset(&gen\_batch, 0, sizeof(gen\_batch));  
  
 gen\_batch.n\_tokens = 1;  
 std::vector<llama\_token> gen\_tokens(1);  
 std::vector<llama\_pos> gen\_pos(1);  
 std::vector<int32\_t> gen\_n\_seq\_id(1, 1);  
 std::vector<llama\_seq\_id\*> gen\_seq\_ids(1);  
 std::vector<llama\_seq\_id> gen\_seq\_id\_data(1, 0);  
 std::vector<int8\_t> gen\_logits(1, 1);  
  
 gen\_batch.token = gen\_tokens.data();  
 gen\_batch.pos = gen\_pos.data();  
 gen\_batch.n\_seq\_id = gen\_n\_seq\_id.data();  
 gen\_batch.seq\_id = gen\_seq\_ids.data();  
 gen\_seq\_ids[0] = &gen\_seq\_id\_data[0];  
 gen\_batch.logits = gen\_logits.data();  
  
 while (num\_tokens\_generated < max\_tokens) {  
 *// Obtener logits para predecir el siguiente token* float\* logits = llama\_get\_logits(context\_);  
 if (logits == nullptr) {  
 **LOGE**("Error: No se pudieron obtener los logits");  
 break;  
 }  
  
 const int n\_vocab = llama\_vocab\_n\_tokens(vocab);  
  
 *// Seleccionar el token con mayor probabilidad (greedy decoding)* float max\_logit = -**INFINITY**;  
 int max\_token\_id = -1;  
  
 for (int token\_id = 0; token\_id < n\_vocab; token\_id++) {  
 if (logits[token\_id] > max\_logit) {  
 max\_logit = logits[token\_id];  
 max\_token\_id = token\_id;  
 }  
 }  
  
 if (max\_token\_id < 0 || max\_token\_id >= n\_vocab) {  
 **LOGE**("Token inválido seleccionado: %d", max\_token\_id);  
 break;  
 }  
  
 new\_token = max\_token\_id;  
  
 *// Verificar fin de secuencia (EOS)* if (new\_token == llama\_vocab\_eos(vocab)) {  
 **LOGD**("Encontrado token EOS, terminando generación");  
 break;  
 }  
  
 *// Convertir token a texto* char token\_text[128]; *// Buffer más grande para mayor seguridad* int len = llama\_token\_to\_piece(vocab, new\_token, token\_text, sizeof(token\_text) - 1, 0, false);  
 if (len > 0 && len < (int)sizeof(token\_text)) {  
 token\_text[len] = '\0'; *// Asegurar terminación null* response += token\_text;  
 }  
  
 *// Actualizar el batch para la generación* gen\_tokens[0] = new\_token;  
 gen\_pos[0] = n\_tokens + num\_tokens\_generated;  
  
 *// Procesar el token generado* **LOGD**("Procesando token generado #%d: %d", num\_tokens\_generated + 1, new\_token);  
 if (llama\_decode(context\_, gen\_batch) != 0) {  
 **LOGE**("Error al procesar token generado #%d", num\_tokens\_generated + 1);  
 break;  
 }  
  
 num\_tokens\_generated++;  
 }  
  
 **LOGD**("Generación completada, tokens generados: %d", num\_tokens\_generated);  
 if (response.empty()) {  
 return "No se pudo generar una respuesta.";  
 }  
 return response;  
 }  
 catch (const std::exception& e) {  
 **LOGE**("Excepción durante la generación: %s", e.what());  
 return "Error durante la generación: " + std::string(e.what());  
 }  
 catch (...) {  
 **LOGE**("Excepción desconocida durante la generación");  
 return "Error desconocido durante la generación.";  
 }  
}  
  
*// Nuevo método para generación incremental con callback*std::string LlamaWrapper::generateResponseStreaming(JNIEnv\* env, jobject callback\_obj,  
 const std::string& prompt, int max\_tokens) {  
 **LOGD**("Generando respuesta streaming para prompt: %s", prompt.c\_str());  
 if (!model\_loaded\_ || !context\_ || !model\_) {  
 **LOGE**("Modelo no cargado o contexto no disponible");  
 return "Error: Modelo no cargado correctamente";  
 }  
  
 try {  
 *// Limpiar el contexto KV* llama\_kv\_self\_clear(context\_);  
  
 *// Obtener el método onToken del callback* jclass callback\_class = env->GetObjectClass(callback\_obj);  
 jmethodID onToken\_method = env->GetMethodID(callback\_class, "onToken", "(Ljava/lang/String;)V");  
  
 if (onToken\_method == nullptr) {  
 **LOGE**("No se pudo encontrar el método onToken en el callback");  
 return "Error: Callback no válido";  
 }  
  
 *// Verificación segura del vocabulario* const struct llama\_vocab\* vocab = llama\_model\_get\_vocab(model\_);  
 if (vocab == nullptr) {  
 **LOGE**("No se pudo obtener el vocabulario del modelo");  
 return "Error: No se pudo acceder al vocabulario del modelo";  
 }  
  
 *// Tokenización segura* std::vector<llama\_token> tokens;  
 tokens.resize(std::min(context\_size\_, 1024)); *// Limitar para mayor seguridad* **LOGD**("Tokenizando texto, longitud: %zu", prompt.length());  
 int n\_tokens = llama\_tokenize(  
 vocab,  
 prompt.c\_str(),  
 prompt.length(),  
 tokens.data(),  
 tokens.size(),  
 true, *// add\_special* false *// parse\_special* );  
  
 if (n\_tokens <= 0) {  
 **LOGE**("Error al tokenizar texto o prompt vacío");  
 return "Error: No se pudo tokenizar el texto de entrada";  
 }  
  
 *// Ajustar el tamaño del vector al número real de tokens* tokens.resize(n\_tokens);  
 **LOGD**("Texto tokenizado en %d tokens", n\_tokens);  
  
 *// Configurar batch para procesar los tokens del prompt* struct llama\_batch batch;  
 memset(&batch, 0, sizeof(batch));  
  
 batch.n\_tokens = n\_tokens;  
 batch.token = tokens.data();  
  
 *// Alojar memoria para los otros campos del batch* std::vector<llama\_pos> positions(n\_tokens);  
 std::vector<int32\_t> n\_seq\_id(n\_tokens, 1); *// Cada token pertenece a 1 secuencia* std::vector<llama\_seq\_id\*> seq\_ids(n\_tokens);  
 std::vector<llama\_seq\_id> seq\_id\_data(n\_tokens);  
 std::vector<int8\_t> logits(n\_tokens, 0);  
  
 *// Configurar posiciones* for (int i = 0; i < n\_tokens; i++) {  
 positions[i] = i;  
 seq\_id\_data[i] = 0; *// Todos los tokens pertenecen a la secuencia 0* seq\_ids[i] = &seq\_id\_data[i];  
 }  
  
 *// Solicitar logits solo para el último token* logits[n\_tokens - 1] = 1;  
  
 *// Asignar los vectores a los campos del batch* batch.pos = positions.data();  
 batch.n\_seq\_id = n\_seq\_id.data();  
 batch.seq\_id = seq\_ids.data();  
 batch.logits = logits.data();  
  
 *// Procesar el batch con mejor manejo de errores* **LOGD**("Procesando batch de entrada con %d tokens", n\_tokens);  
 int decode\_result = llama\_decode(context\_, batch);  
 if (decode\_result != 0) {  
 **LOGE**("Error al procesar el batch de entrada: código %d", decode\_result);  
 return "Error: Fallo al procesar el prompt (código: " + std::to\_string(decode\_result) + ")";  
 }  
  
 *// Generar tokens de respuesta uno por uno* std::string response;  
 llama\_token new\_token = 0;  
 int num\_tokens\_generated = 0;  
  
 *// Para la generación, creamos un batch sencillo para un solo token* struct llama\_batch gen\_batch;  
 memset(&gen\_batch, 0, sizeof(gen\_batch));  
  
 gen\_batch.n\_tokens = 1;  
 std::vector<llama\_token> gen\_tokens(1);  
 std::vector<llama\_pos> gen\_pos(1);  
 std::vector<int32\_t> gen\_n\_seq\_id(1, 1);  
 std::vector<llama\_seq\_id\*> gen\_seq\_ids(1);  
 std::vector<llama\_seq\_id> gen\_seq\_id\_data(1, 0);  
 std::vector<int8\_t> gen\_logits(1, 1);  
  
 gen\_batch.token = gen\_tokens.data();  
 gen\_batch.pos = gen\_pos.data();  
 gen\_batch.n\_seq\_id = gen\_n\_seq\_id.data();  
 gen\_batch.seq\_id = gen\_seq\_ids.data();  
 gen\_seq\_ids[0] = &gen\_seq\_id\_data[0];  
 gen\_batch.logits = gen\_logits.data();  
  
 while (num\_tokens\_generated < max\_tokens) {  
 *// Obtener logits para predecir el siguiente token* float\* logits = llama\_get\_logits(context\_);  
 if (logits == nullptr) {  
 **LOGE**("Error: No se pudieron obtener los logits");  
 break;  
 }  
  
 const int n\_vocab = llama\_vocab\_n\_tokens(vocab);  
  
