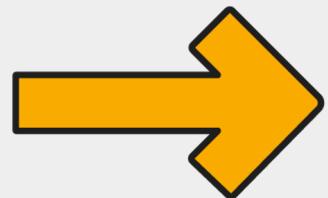


第二部分

# 我们今天的 Code Lab

今天我们要完成一个什么样的App?如何完成?



# Gemini CodeLab

使用Gemini的API来实现一个简单的聊天应用界面。

## 基本功能

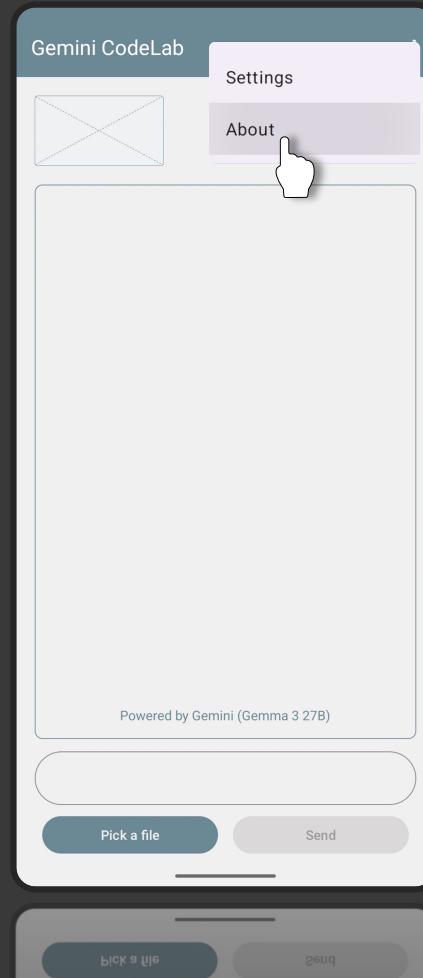
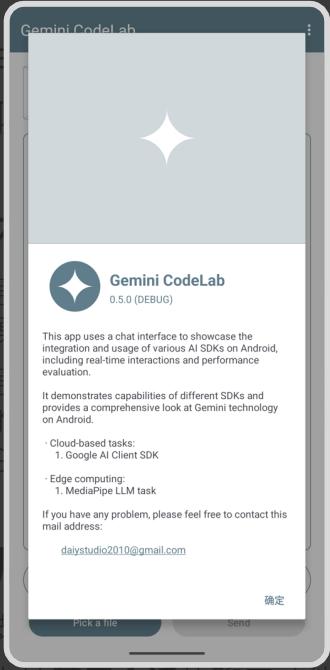
- 通过使用**Gemini云端API**，使用不同的模型，进行基本的问答
- 通过使用**MediaPipe提供的Android端的API**，使用Gemma 2 2B或者Gemma3 1B模型进行基本的问答

