在 Android 项目中使用 JNI 和 C++ 实现类似 librosa 中 LogmelFilterBank 的功能，涉及到以下几个步骤：

1. **计算 Mel 频率谱图 (Mel-spectrogram)**：
   - 这一步通常包括对音频信号进行短时傅里叶变换（STFT），并使用 Mel 滤波器组将其从频率域转换到 Mel 频率域。
2. **计算对数 Mel 频率谱图**：
   - 在计算 Mel 频率谱图之后，通常对其进行对数变换，以使其更符合人耳的听觉特性。

为了实现这个功能，我们将使用以下步骤：

- 使用 STFT（短时傅里叶变换）计算频谱。
- 创建 Mel 滤波器组。
- 计算 Mel 频率谱图。
- 进行对数转换以获得 LogMel 频率谱图。

1. C++ 实现 LogMelFilterBank

我们可以通过 JNI 在 C++ 中实现这些步骤。为了简化过程，下面将使用 C++ 和 FFTW（快速

傅里叶变换库）来进行 STFT，并手动创建 Mel 滤波器组，最后计算对数 Mel 频率谱图。

示例代码

步骤 1：依赖库

- 使用 FFTW 库进行傅里叶变换。
- 手动计算 Mel 滤波器和 Mel 频率谱图。

你需要将 FFTW 集成到你的 Android 项目中。可以通过下载并编译 FFTW 或使用 NDK 管理器来引入它。

步骤 2：编写 C++ 实现

```
#include <jni.h>
#include <vector>
#include <cmath>
#include <fftw3.h>

#define PI 3.14159265358979323846

// 创建 Mel 滤波器组
std::vector<std::vector<float>>
createMelFilterBank(int nfft, int
n_mels, int sample_rate) {
```

```
    std::vector<std::vector<float>>
mel_filters(n_mels,
std::vector<float>(nfft / 2 + 1, 0));

    // Mel 滤波器的计算
    float min_freq = 0.0f;
    float max_freq = sample_rate /
2.0f;
    float min_mel = 2595 * log10(1 +
min_freq / 700.0f);
    float max_mel = 2595 * log10(1 +
max_freq / 700.0f);

    // 计算 Mel 滤波器中心频率
    std::vector<float>
mel_freqs(n_mels);
    for (int m = 0; m < n_mels; ++m) {
        mel_freqs[m] = min_mel +
(max_mel - min_mel) * m / (n_mels -
1);
    }

    // 将 Mel 频率转换回 Hz
    std::vector<float>
hz_freqs(n_mels);
    for (int m = 0; m < n_mels; ++m) {
        hz_freqs[m] = 700.0f *
(pow(10.0f, mel_freqs[m] / 2595.0f) -
1);
```

```
    }

    // 构建 Mel 滤波器组
    for (int m = 0; m < n_mels; ++m) {
        float f1 = hz_freqs[m];
        float f2 = (m + 1 < n_mels) ?
hz_freqs[m + 1] : sample_rate / 2.0f;
        for (int k = 0; k < nfft / 2 +
1; ++k) {
            float freq = k *
(sample_rate / 2.0f) / (nfft / 2);
            if (freq >= f1 && freq <=
f2) {
                mel_filters[m][k] =
(freq - f1) / (f2 - f1);
            }
        }
    }

    return mel_filters;
}

// 计算 STFT
std::vector<std::vector<float>>
computeSTFT(const std::vector<float>&
signal, int nfft, int hop_size) {
    int num_frames = (signal.size() -
nfft) / hop_size + 1;
```

```
    std::vector<std::vector<float>>
stft(num_frames,
std::vector<float>(nfft / 2 + 1, 0));

    fftwf_complex* out =
(fftwf_complex*)
fftwf_malloc(sizeof(fftwf_complex) *
nfft);
    fftwf_plan plan =
fftwf_plan_dft_r2c_1d(nfft, nullptr,
out, FFTW_ESTIMATE);

    std::vector<float> frame(nfft);
    for (int i = 0; i < num_frames; +
+i) {
        int offset = i * hop_size;

        // 获取音频帧并应用窗函数（例如
Hamming 窗）
        for (int j = 0; j < nfft; ++j)
{
            frame[j] = signal[offset +
j] * 0.54 - 0.46 * cos(2 * PI * j /
(nfft - 1));
        }

        // 执行傅里叶变换
        fftwf_execute_dft_r2c(plan,
frame.data(), out);
```

```
            // 计算频谱