 *// Seleccionar el token con mayor probabilidad (greedy decoding)* float max\_logit = -**INFINITY**;  
 int max\_token\_id = -1;  
  
 for (int token\_id = 0; token\_id < n\_vocab; token\_id++) {  
 if (logits[token\_id] > max\_logit) {  
 max\_logit = logits[token\_id];  
 max\_token\_id = token\_id;  
 }  
 }  
  
 if (max\_token\_id < 0 || max\_token\_id >= n\_vocab) {  
 **LOGE**("Token inválido seleccionado: %d", max\_token\_id);  
 break;  
 }  
  
 new\_token = max\_token\_id;  
  
 *// Verificar fin de secuencia (EOS)* if (new\_token == llama\_vocab\_eos(vocab)) {  
 **LOGD**("Encontrado token EOS, terminando generación");  
 break;  
 }  
  
 *// Convertir token a texto* char token\_text[128]; *// Buffer más grande para mayor seguridad* int len = llama\_token\_to\_piece(vocab, new\_token, token\_text, sizeof(token\_text) - 1, 0, false);  
 if (len > 0 && len < (int)sizeof(token\_text)) {  
 token\_text[len] = '\0'; *// Asegurar terminación null* response += token\_text;  
  
 *// Llamar al callback con el token actual* jstring jtoken = env->NewStringUTF(token\_text);  
 env->CallVoidMethod(callback\_obj, onToken\_method, jtoken);  
 env->DeleteLocalRef(jtoken);  
  
 *// Verificar si hay excepciones después de la llamada al callback* if (env->ExceptionCheck()) {  
 env->ExceptionDescribe();  
 env->ExceptionClear();  
 **LOGE**("Excepción durante la llamada al callback");  
 }  
 }  
  
 *// Actualizar el batch para la generación* gen\_tokens[0] = new\_token;  
 gen\_pos[0] = n\_tokens + num\_tokens\_generated;  
  
 *// Procesar el token generado* **LOGD**("Procesando token generado #%d: %d", num\_tokens\_generated + 1, new\_token);  
 if (llama\_decode(context\_, gen\_batch) != 0) {  
 **LOGE**("Error al procesar token generado #%d", num\_tokens\_generated + 1);  
 break;  
 }  
  
 num\_tokens\_generated++;  
 }  
  
 **LOGD**("Generación streaming completada, tokens generados: %d", num\_tokens\_generated);  
 if (response.empty()) {  
 return "No se pudo generar una respuesta.";  
 }  
 return response;  
 }  
 catch (const std::exception& e) {  
 **LOGE**("Excepción durante la generación streaming: %s", e.what());  
 return "Error durante la generación: " + std::string(e.what());  
 }  
 catch (...) {  
 **LOGE**("Excepción desconocida durante la generación streaming");  
 return "Error desconocido durante la generación.";  
 }  
}  
  
*// Verificar si el modelo está cargado*bool LlamaWrapper::isModelLoaded() const {  
 return model\_loaded\_ && model\_ != nullptr && context\_ != nullptr;  
}  
  
*// Descargar el modelo y liberar recursos*void LlamaWrapper::unloadModel() {  
 if (context\_) {  
 llama\_free(context\_);  
 context\_ = nullptr;  
 }  
 if (model\_) {  
 llama\_model\_free(model\_);  
 model\_ = nullptr;  
 }  
 model\_loaded\_ = false;  
 **LOGD**("Modelo descargado");  
}  
  
*// Implementaciones de las funciones JNI*extern "C" {  
**JNIEXPORT** jlong **JNICALL** Java\_com\_example\_myllamaapp\_LlamaInterface\_createLlamaWrapper(JNIEnv\* env, jobject thiz) {  
 LlamaWrapper\* wrapper = new LlamaWrapper();  
 return reinterpret\_cast<jlong>(wrapper);  
}  
  
**JNIEXPORT** void **JNICALL** Java\_com\_example\_myllamaapp\_LlamaInterface\_deleteLlamaWrapper(JNIEnv\* env, jobject thiz, jlong handle) {  
 if (handle != 0) {  
 LlamaWrapper\* wrapper = reinterpret\_cast<LlamaWrapper\*>(handle);  
 delete wrapper;  
 }  
}  
  
**JNIEXPORT** jboolean **JNICALL** Java\_com\_example\_myllamaapp\_LlamaInterface\_loadModel(JNIEnv\* env, jobject thiz, jlong handle, jstring model\_path) {  
 if (handle == 0 || model\_path == nullptr) {  
 return **JNI\_FALSE**;  
 }  
 LlamaWrapper\* wrapper = reinterpret\_cast<LlamaWrapper\*>(handle);  
 const char\* path = env->GetStringUTFChars(model\_path, nullptr);  
 bool result = wrapper->loadModel(path);  
 env->ReleaseStringUTFChars(model\_path, path);  
 return result ? **JNI\_TRUE** : **JNI\_FALSE**;  
}  
  
*// Método original sin streaming***JNIEXPORT** jstring **JNICALL** Java\_com\_example\_myllamaapp\_LlamaInterface\_generateResponse(JNIEnv\* env, jobject thiz, jlong handle, jstring prompt, jint max\_tokens) {  
 if (handle == 0 || prompt == nullptr) {  
 return env->NewStringUTF("Error: Parámetros inválidos");  
 }  
 LlamaWrapper\* wrapper = reinterpret\_cast<LlamaWrapper\*>(handle);  
 const char\* prompt\_str = env->GetStringUTFChars(prompt, nullptr);  
 std::string response = wrapper->generateResponse(prompt\_str, max\_tokens);  
 env->ReleaseStringUTFChars(prompt, prompt\_str);  
 return env->NewStringUTF(response.c\_str());  
}  
  
*// Nuevo método con streaming***JNIEXPORT** jstring **JNICALL** Java\_com\_example\_myllamaapp\_LlamaInterface\_generateResponseStreaming(JNIEnv\* env, jobject thiz, jlong handle, jobject callback, jstring prompt, jint max\_tokens) {  
 if (handle == 0 || prompt == nullptr || callback == nullptr) {  
 return env->NewStringUTF("Error: Parámetros inválidos");  
 }  
 LlamaWrapper\* wrapper = reinterpret\_cast<LlamaWrapper\*>(handle);  
 const char\* prompt\_str = env->GetStringUTFChars(prompt, nullptr);  
 std::string response = wrapper->generateResponseStreaming(env, callback, prompt\_str, max\_tokens);  
 env->ReleaseStringUTFChars(prompt, prompt\_str);  
 return env->NewStringUTF(response.c\_str());  
}  
  
**JNIEXPORT** jboolean **JNICALL** Java\_com\_example\_myllamaapp\_LlamaInterface\_isModelLoaded(JNIEnv\* env, jobject thiz, jlong handle) {  
 if (handle == 0) {  
 return **JNI\_FALSE**;  
 }  
 LlamaWrapper\* wrapper = reinterpret\_cast<LlamaWrapper\*>(handle);  
 return wrapper->isModelLoaded() ? **JNI\_TRUE** : **JNI\_FALSE**;  
}  
  
**JNIEXPORT** void **JNICALL** Java\_com\_example\_myllamaapp\_LlamaInterface\_unloadModel(JNIEnv\* env, jobject thiz, jlong handle) {  
 if (handle != 0) {  
 LlamaWrapper\* wrapper = reinterpret\_cast<LlamaWrapper\*>(handle);  
 wrapper->unloadModel();  
 }  
}  
}

**Clase PermissionManager - Núcleo del sistema**

class PermissionManager(private val context: Context) {

private val TAG = "PermissionManager"

**Permisos adaptativos por versión de Android**

fun getRequiredPermissions(): Array<String> {

return when {

Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.TIRAMISU -> {

*// Android 13+*

arrayOf(

Manifest.permission.READ\_MEDIA\_IMAGES,

Manifest.permission.READ\_MEDIA\_VIDEO,

Manifest.permission.READ\_MEDIA\_AUDIO

)