## 扩展功能

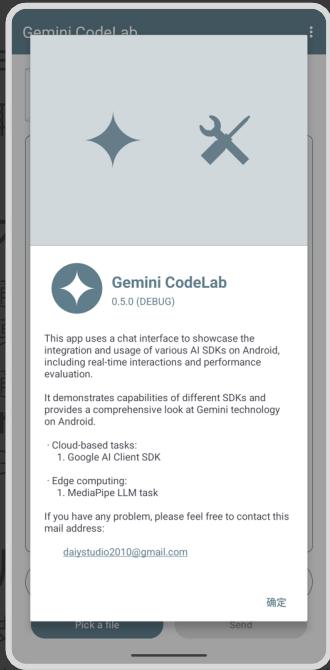
- 支持**图片**输入和**文档**输入
- 支持**基于会话**的聊天功能



# Gemini CodeLab

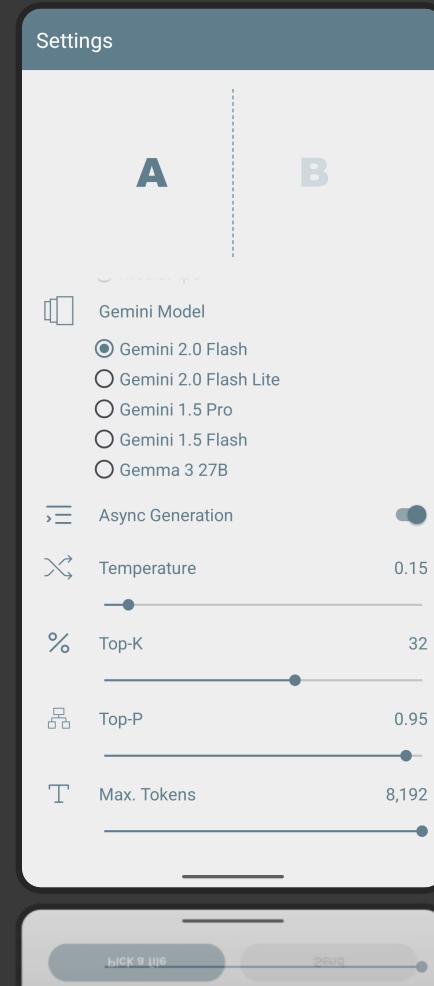


# Gemini CodeLab 调试模式



点击顶部插画5次

更多设置选项



统计数据

# Gemini CodeLab

使用Gemini的API来实现一个简单的聊天应用界面。

## 基本功能

- 通过使用**Gemini云端API**，使用不同的模型，进行基本的问答
- 通过使用**MediaPipe提供的Android端的API**，使用Gemma 2 2B或者Gemma3 1B模型进行基本的问答

## 扩展功能

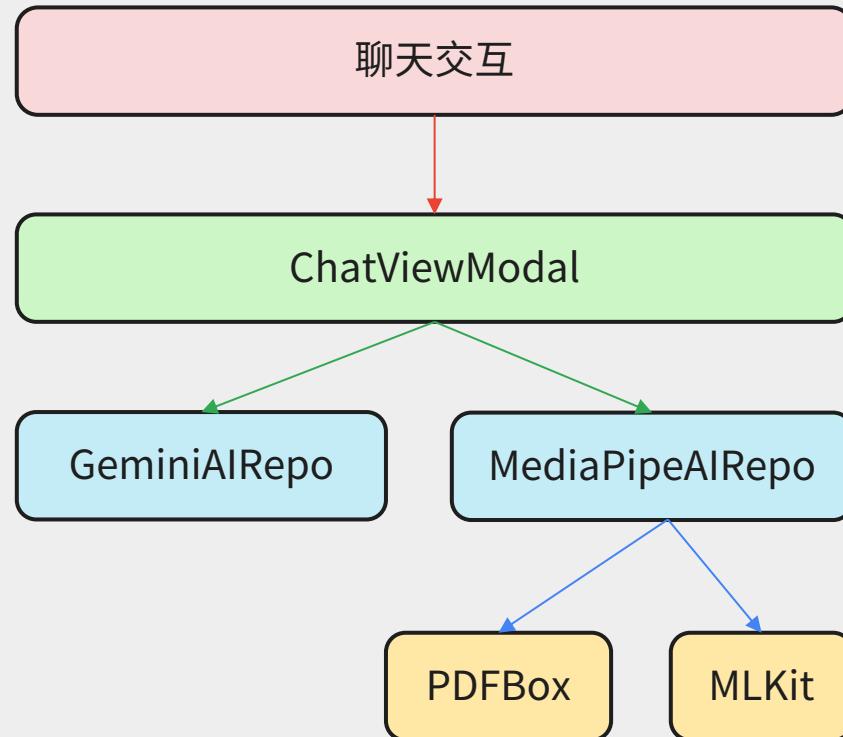
- 支持**图片**输入和**文档**输入
- 支持**基于会话**的聊天功能



# 程序架构

我们使用标准MVVM架构来程序的结构分层更加清晰，职责明确：

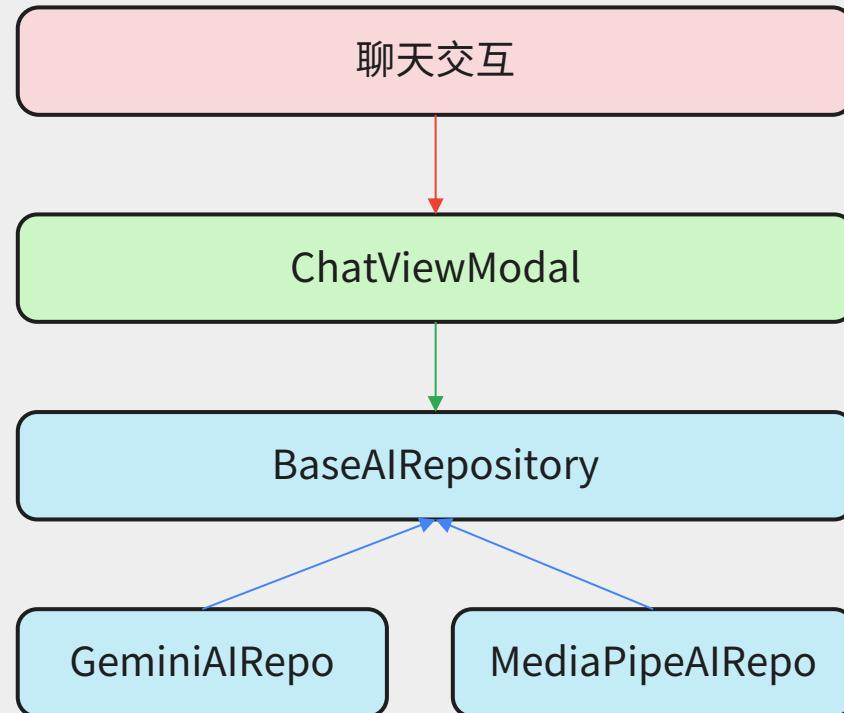
- **应用界面**：只负责展示，不处理数据逻辑
- **ViewModel**：处理上层应用所需的数据、状态和操作逻辑
- **数据仓库**：只负责和数据相关的逻辑，直接和AI引擎直接交互