            for (int j = 0; j < nfft / 2 +
1; ++j) {
                stft[i][j] =
std::sqrt(out[j][0] * out[j][0] +
out[j][1] * out[j][1]);
            }
        }

        fftwf_free(out);
        fftwf_destroy_plan(plan);

        return stft;
}

// 计算 Log-Mel 频率谱图
std::vector<std::vector<float>>
computeLogMelSpectrogram(const
std::vector<float>& signal, int nfft,
int hop_size, int n_mels, int
sample_rate) {
        // 计算 STFT
        std::vector<std::vector<float>>
stft = computeSTFT(signal, nfft,
hop_size);

        // 创建 Mel 滤波器组
        std::vector<std::vector<float>>
mel_filters =
```

```
createMelFilterBank(nfft, n_mels,
sample_rate);

    // 计算 Mel 频率谱图
    std::vector<std::vector<float>>
mel_spectrogram(stft.size(),
std::vector<float>(n_mels, 0));
    for (int i = 0; i < stft.size(); +
+i) {
        for (int m = 0; m < n_mels; +
+m) {
            for (int k = 0; k < nfft /
2 + 1; ++k) {
                mel_spectrogram[i][m]
+= stft[i][k] * mel_filters[m][k];
            }
        }
    }

    // 对 Mel 频率谱图进行对数变换
    for (int i = 0; i <
mel_spectrogram.size(); ++i) {
        for (int j = 0; j <
mel_spectrogram[i].size(); ++j) {
            mel_spectrogram[i][j] =
logf(mel_spectrogram[i][j] + 1e-6); //
添加小常数以避免 log(0)
        }
    }
```

```
    return mel_spectrogram;
}

// JNI 方法：加载音频并计算 Log-Mel 频率谱图
extern "C"
JNIEXPORT jobjectArray JNICALL
Java_com_example_audio_AudioProcessor_computeLogMelSpectrogram(JNIEnv *env, jobject thiz, jfloatArray audio_data, jint nfft, jint hop_size, jint n_mels, jint sample_rate) {
    // 获取音频数据
    jfloat* signal = env->GetFloatArrayElements(audio_data, nullptr);
    jsize signal_length = env->GetArrayLength(audio_data);

    // 将信号转换为 std::vector
    std::vector<float> signal_vector(signal, signal + signal_length);

    // 计算 Log-Mel 频率谱图
    std::vector<std::vector<float>> logmel_spectrogram = computeLogMelSpectrogram(signal_vector
```

```
, nfft, hop_size, n_mels,
sample_rate);

    // 将结果转换为 JNI 数组返回
    jclass floatArrayClass = env-
>FindClass("[F");
    jobjectArray resultArray = env-
>NewObjectArray(logmel_spectrogram.siz
e(), floatArrayClass, nullptr);

    for (int i = 0; i <
logmel_spectrogram.size(); ++i) {
        jfloatArray rowArray = env-
>NewFloatArray(logmel_spectrogram[i].s
ize());
        env-
>SetFloatArrayRegion(rowArray, 0,
logmel_spectrogram[i].size(),
logmel_spectrogram[i].data());
        env-
>SetObjectArrayElement(resultArray, i,
rowArray);
    }

    // 释放 JNI 数组
    env-
>ReleaseFloatArrayElements(audio_data,
signal, JNI_ABORT);

    return resultArray;
```

```
}
```

2. Java 接口

在 Java 中，我们通过 JNI 调用 C++ 编写的函数，传递音频数据并返回 Log-Mel 频率谱图。

```
package com.example.audio;

public class AudioProcessor {
    static {

System.loadLibrary("audio_processor");
    }

    // Native
```