}

Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.R -> {

*// Android 11-12*

arrayOf(Manifest.permission.READ\_EXTERNAL\_STORAGE)

}

else -> {

*// Android 10 y anterior*

arrayOf(

Manifest.permission.READ\_EXTERNAL\_STORAGE,

Manifest.permission.WRITE\_EXTERNAL\_STORAGE

)

}

}

}

**Verificación inteligente de permisos**

fun checkPermissions(): Boolean {

val permissions = getRequiredPermissions()

return if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.R) {

*// En Android 11+, verificamos si tenemos acceso a todos los archivos si es necesario*

Environment.isExternalStorageManager() || permissions.all {

ContextCompat.checkSelfPermission(context, it) == PackageManager.PERMISSION\_GRANTED

}

} else {

*// Para versiones anteriores, verificamos los permisos normales*

permissions.all {

ContextCompat.checkSelfPermission(context, it) == PackageManager.PERMISSION\_GRANTED

}

}

}

**Gestión del permiso especial MANAGE\_EXTERNAL\_STORAGE**

fun requestManageExternalStoragePermission() {

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.R) {

try {

val intent = Intent(Settings.ACTION\_MANAGE\_APP\_ALL\_FILES\_ACCESS\_PERMISSION)

intent.addCategory("android.intent.category.DEFAULT")

intent.data = Uri.parse("package:${context.packageName}")

context.startActivity(intent)

} catch (e: Exception) {

Log.e(TAG, "Error al solicitar permiso MANAGE\_EXTERNAL\_STORAGE", e)

*// Alternativa si lo anterior falla*

val intent = Intent(Settings.ACTION\_MANAGE\_ALL\_FILES\_ACCESS\_PERMISSION)

context.startActivity(intent)

}

}

}

**Diagnóstico completo del sistema**

fun diagnosticPermissions(): String {

val sb = StringBuilder()

sb.appendLine("Android SDK: ${Build.VERSION.SDK\_INT}")

*// Verificar permisos regulares*

for (permission in getRequiredPermissions()) {

val isGranted = ContextCompat.checkSelfPermission(

context, permission

) == PackageManager.PERMISSION\_GRANTED

sb.appendLine("Permiso: $permission - ${if (isGranted) "CONCEDIDO" else "DENEGADO"}")

}

*// Verificar permiso especial en Android 11+*

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.R) {

sb.appendLine("MANAGE\_EXTERNAL\_STORAGE: ${if (Environment.isExternalStorageManager()) "CONCEDIDO" else "DENEGADO"}")

}

return sb.toString()

}

**Composable PermissionsHandler - Integración con Jetpack Compose**

@Composable

fun PermissionsHandler(

onPermissionsGranted: () -> Unit,

onPermissionsDenied: () -> Unit,

shouldRequestPermissions: Boolean = true

) {

val context = androidx.compose.ui.platform.LocalContext.current

val permissionManager = remember { PermissionManager(context) }

var permissionsGranted by remember { mutableStateOf(permissionManager.checkPermissions()) }

**Launcher para solicitud de permisos múltiples**

val permissionLauncher = rememberLauncherForActivityResult(

ActivityResultContracts.RequestMultiplePermissions()

) { permissions ->

*// Verificar si todos los permisos fueron concedidos*

val allGranted = permissions.values.all { it }

if (allGranted) {

Log.d("PermissionsHandler", "Todos los permisos básicos concedidos")

*// En Android 11+, es posible que también necesitemos MANAGE\_EXTERNAL\_STORAGE*

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.R &&

!Environment.isExternalStorageManager()) {

*// Solicitamos el permiso especial*

permissionManager.requestManageExternalStoragePermission()

} else {

permissionsGranted = true

onPermissionsGranted()

}

} else {

Log.d("PermissionsHandler", "Algunos permisos fueron denegados")

permissionsGranted = false

onPermissionsDenied()

}

}

**Efecto de lanzamiento para verificación automática**

LaunchedEffect(shouldRequestPermissions) {

if (permissionManager.checkPermissions()) {

permissionsGranted = true

onPermissionsGranted()

} else if (shouldRequestPermissions) {

permissionLauncher.launch(permissionManager.getRequiredPermissions())

} else {

onPermissionsDenied()

}

}

Código **PermissionManager.kt**:

package com.example.myllamaapp  
  
import android.Manifest  
import android.app.Activity  
import android.content.Context  
import android.content.Intent  
import android.content.pm.PackageManager  
import android.net.Uri  
import android.os.Build  
import android.os.Environment  
import android.provider.Settings  
import android.util.Log  
import androidx.activity.compose.rememberLauncherForActivityResult  
import androidx.activity.result.contract.ActivityResultContracts  
import androidx.compose.runtime.Composable  
import androidx.compose.runtime.LaunchedEffect  
import androidx.compose.runtime.getValue  
import androidx.compose.runtime.mutableStateOf  
import androidx.compose.runtime.remember  
import androidx.compose.runtime.setValue  
import androidx.core.app.ActivityCompat  
import androidx.core.content.ContextCompat  
  
class PermissionManager(private val context: Context) {  
 private val TAG = "PermissionManager"  
  
 *// Determinar qué permisos necesitamos según la versión de Android* fun getRequiredPermissions(): Array<String> {  
 return when {  
 Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*TIRAMISU* -> {  
 *// Android 13+  
 arrayOf*(  
 Manifest.permission.*READ\_MEDIA\_IMAGES*,  
 Manifest.permission.*READ\_MEDIA\_VIDEO*,  
 Manifest.permission.*READ\_MEDIA\_AUDIO* )  
 }  
 Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*R* -> {  
 *// Android 11-12  
 arrayOf*(Manifest.permission.*READ\_EXTERNAL\_STORAGE*)  
 }  
 else -> {  
 *// Android 10 y anterior  
 arrayOf*(  
 Manifest.permission.*READ\_EXTERNAL\_STORAGE*,  
 Manifest.permission.*WRITE\_EXTERNAL\_STORAGE* )  
 }  
 }  
 }  
  
 *// Verificar si todos los permisos están concedidos* fun checkPermissions(): Boolean {  
 val permissions = getRequiredPermissions()  
  
 return if (Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*R*) {  
 *// En Android 11+, verificamos si tenemos acceso a todos los archivos si es necesario* Environment.isExternalStorageManager() || permissions.*all* **{** ContextCompat.checkSelfPermission(context, **it**) == PackageManager.*PERMISSION\_GRANTED* **}** } else {  
 *// Para versiones anteriores, verificamos los permisos normales* permissions.*all* **{** ContextCompat.checkSelfPermission(context, **it**) == PackageManager.*PERMISSION\_GRANTED* **}** }  
 }  
  
 *// Verificar si debemos mostrar explicación para algún permiso* fun shouldShowRequestPermissionRationale(activity: Activity): Boolean {  
 return getRequiredPermissions().*any* **{** ActivityCompat.shouldShowRequestPermissionRationale(activity, **it**)  
 **}** }  
  
 *// Solicitar permisos estándar* fun requestPermissions(activity: Activity, requestCode: Int) {  
 ActivityCompat.requestPermissions(  
 activity,  
 getRequiredPermissions(),  
 requestCode  
 )  
 }  
  
 *// Solicitar acceso completo al almacenamiento (solo Android 11+)* fun requestManageExternalStoragePermission() {  
 if (Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*R*) {  
 try {  
 val intent = Intent(Settings.*ACTION\_MANAGE\_APP\_ALL\_FILES\_ACCESS\_PERMISSION*)  
 intent.addCategory("android.intent.category.DEFAULT")  
 intent.*data* = Uri.parse("package:${context.*packageName*}")  
 context.startActivity(intent)  
 } catch (e: Exception) {  
 Log.e(TAG, "Error al solicitar permiso MANAGE\_EXTERNAL\_STORAGE", e)  
 *// Alternativa si lo anterior falla* val intent = Intent(Settings.*ACTION\_MANAGE\_ALL\_FILES\_ACCESS\_PERMISSION*)  
 context.startActivity(intent)  
 }  
 }  
 }  
  
 *// Abrir configuración de la aplicación* fun openAppSettings() {  
 val intent = Intent(Settings.*ACTION\_APPLICATION\_DETAILS\_SETTINGS*)  
 intent.*data* = Uri.fromParts("package", context.*packageName*, null)  
 context.startActivity(intent)  
 }  
  
 *// Verificar permisos y realizar diagnóstico detallado* fun diagnosticPermissions(): String {  
 val sb = StringBuilder()  
  
 sb.*appendLine*("Android SDK: ${Build.VERSION.*SDK\_INT*}")  
  