# 程序架构

我们使用标准MVVM架构来程序的结构分层更加清晰，职责明确：

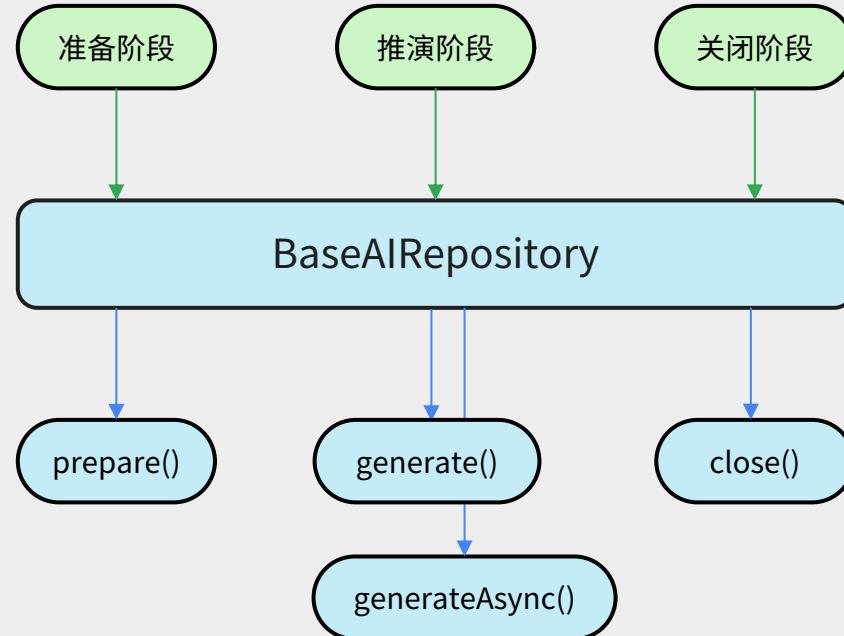
- **应用界面**：只负责数据展示，不处理任何相关的逻辑
- **ViewModel**：处理上层应用所需的数据、状态和操作逻辑
- **数据仓库**：只负责和数据相关的逻辑，直接和AI引擎直接交互