 *// Verificar permisos regulares* for (permission in getRequiredPermissions()) {  
 val isGranted = ContextCompat.checkSelfPermission(  
 context, permission  
 ) == PackageManager.*PERMISSION\_GRANTED* sb.*appendLine*("Permiso: $permission - ${if (isGranted) "CONCEDIDO" else "DENEGADO"}")  
 }  
  
 *// Verificar permiso especial en Android 11+* if (Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*R*) {  
 sb.*appendLine*("MANAGE\_EXTERNAL\_STORAGE: ${if (Environment.isExternalStorageManager()) "CONCEDIDO" else "DENEGADO"}")  
 }  
  
 return sb.toString()  
 }  
}  
  
*// Composable para gestionar permisos*@Composable  
fun PermissionsHandler(  
 onPermissionsGranted: () -> Unit,  
 onPermissionsDenied: () -> Unit,  
 shouldRequestPermissions: Boolean = true  
) {  
 val context = androidx.compose.ui.platform.*LocalContext*.current  
 val permissionManager = remember **{** PermissionManager(context) **}** var permissionsGranted by remember **{** *mutableStateOf*(permissionManager.checkPermissions()) **}** *// Solicitar permisos si es necesario* val permissionLauncher = rememberLauncherForActivityResult(  
 ActivityResultContracts.RequestMultiplePermissions()  
 ) **{** permissions **->** *// Verificar si todos los permisos fueron concedidos* val allGranted = permissions.values.*all* **{ it }** if (allGranted) {  
 Log.d("PermissionsHandler", "Todos los permisos básicos concedidos")  
  
 *// En Android 11+, es posible que también necesitemos MANAGE\_EXTERNAL\_STORAGE* if (Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*R* &&  
 !Environment.isExternalStorageManager()) {  
  
 *// Solicitamos el permiso especial* permissionManager.requestManageExternalStoragePermission()  
 } else {  
 permissionsGranted = true  
 onPermissionsGranted()  
 }  
 } else {  
 Log.d("PermissionsHandler", "Algunos permisos fueron denegados")  
 permissionsGranted = false  
 onPermissionsDenied()  
 }  
 **}** *// Verificar inicialmente y después de cada cambio* LaunchedEffect(shouldRequestPermissions) **{** if (permissionManager.checkPermissions()) {  
 permissionsGranted = true  
 onPermissionsGranted()  
 } else if (shouldRequestPermissions) {  
 permissionLauncher.launch(permissionManager.getRequiredPermissions())  
 } else {  
 onPermissionsDenied()  
 }  
 **}**}

**2. Contrato JNI**

La interfaz debe coincidir exactamente con la signatura JNI definida en C++:

*// En llama\_wrapper.cpp*

jmethodID onToken\_method = env->GetMethodID(callback\_class, "onToken", "(Ljava/lang/String;)V");

* **Nombre del método**: "onToken"
* **Signatura**: "(Ljava/lang/String;)V" (recibe String, retorna void)

Código **TokenCallback.kt**:

package com.example.myllamaapp  
  
interface TokenCallback {  
 fun onToken(token: String)  
}

**Gestión de la biblioteca nativa**

*// Cargar la biblioteca nativa*

init {

try {

System.loadLibrary("llama\_wrapper")

Log.d(TAG, "Biblioteca llama\_wrapper cargada exitosamente")

} catch (e: UnsatisfiedLinkError) {

Log.e(TAG, "Error al cargar la biblioteca llama\_wrapper: ${e.message}")

}

}

**Carga dinámica de biblioteca:**

* System.loadLibrary("llama\_wrapper"): Busca libllama\_wrapper.so en los directorios nativos de la APK.
* Manejo de errores robusto: Captura UnsatisfiedLinkError si la biblioteca no se encuentra.
* Inicialización temprana: Se ejecuta en el bloque init, garantizando carga inmediata.

**Gestión del handle nativo**

private var nativeHandle: Long = 0

init {

*// ... carga de biblioteca ...*

nativeHandle = createLlamaWrapper()

if (nativeHandle == 0L) {

Log.e(TAG, "Error al crear instancia nativa de LlamaWrapper")

}

}

**Finalización automática de recursos**

protected fun finalize() {

try {

if (nativeHandle != 0L) {

deleteLlamaWrapper(nativeHandle)

nativeHandle = 0

}

} catch (e: Throwable) {

Log.e(TAG, "Error en finalize: ${e.message}")

}

}

**API de carga de modelos**

fun loadModel(modelPath: String): Boolean {

if (nativeHandle == 0L) {

Log.e(TAG, "Instancia nativa no inicializada")

return false

}

return loadModel(nativeHandle, modelPath)

}

**Generación síncrona (método tradicional)**

fun generateResponse(prompt: String, maxTokens: Int): String {

if (nativeHandle == 0L) {

Log.e(TAG, "Instancia nativa no inicializada")

return "Error: Instancia nativa no inicializada"

}

return generateResponse(nativeHandle, prompt, maxTokens)

}

**Generación asíncrona con streaming**

fun generateResponseStreaming(prompt: String, maxTokens: Int, callback: TokenCallback): String {

if (nativeHandle == 0L) {

Log.e(TAG, "Instancia nativa no inicializada")

return "Error: Instancia nativa no inicializada"

}

return generateResponseStreaming(nativeHandle, callback, prompt, maxTokens)

}

**Métodos utilitarios**

fun isModelLoaded(): Boolean {

if (nativeHandle == 0L) {

return false

}

return isModelLoaded(nativeHandle)

}

fun unloadModel() {

if (nativeHandle != 0L) {

unloadModel(nativeHandle)

}

}

**Declaraciones de métodos nativos**

private external fun createLlamaWrapper(): Long

private external fun deleteLlamaWrapper(handle: Long)

private external fun loadModel(handle: Long, modelPath: String): Boolean

private external fun generateResponse(handle: Long, prompt: String, maxTokens: Int): String

private external fun generateResponseStreaming(handle: Long, callback: TokenCallback, prompt: String, maxTokens: Int): String

private external fun isModelLoaded(handle: Long): Boolean

private external fun unloadModel(handle: Long)

**1. Validación de estado consistente**

if (nativeHandle == 0L) {

Log.e(TAG, "Instancia nativa no inicializada")

return false *// o mensaje de error apropiado*

}

**2. Manejo de errores graceful**

try {

System.loadLibrary("llama\_wrapper")

*// ...*

} catch (e: UnsatisfiedLinkError) {

Log.e(TAG, "Error al cargar la biblioteca llama\_wrapper: ${e.message}")

}

**3. Limpieza automática de recursos**

*// 1. Crear instancia (automático en init)*

val llamaInterface = LlamaInterface()

*// 2. Cargar modelo*

val success = llamaInterface.loadModel("/path/to/model.gguf")

*// 3a. Generación síncrona*

val response = llamaInterface.generateResponse("Hola", 50)

*// 3b. O generación con streaming*

val streamingResponse = llamaInterface.generateResponseStreaming(

"Hola",

50,

object : TokenCallback {

override fun onToken(token: String) {

*// Actualizar UI en tiempo real*

}

}

)

*// 4. Verificar estado (opcional)*

if (llamaInterface.isModelLoaded()) {

*// Modelo listo para usar*

}

*// 5. Limpiar recursos (opcional, automático en GC)*

llamaInterface.unloadModel()

Código **LlamaInterface.kt**:

package com.example.myllamaapp  
import android.util.Log  
  
class LlamaInterface {  
 private val TAG = "LlamaInterface"  
 *// Puntero nativo al objeto LlamaWrapper* private var nativeHandle: Long = 0  
  
 *// Cargar la biblioteca nativa* init {  
 try {  
 System.loadLibrary("llama\_wrapper")  
 Log.d(TAG, "Biblioteca llama\_wrapper cargada exitosamente")  
 } catch (e: UnsatisfiedLinkError) {  
 Log.e(TAG, "Error al cargar la biblioteca llama\_wrapper: ${e.message}")  
 }  
 *// Inicializar el wrapper nativo* nativeHandle = createLlamaWrapper()  
 if (nativeHandle == 0L) {  
 Log.e(TAG, "Error al crear instancia nativa de LlamaWrapper")  
 }  
 }  
  
 *// Liberar recursos al finalizar* protected fun finalize() {  
 try {  
 if (nativeHandle != 0L) {  
 deleteLlamaWrapper(nativeHandle)  
 nativeHandle = 0  
 }  
 } catch (e: Throwable) {  
 Log.e(TAG, "Error en finalize: ${e.message}")  
 }  
 }  
  
 *// Cargar modelo desde una ruta* fun loadModel(modelPath: String): Boolean {  
 if (nativeHandle == 0L) {  
 Log.e(TAG, "Instancia nativa no inicializada")  
 return false  
 }  
 return loadModel(nativeHandle, modelPath)  
 }  
  
 *// Método original - mantenerlo para compatibilidad* fun generateResponse(prompt: String, maxTokens: Int): String {  
 if (nativeHandle == 0L) {  
 Log.e(TAG, "Instancia nativa no inicializada")  
 return "Error: Instancia nativa no inicializada"  
 }  
 *// Este método internamente no usará un callback* return generateResponse(nativeHandle, prompt, maxTokens)  
 }  
  
 *// Nuevo método para generación con streaming* fun generateResponseStreaming(prompt: String, maxTokens: Int, callback: TokenCallback): String {  
 if (nativeHandle == 0L) {  
 Log.e(TAG, "Instancia nativa no inicializada")  
 return "Error: Instancia nativa no inicializada"  
 }  
 return generateResponseStreaming(nativeHandle, callback, prompt, maxTokens)  
 }  
  
 *// Verificar si el modelo está cargado* fun isModelLoaded(): Boolean {  
 if (nativeHandle == 0L) {  
 return false  
 }  
 return isModelLoaded(nativeHandle)  
 }  
  
 *// Descargar modelo y liberar recursos* fun unloadModel() {  
 if (nativeHandle != 0L) {  
 unloadModel(nativeHandle)  
 }  
 }  
  
 *// Métodos nativos* private external fun createLlamaWrapper(): Long  
 private external fun deleteLlamaWrapper(handle: Long)  
 private external fun loadModel(handle: Long, modelPath: String): Boolean  
 private external fun generateResponse(handle: Long, prompt: String, maxTokens: Int): String  
 private external fun generateResponseStreaming(handle: Long, callback: TokenCallback, prompt: String, maxTokens: Int): String  
 private external fun isModelLoaded(handle: Long): Boolean  
 private external fun unloadModel(handle: Long)  
}

**Arquitectura reactiva con StateFlow**

private val \_uiState = MutableStateFlow(LlamaUiState())

val uiState: StateFlow<LlamaUiState> = \_uiState.asStateFlow()

**Inicialización de dependencias**

private val llamaInterface = LlamaInterface()

private val MODEL\_FILENAME = "phi-2.Q3\_K\_L.gguf"

private val possibleStoragePaths = listOf(

"${Environment.getExternalStorageDirectory().absolutePath}/LLaMA",

"/sdcard/LLaMA",

"/storage/emulated/0/LLaMA",

"/storage/sdcard0/LLaMA",

"/storage/sdcard1/LLaMA",

"/mnt/sdcard/LLaMA"

)

**Diagnóstico avanzado del sistema de archivos**

fun debugStorageAccess(context: Context) {

viewModelScope.launch {

withContext(Dispatchers.IO) {

try {

\_uiState.update { it.copy(isLoading = true, statusMessage = "Ejecutando diagnóstico...") }

val sb = StringBuilder()

*// 1. Información general de permisos*

val permissionManager = PermissionManager(context)

sb.appendLine("=== INFORMACIÓN DE PERMISOS ===")

sb.appendLine(permissionManager.diagnosticPermissions())

**Parte 1: Análisis de permisos**

val permissionManager = PermissionManager(context)

sb.appendLine(permissionManager.diagnosticPermissions())

Verifica el estado completo de permisos usando la clase PermissionManager vista anteriormente.

**Parte 2: Información del almacenamiento principal**

val externalDir = Environment.getExternalStorageDirectory()

sb.appendLine("Directorio principal: ${externalDir.absolutePath}")

sb.appendLine("¿Existe? ${externalDir.exists()}, ¿Lectura? ${externalDir.canRead()}")

Examina el directorio de almacenamiento externo principal del dispositivo.

**Parte 3: Verificación exhaustiva de rutas**

for (path in possibleStoragePaths) {

val dir = File(path)

sb.appendLine("Ruta: $path")

sb.appendLine("¿Existe? ${dir.exists()}, ¿Lectura? ${dir.canRead()}")

if (dir.exists() && dir.canRead()) {

val files = dir.listFiles()

if (files != null) {

sb.appendLine("Archivos encontrados: ${files.size}")

for (file in files) {

sb.appendLine(" - ${file.name} (${file.length()} bytes)")

if (file.name == MODEL\_FILENAME) {

modelFound = true

sb.appendLine(" \*\*\* MODELO ENCONTRADO \*\*\*")

\_uiState.update { it.copy(modelPath = file.absolutePath) }

}

}

}

}

}

**Parte 4: Verificación específica del modelo**

val specificPaths = possibleStoragePaths.map { "$it/$MODEL\_FILENAME" }

for (path in specificPaths) {

val file = File(path)

sb.appendLine("Archivo: $path")

sb.appendLine("¿Existe? ${file.exists()}, ¿Lectura? ${file.canRead()}, Tamaño: ${file.length()} bytes")

}

Verifica específicamente la existencia del archivo del modelo en cada ubicación.

**Búsqueda optimizada de modelos**

fun findModelInExternalStorage() {

viewModelScope.launch {

withContext(Dispatchers.IO) {

try {

\_uiState.update { it.copy(isLoading = true, statusMessage = "Buscando modelo...") }

for (path in possibleStoragePaths) {

val modelDir = File(path)

val modelFile = File(modelDir, MODEL\_FILENAME)

Log.d(TAG, "Buscando modelo en: ${modelFile.absolutePath}")

if (modelFile.exists() && modelFile.canRead()) {

Log.d(TAG, "Modelo encontrado en: ${modelFile.absolutePath}")

Log.d(TAG, "Tamaño del archivo: ${modelFile.length()} bytes")

\_uiState.update {

it.copy(

isLoading = false,

modelPath = modelFile.absolutePath,

statusMessage = "Modelo encontrado en ${modelFile.absolutePath}"

)

}

return@withContext

}

}

}

}

}

}

**Carga de modelos con manejo robusto de errores**

fun loadModel(modelPath: String) {

viewModelScope.launch {

withContext(Dispatchers.IO) {

try {

\_uiState.update { it.copy(isLoading = true, statusMessage = "Cargando modelo...") }

val success = llamaInterface.loadModel(modelPath)

if (success) {

\_uiState.update {

it.copy(

isLoading = false,

isModelLoaded = true,

statusMessage = "Modelo cargado exitosamente"

)

}

} else {

\_uiState.update {

it.copy(

isLoading = false,

isModelLoaded = false,

statusMessage = "Error al cargar el modelo"

)

}

}

} catch (e: Exception) {

Log.e(TAG, "Error al cargar el modelo", e)

\_uiState.update {

it.copy(

isLoading = false,

isModelLoaded = false,

statusMessage = "Error al cargar el modelo: ${e.message}"

)

}

}

}

}

}

**Generación síncrona (compatibilidad)**

fun generateResponse(prompt: String) {

if (!llamaInterface.isModelLoaded()) {

\_uiState.update { it.copy(statusMessage = "Modelo no cargado") }

return

}

viewModelScope.launch {

withContext(Dispatchers.IO) {

try {

\_uiState.update {

it.copy(

isGenerating = true,

statusMessage = "Generando respuesta..."