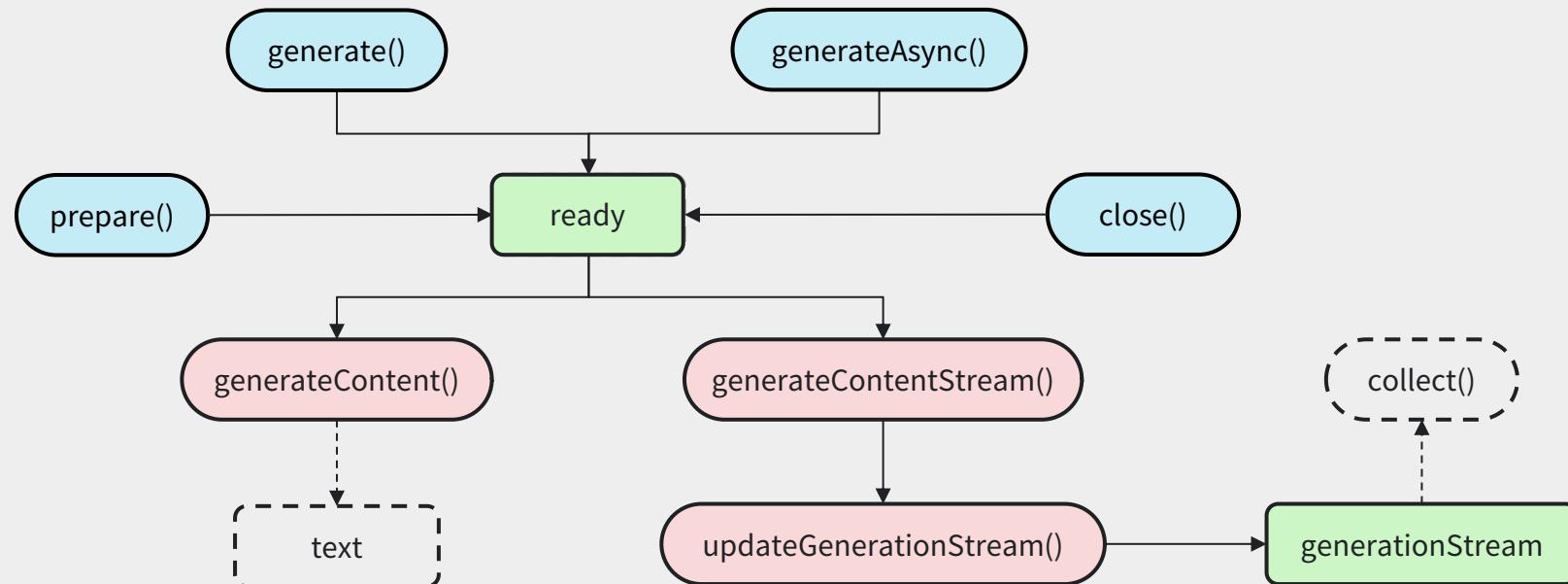
# 基本流程

一个完整的AI引擎的使用流程，主要分为以下几个步骤：

- **准备阶段**: AI引擎初始化，加载相应的模型资源，设置相关的参数
- **推演阶段**: 将用户输入的文字，媒体资源封装，传给AI模型，将模型输出处理后返回给用户
- **关闭阶段**: 停止相关的操作，释放引擎占用的系统资源



# BaseAIRrepository



# BaseAIRrepository

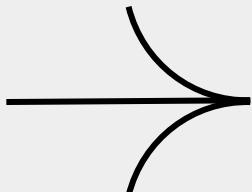
继承的类需要根据自己的情况，使用不同的API去实现这四个方法：

`prepare()`

`generateContent()`

`generateContentStream()`

`close()`



## GeminiAIRrepository

`GenerativeModel()`

`generateContent()`

`generateContentStream()`

## MediaPipeAIRrepository

`LlmInference()`    `LlmInferenceSession()`

`generateResponse()`

`generateResponseAsync()`

`session.close()`    `inference.close()`

# MediaPipeAIRepository



## 准备阶段

创建一个`LlmInference`的实例，用  
来做后续的推演。

- `LlmInferenceSession`是可选的，不是  
必须创建
- 需要把对应的**模型放到指定的位置**，否  
则引擎无法正确加载

## 推演阶段

使用同步生成或者流式生成API生成  
内容。

- 本地模型不支持多模态，所以**图片或者文  
档内容需要自己预处理**
- 尽量让模型**生成精简的回复**，避免结尾的  
内容重复

## 关闭阶段

关闭模型相关的实例，释放系统资  
源。

- MediaPipe LLM只有**0.10.22**可以支持  
**Gemma3 1B**的模型
- 关闭模型前需要**停止推演**，否则需要捕  
捉状态异常

# MediaPipeAIRepository

```
override fun prepare() {  
    val ready = if (!modelExists) {  
        false  
    } else {  
        val inferenceOptions = LlmInferenceOptions.builder()  
            .setModelPath(modelPath).build()  
        val sessionOptions = LlmInferenceSession.LlmInferenceSessionOptions.builder().build()  
  
        llmInference = LlmInference.createFromOptions(context, inferenceOptions)  
        llmSession = LlmInferenceSession.createFromOptions(llmInference, sessionOptions)  
  
        true  
    }  
    setReady(ready)  
}
```

Session的建立不是必须的，但可以有更多配置

# MediaPipeAIRepository

```
override fun prepare() {  
    val ready = if (!modelExists) {  
        false  
    } else {  
        val inferenceOptions = LlmInference.LlmInferenceOptions.builder()  
            .setModelPath(modelPath).build()  
        val sessionOptions = LlmInferenceSession.LlmInferenceSessionOptions.builder().build()  
  
        llmInference = LlmInference.createFromOptions(context, inferenceOptions)  
        llmSession = LlmInferenceSession.createFromOptions(llmInference, sessionOptions)  
  
        true  
    }  
}  
}  
  
} setReady(ready)
```