)

}

val response = llamaInterface.generateResponse(prompt, 200)

\_uiState.update {

it.copy(

isGenerating = false,

response = response,

statusMessage = "Respuesta generada"

)

}

}

}

}

}

**Generación streaming - Funcionalidad avanzada**

fun generateResponseStreaming(prompt: String) {

if (!llamaInterface.isModelLoaded()) {

\_uiState.update { it.copy(statusMessage = "Modelo no cargado") }

return

}

viewModelScope.launch {

withContext(Dispatchers.IO) {

try {

\_uiState.update {

it.copy(

isGenerating = true,

statusMessage = "Generando respuesta...",

response = "" *// Limpiar la respuesta anterior*

)

}

*// Crear callback que recibirá tokens en tiempo real*

val tokenCallback = object : TokenCallback {

override fun onToken(token: String) {

viewModelScope.launch(Dispatchers.Main) {

\_uiState.update {

it.copy(response = it.response + token)

}

}

}

}

*// Llamar al método con streaming*

val finalResponse = llamaInterface.generateResponseStreaming(prompt, 200, tokenCallback)

\_uiState.update {

it.copy(

isGenerating = false,

statusMessage = "Respuesta generada"

)

}

}

}

}

}

**Gestión del ciclo de vida**

override fun onCleared() {

super.onCleared()

llamaInterface.unloadModel()

}

**Estado de la aplicación - Data class**

data class LlamaUiState(

val isLoading: Boolean = false,

val isGenerating: Boolean = false,

val isModelLoaded: Boolean = false,

val modelPath: String = "",

val prompt: String = "",

val response: String = "",

val statusMessage: String = "Esperando permisos...",

val diagnosticInfo: String = ""

)

**1. Scoped coroutines**

viewModelScope.launch {

withContext(Dispatchers.IO) {

*// Operaciones de archivo en hilo de fondo*

}

}

**2. Thread switching para UI**

viewModelScope.launch(Dispatchers.Main) {

\_uiState.update { */\* actualizar UI \*/* }

}

**3. Estado thread-safe**

\_uiState.update { currentState ->

currentState.copy(newProperty = newValue)

}

Código **LlamaViewModel.kt**:

package com.example.myllamaapp  
  
import android.content.Context  
import android.os.Environment  
import android.util.Log  
import androidx.lifecycle.ViewModel  
import androidx.lifecycle.*viewModelScope*import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.flow.MutableStateFlow  
import kotlinx.coroutines.flow.StateFlow  
import kotlinx.coroutines.flow.asStateFlow  
import kotlinx.coroutines.flow.update  
import kotlinx.coroutines.launch  
import kotlinx.coroutines.withContext  
import java.io.File  
  
class LlamaViewModel : ViewModel() {  
 private val TAG = "LlamaViewModel"  
  
 private val \_uiState = *MutableStateFlow*(LlamaUiState())  
 val uiState: StateFlow<LlamaUiState> = \_uiState.*asStateFlow*()  
  
 private val llamaInterface = LlamaInterface()  
  
 *// Nombre del archivo del modelo* private val MODEL\_FILENAME = "phi-2.Q3\_K\_L.gguf"  
  
 *// Posibles rutas de almacenamiento externo* private val possibleStoragePaths = *listOf*(  
 "${Environment.getExternalStorageDirectory().*absolutePath*}/LLaMA",  
 "/sdcard/LLaMA",  
 "/storage/emulated/0/LLaMA",  
 "/storage/sdcard0/LLaMA",  
 "/storage/sdcard1/LLaMA",  
 "/mnt/sdcard/LLaMA"  
 )  
  
 *// Función avanzada de diagnóstico de almacenamiento* fun debugStorageAccess(context: Context) {  
 *viewModelScope*.*launch* **{** withContext(Dispatchers.IO) **{** try {  
 \_uiState.*update* **{ it**.copy(isLoading = true, statusMessage = "Ejecutando diagnóstico...") **}** val sb = StringBuilder()  
  
 *// 1. Información general de permisos* val permissionManager = PermissionManager(context)  
 sb.*appendLine*("=== INFORMACIÓN DE PERMISOS ===")  
 sb.*appendLine*(permissionManager.diagnosticPermissions())  
 sb.*appendLine*()  
  
 *// 2. Información del almacenamiento externo* sb.*appendLine*("=== INFORMACIÓN DE ALMACENAMIENTO ===")  
 val externalDir = Environment.getExternalStorageDirectory()  
 sb.*appendLine*("Directorio principal: ${externalDir.*absolutePath*}")  
 sb.*appendLine*("¿Existe? ${externalDir.exists()}, ¿Lectura? ${externalDir.canRead()}")  
 sb.*appendLine*()  
  
 *// 3. Verificar todas las posibles rutas* sb.*appendLine*("=== VERIFICACIÓN DE RUTAS ===")  
 var modelFound = false  
  
 for (path in possibleStoragePaths) {  
 val dir = File(path)  
 sb.*appendLine*("Ruta: $path")  
 sb.*appendLine*("¿Existe? ${dir.exists()}, ¿Lectura? ${dir.canRead()}")  
  
 if (dir.exists() && dir.canRead()) {  
 val files = dir.listFiles()  
 if (files != null) {  
 sb.*appendLine*("Archivos encontrados: ${files.size}")  
 for (file in files) {  
 sb.*appendLine*(" - ${file.*name*} (${file.length()} bytes)")  
 if (file.*name* == MODEL\_FILENAME) {  
 modelFound = true  
 sb.*appendLine*(" \*\*\* MODELO ENCONTRADO \*\*\*")  
  
 *// Establecer la ruta del modelo* \_uiState.*update* **{  
 it**.copy(modelPath = file.*absolutePath*)  
 **}** }  
 }  
 } else {  
 sb.*appendLine*("No se pudo listar archivos (null)")  
 }  
 }  
 sb.*appendLine*()  
 }  
  
 *// 4. Verificar archivo específico* sb.*appendLine*("=== VERIFICACIÓN DEL ARCHIVO DEL MODELO ===")  
 val specificPaths = possibleStoragePaths.*map* **{** "$**it**/$MODEL\_FILENAME" **}** for (path in specificPaths) {  
 val file = File(path)  
 sb.*appendLine*("Archivo: $path")  
 sb.*appendLine*("¿Existe? ${file.exists()}, ¿Lectura? ${file.canRead()}, Tamaño: ${file.length()} bytes")  
 }  
  
 *// Actualizar estado de la UI* val resultMessage = if (modelFound) {  
 "Diagnóstico completo: Modelo encontrado"  
 } else {  
 "Diagnóstico completo: Modelo NO encontrado"  
 }  
  
 Log.d(TAG, "Resultado de diagnóstico:\n${sb}")  
  
 \_uiState.*update* **{  
 it**.copy(  
 isLoading = false,  
 statusMessage = resultMessage,  
 diagnosticInfo = sb.toString()  
 )  
 **}** } catch (e: Exception) {  
 Log.e(TAG, "Error en diagnóstico", e)  
 \_uiState.*update* **{  
 it**.copy(  
 isLoading = false,  
 statusMessage = "Error en diagnóstico: ${e.message}"  
 )  
 **}** }  
 **}  
 }** }  
  
 fun findModelInExternalStorage() {  
 *viewModelScope*.*launch* **{** withContext(Dispatchers.IO) **{** try {  
 \_uiState.*update* **{ it**.copy(isLoading = true, statusMessage = "Buscando modelo...") **}** *// Verificar en cada posible ubicación* for (path in possibleStoragePaths) {  
 val modelDir = File(path)  
 val modelFile = File(modelDir, MODEL\_FILENAME)  
  
 Log.d(TAG, "Buscando modelo en: ${modelFile.*absolutePath*}")  
  
 if (modelFile.exists() && modelFile.canRead()) {  
 Log.d(TAG, "Modelo encontrado en: ${modelFile.*absolutePath*}")  
 Log.d(TAG, "Tamaño del archivo: ${modelFile.length()} bytes")  
  
 \_uiState.*update* **{  
 it**.copy(  
 isLoading = false,  
 modelPath = modelFile.*absolutePath*,  
 statusMessage = "Modelo encontrado en ${modelFile.*absolutePath*}"  
 )  
 **}** return@withContext  
 }  
 }  
  
 *// Si llegamos aquí, no se encontró el modelo* \_uiState.*update* **{  
 it**.copy(  
 isLoading = false,  
 statusMessage = "Modelo no encontrado. Verifica permisos y ubicación."  
 )  
 **}** } catch (e: Exception) {  
 Log.e(TAG, "Error al buscar el modelo", e)  
 \_uiState.*update* **{  
 it**.copy(  
 isLoading = false,  
 statusMessage = "Error al buscar el modelo: ${e.message}"  
 )  
 **}** }  
 **}  
 }** }  
  
 fun loadModel(modelPath: String) {  
 *viewModelScope*.*launch* **{** withContext(Dispatchers.IO) **{** try {  
 \_uiState.*update* **{ it**.copy(isLoading = true, statusMessage = "Cargando modelo...") **}** val success = llamaInterface.loadModel(modelPath)  
  
 if (success) {  
 \_uiState.*update* **{  
 it**.copy(  
 isLoading = false,  
 isModelLoaded = true,  
 statusMessage = "Modelo cargado exitosamente"  
 )  
 **}** } else {  
 \_uiState.*update* **{  
 it**.copy(  
 isLoading = false,  
 isModelLoaded = false,  
 statusMessage = "Error al cargar el modelo"  
 )  
 **}** }  
 } catch (e: Exception) {  
 Log.e(TAG, "Error al cargar el modelo", e)  
 \_uiState.*update* **{  
 it**.copy(  
 isLoading = false,  
 isModelLoaded = false,  
 statusMessage = "Error al cargar el modelo: ${e.message}"  
 )  
 **}** }  
 **}  
 }** }  
  
 *// Método original - mantenerlo para compatibilidad* fun generateResponse(prompt: String) {  
 if (!llamaInterface.isModelLoaded()) {  
 \_uiState.*update* **{ it**.copy(statusMessage = "Modelo no cargado") **}** return  
 }  
  
 *viewModelScope*.*launch* **{** withContext(Dispatchers.IO) **{** try {  
 \_uiState.*update* **{  
 it**.copy(  
 isGenerating = true,  
 statusMessage = "Generando respuesta..."  
 )  
 **}** val response = llamaInterface.generateResponse(prompt, 200)  
  
 \_uiState.*update* **{  
 it**.copy(  
 isGenerating = false,  
 response = response,  
 statusMessage = "Respuesta generada"  
 )  
 **}** } catch (e: Exception) {  
 Log.e(TAG, "Error al generar respuesta", e)  
 \_uiState.*update* **{  
 it**.copy(  
 isGenerating = false,  
 statusMessage = "Error al generar respuesta: ${e.message}"  
 )  
 **}** }  
 **}  
 }** }  
  
 *// Nuevo método para generar respuesta con streaming* fun generateResponseStreaming(prompt: String) {  
 if (!llamaInterface.isModelLoaded()) {  
 \_uiState.*update* **{ it**.copy(statusMessage = "Modelo no cargado") **}** return  
 }  
  
 *viewModelScope*.*launch* **{** withContext(Dispatchers.IO) **{** try {  
 \_uiState.*update* **{  
 it**.copy(  
 isGenerating = true,  
 statusMessage = "Generando respuesta...",  
 response = "" *// Limpiar la respuesta anterior* )  
 **}** *// Crear callback que recibirá tokens en tiempo real* val tokenCallback = object : TokenCallback {  
 override fun onToken(token: String) {  
 *viewModelScope*.*launch*(Dispatchers.Main) **{** \_uiState.*update* **{  
 it**.copy(response = **it**.response + token)  
 **}  
 }** }  
 }  
  
 *// Llamar al método con streaming* val finalResponse = llamaInterface.generateResponseStreaming(prompt, 200, tokenCallback)  
  
 \_uiState.*update* **{  
 it**.copy(  
 isGenerating = false,  
 statusMessage = "Respuesta generada"  
 )  
 **}** } catch (e: Exception) {  
 Log.e(TAG, "Error al generar respuesta streaming", e)  
 \_uiState.*update* **{  
 it**.copy(  
 isGenerating = false,  
 statusMessage = "Error al generar respuesta: ${e.message}"  
 )  
 **}** }  
 **}  
 }** }  
  
 fun updatePrompt(prompt: String) {  
 \_uiState.*update* **{ it**.copy(prompt = prompt) **}** }  
  
 override fun onCleared() {  
 super.onCleared()  
 llamaInterface.unloadModel()  
 }  
}  
  
data class LlamaUiState(  
 val isLoading: Boolean = false,  
 val isGenerating: Boolean = false,  
 val isModelLoaded: Boolean = false,  
 val modelPath: String = "",  
 val prompt: String = "",  
 val response: String = "",  
 val statusMessage: String = "Esperando permisos...",  
 val diagnosticInfo: String = ""  
)

**Configuración de la Activity principal**

class MainActivity : ComponentActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContent {

MyLlamaAppTheme {

Surface(

modifier = Modifier.fillMaxSize(),

color = MaterialTheme.colorScheme.background

) {

LlamaApp()

}

}

}

}

}

**Composable principal - Arquitectura reactiva**

@OptIn(ExperimentalMaterial3Api::class)

@Composable

fun LlamaApp() {

val viewModel: LlamaViewModel = viewModel()

val uiState by viewModel.uiState.collectAsStateWithLifecycle()

val context = LocalContext.current

var showDiagnosticInfo by remember { mutableStateOf(false) }

**Gestión inteligente de permisos**

PermissionsHandler(

onPermissionsGranted = {

*// Cuando los permisos son concedidos, buscamos el modelo*

viewModel.findModelInExternalStorage()

},

onPermissionsDenied = {

*// Cuando los permisos son denegados, mostramos información*

Log.d("LlamaApp", "Permisos denegados")

}

)

**Encabezado de estado - Información centralizada**

Card(

modifier = Modifier.fillMaxWidth()

) {

Column(

modifier = Modifier.padding(16.dp)

) {

Text(

text = "LLaMA.cpp para Android",

style = MaterialTheme.typography.headlineSmall,

fontWeight = FontWeight.Bold

)

Spacer(modifier = Modifier.height(8.dp))

Text(

text = "Estado: ${uiState.statusMessage}",

style = MaterialTheme.typography.bodyMedium

)

**Botones principales de acción**

Row(

modifier = Modifier.fillMaxWidth(),

horizontalArrangement = Arrangement.SpaceBetween,

verticalAlignment = Alignment.CenterVertically

) {

*// Botón para cargar/buscar modelo*

Button(

onClick = {

if (uiState.modelPath.isNotEmpty()) {

viewModel.loadModel(uiState.modelPath)

} else {

viewModel.findModelInExternalStorage()

}

},

enabled = !uiState.isLoading && !uiState.isModelLoaded

) {

Text(if (uiState.modelPath.isEmpty()) "Buscar Modelo" else "Cargar Modelo")

}

*// Indicador de estado del modelo*

Text(

text = if (uiState.isModelLoaded) "Modelo Cargado ✓" else "Modelo No Cargado",

modifier = Modifier.padding(horizontal = 8.dp)

)

}

**Herramientas de diagnóstico y permisos**

Row(

modifier = Modifier.fillMaxWidth(),

horizontalArrangement = Arrangement.SpaceBetween

) {

*// Botón de diagnóstico*

OutlinedButton(

onClick = {

viewModel.debugStorageAccess(context)

showDiagnosticInfo = true

},

modifier = Modifier.weight(1f)

) {

Text("Diagnóstico")

}

Spacer(modifier = Modifier.width(8.dp))

*// Botón para solicitar permisos especiales*

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.R) {

OutlinedButton(

onClick = {

val intent = Intent(Settings.ACTION\_MANAGE\_APP\_ALL\_FILES\_ACCESS\_PERMISSION)

intent.data = Uri.parse("package:${context.packageName}")

context.startActivity(intent)

},

modifier = Modifier.weight(1f)

) {

Text("Solicitar Permisos")

}

} else {

OutlinedButton(

onClick = {

val intent = Intent(Settings.ACTION\_APPLICATION\_DETAILS\_SETTINGS)

intent.data = Uri.fromParts("package", context.packageName, null)

context.startActivity(intent)

},

modifier = Modifier.weight(1f)

) {

Text("Permisos")

}

}

}

**Diálogo de diagnóstico modal**

if (showDiagnosticInfo && uiState.diagnosticInfo.isNotEmpty()) {

AlertDialog(

onDismissRequest = { showDiagnosticInfo = false },

title = { Text("Diagnóstico de Almacenamiento") },

text = {

Text(

text = uiState.diagnosticInfo,

modifier = Modifier

.verticalScroll(rememberScrollState())

.heightIn(max = 400.dp)

)

},

confirmButton = {

Button(onClick = { showDiagnosticInfo = false }) {

Text("Cerrar")

}

}

)

}

**Área de entrada de texto**

OutlinedTextField(

value = uiState.prompt,

onValueChange = { viewModel.updatePrompt(it) },

label = { Text("Ingresa tu prompt") },

modifier = Modifier

.fillMaxWidth()

.height(120.dp),

enabled = uiState.isModelLoaded && !uiState.isGenerating

)

**Botón de generación inteligente**

Button(

onClick = {

*// Usar el nuevo método con streaming*

viewModel.generateResponseStreaming(uiState.prompt)

},

modifier = Modifier.align(Alignment.End),

enabled = uiState.isModelLoaded && !uiState.isGenerating && uiState.prompt.isNotBlank()

) {

Text(if (uiState.isGenerating) "Generando..." else "Generar Respuesta")

}

**Área de visualización de respuesta**

Surface(

modifier = Modifier

.fillMaxWidth()

.weight(1f),

color = MaterialTheme.colorScheme.surfaceVariant

) {

Column(

modifier = Modifier

.padding(16.dp)

.verticalScroll(rememberScrollState())

) {

Text(

text = "Respuesta:",

style = MaterialTheme.typography.titleMedium,

fontWeight = FontWeight.Bold

)

Spacer(modifier = Modifier.height(8.dp))

Text(

text = uiState.response.ifBlank { "La respuesta aparecerá aquí" },

style = MaterialTheme.typography.bodyMedium

)

}

}

**Patrones de diseño implementados**

**1. Estado unidireccional**

UI → ViewModel → Estado → UI (recomposición)

**2. Composición declarativa**

@Composable

fun Component() {

if (condition) {

ShowThis()

} else {

ShowThat()

}

}

Código **MainActivity.kt**:

package com.example.myllamaapp  
  
import android.Manifest  
import android.content.Intent  
import android.net.Uri  
import android.os.Build  
import android.os.Bundle  
import android.os.Environment  
import android.provider.Settings  
import android.util.Log  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.setContent  
import androidx.compose.foundation.layout.\*  
import androidx.compose.foundation.rememberScrollState  
import androidx.compose.foundation.verticalScroll  
import androidx.compose.material3.\*  
import androidx.compose.runtime.\*  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.platform.*LocalContext*import androidx.compose.ui.text.font.FontWeight  
import androidx.compose.ui.unit.dp  
import androidx.lifecycle.viewmodel.compose.viewModel  
import androidx.lifecycle.compose.collectAsStateWithLifecycle  
import com.example.myllamaapp.ui.theme.MyLlamaAppTheme  
  
class MainActivity : ComponentActivity() {  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *setContent* **{** MyLlamaAppTheme **{** Surface(  
 modifier = Modifier.*fillMaxSize*(),  
 color = MaterialTheme.colorScheme.background  
 ) **{** LlamaApp()  
 **}  
 }  
 }** }  
}  
  
@OptIn(ExperimentalMaterial3Api::class)  
@Composable  
fun LlamaApp() {  
 val viewModel: LlamaViewModel = viewModel()  
 val uiState by viewModel.uiState.collectAsStateWithLifecycle()  
 val context = *LocalContext*.current  
  
 *// Estado para mostrar información de diagnóstico* var showDiagnosticInfo by remember **{** *mutableStateOf*(false) **}** *// Gestión de permisos usando nuestro componente personalizado* PermissionsHandler(  
 onPermissionsGranted = **{** *// Cuando los permisos son concedidos, buscamos el modelo* viewModel.findModelInExternalStorage()  
 **}**,  
 onPermissionsDenied = **{** *// Cuando los permisos son denegados, mostramos información* Log.d("LlamaApp", "Permisos denegados")  
 **}** )  
  
 *// Interfaz principal* Column(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*padding*(16.*dp*)  
 ) **{** *// Área superior: Estado y controles* Card(  
 modifier = Modifier.*fillMaxWidth*()  
 ) **{** Column(  
 modifier = Modifier.*padding*(16.*dp*)  
 ) **{** Text(  
 text = "LLaMA.cpp para Android",  
 style = MaterialTheme.typography.headlineSmall,  
 fontWeight = FontWeight.Bold  
 )  
  
 Spacer(modifier = Modifier.*height*(8.*dp*))  
  
 Text(  
 text = "Estado: ${uiState.statusMessage}",  
 style = MaterialTheme.typography.bodyMedium  
 )  
  
 Spacer(modifier = Modifier.*height*(16.*dp*))  
  
 *// Información sobre la ubicación del modelo* if (uiState.modelPath.*isNotEmpty*()) {  
 Text(  
 text = "Modelo: ${uiState.modelPath}",  
 style = MaterialTheme.typography.bodySmall  
 )  
  
 Spacer(modifier = Modifier.*height*(8.*dp*))  
 }  
  
 *// Fila de botones principales* Row(  
 modifier = Modifier.*fillMaxWidth*(),  
 horizontalArrangement = Arrangement.SpaceBetween,  
 verticalAlignment = Alignment.CenterVertically  
 ) **{** *// Botón para cargar/buscar modelo* Button(  
 onClick = **{** if (uiState.modelPath.*isNotEmpty*()) {  
 viewModel.loadModel(uiState.modelPath)  
 } else {  
 viewModel.findModelInExternalStorage()  
 }  
 **}**,  
 enabled = !uiState.isLoading && !uiState.isModelLoaded  
 ) **{** Text(if (uiState.modelPath.*isEmpty*()) "Buscar Modelo" else "Cargar Modelo")  
 **}** *// Indicador de estado del modelo* Text(  
 text = if (uiState.isModelLoaded) "Modelo Cargado ✓" else "Modelo No Cargado",  
 modifier = Modifier.*padding*(horizontal = 8.*dp*)  
 )  
 **}** Spacer(modifier = Modifier.*height*(8.*dp*))  
  
 *// Botones adicionales para diagnóstico y permisos* Row(  
 modifier = Modifier.*fillMaxWidth*(),  
 horizontalArrangement = Arrangement.SpaceBetween  
 ) **{** *// Botón de diagnóstico* OutlinedButton(  
 onClick = **{** viewModel.debugStorageAccess(context)  
 showDiagnosticInfo = true  
 **}**,  
 modifier = Modifier.*weight*(1f)  
 ) **{** Text("Diagnóstico")  
 **}** Spacer(modifier = Modifier.*width*(8.*dp*))  
  
 *// Botón para solicitar permisos especiales* if (Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*R*) {  
 OutlinedButton(  
 onClick = **{** val intent = Intent(Settings.*ACTION\_MANAGE\_APP\_ALL\_FILES\_ACCESS\_PERMISSION*)  
 intent.*data* = Uri.parse("package:${context.*packageName*}")  
 context.startActivity(intent)  
 **}**,  
 modifier = Modifier.*weight*(1f)  
 ) **{** Text("Solicitar Permisos")  
 **}** } else {  
 OutlinedButton(  
 onClick = **{** val intent = Intent(Settings.*ACTION\_APPLICATION\_DETAILS\_SETTINGS*)  
 intent.*data* = Uri.fromParts("package", context.*packageName*, null)  
 context.startActivity(intent)  
 **}**,  
 modifier = Modifier.*weight*(1f)  
 ) **{** Text("Permisos")  
 **}** }  
 **}  
 }  
 }** *// Diálogo para mostrar información de diagnóstico* if (showDiagnosticInfo && uiState.diagnosticInfo.*isNotEmpty*()) {  
 AlertDialog(  
 onDismissRequest = **{** showDiagnosticInfo = false **}**,  
 title = **{** Text("Diagnóstico de Almacenamiento") **}**,  
 text = **{** Text(  
 text = uiState.diagnosticInfo,  
 modifier = Modifier  
 .*verticalScroll*(rememberScrollState())  
 .*heightIn*(max = 400.*dp*)  
 )  
 **}**,  
 confirmButton = **{** Button(onClick = **{** showDiagnosticInfo = false **}**) **{** Text("Cerrar")  
 **}  
 }** )  
 }  
  
 Spacer(modifier = Modifier.*height*(16.*dp*))  
  
 *// Área media: Entrada del prompt* OutlinedTextField(  
 value = uiState.prompt,  
 onValueChange = **{** viewModel.updatePrompt(**it**) **}**,  
 label = **{** Text("Ingresa tu prompt") **}**,  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*height*(120.*dp*),  
 enabled = uiState.isModelLoaded && !uiState.isGenerating  
 )  
  
 Spacer(modifier = Modifier.*height*(8.*dp*))  
  
 *// En la función LlamaApp() que contiene la UI principal, modifica el botón de generación:* Button(  
 onClick = **{** *// Usar el nuevo método con streaming* viewModel.generateResponseStreaming(uiState.prompt)  
 **}**,  
 modifier = Modifier.*align*(Alignment.End),  
 enabled = uiState.isModelLoaded && !uiState.isGenerating && uiState.prompt.*isNotBlank*()  
 ) **{** Text(if (uiState.isGenerating) "Generando..." else "Generar Respuesta")  
 **}** Spacer(modifier = Modifier.*height*(16.*dp*))  
  
 *// Área inferior: Visualización de respuesta* Surface(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*weight*(1f),  
 color = MaterialTheme.colorScheme.surfaceVariant  
 ) **{** Column(  
 modifier = Modifier  
 .*padding*(16.*dp*)  
 .*verticalScroll*(rememberScrollState())  
 ) **{** Text(  
 text = "Respuesta:",  
 style = MaterialTheme.typography.titleMedium,  
 fontWeight = FontWeight.Bold  
 )  
  
 Spacer(modifier = Modifier.*height*(8.*dp*))  
  
 Text(  
 text = uiState.response.*ifBlank* **{** "La respuesta aparecerá aquí" **}**,  
 style = MaterialTheme.typography.bodyMedium  
 )  
 **}  
 }  
 }**}