不要忘记调用  
setReady()

# MediaPipeAIRepository

```
override suspend fun generateContent(  
    prompt: String,  
    fileUri: String?,  
    mimeType: String?  
): String? {  
    buildContent(prompt, fileUri, mimeType)  
    return llmSession?.generateResponse()  
}
```

```
override suspend fun generateContentStream(  
    prompt: String,  
    fileUri: String?,  
    mimeType: String?  
){  
    buildContent(prompt, fileUri, mimeType)  
    llmSession?.generateResponseAsync() { result, done ->  
        updateGenerationStream(  
            text = if (done) "" else result,  
            status = if (done) Status.DONE else Status.RUNNING  
        )  
    }  
}
```

根据用户输入  
生成提示词

# MediaPipeAIRepository

```
override suspend fun generateContent(  
    prompt: String,  
    fileUri: String?,  
    mimeType: String?  
): String?  
{  
    return llmSession?.generateResponse()  
}
```



二进制数据



图片文件

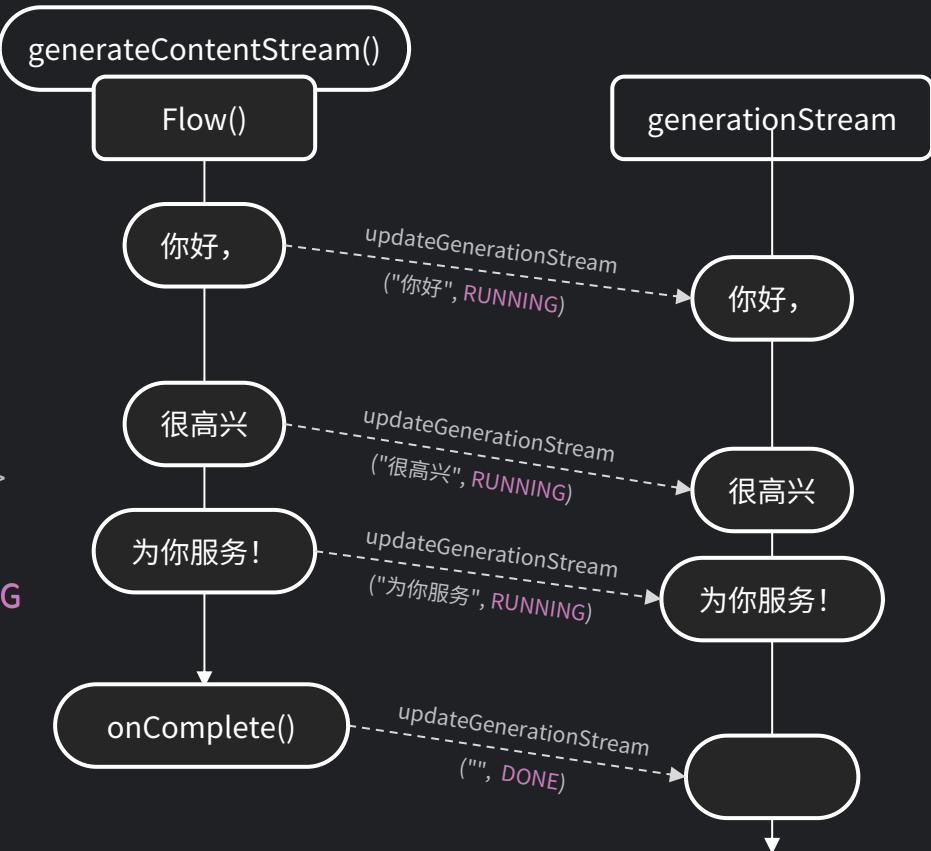


文本提示词

```
buildContent(prompt, fileUri, mimeType) {  
  
    val extractedContent =  
        if (mimeType.contains("pdf")) {  
            StatsUtils.measure("PDF") {  
                ContentUtils.extractTextFromPdf(fileUri)  
            }  
        } else if (mimeType.contains("image/")) {  
            StatsUtils.measureSuspend("IMAGE") {  
                ContentUtils.extractTextFromImage(fileUri)  
            }  
        } else {  
            ""  
        }  
  
    val composedPrompt = buildString {  
        append(extractedContent)  
        append("\n $prompt")  
    }  
  
    llmSession?.addQueryChunk(composedPrompt)  
}
```

# MediaPipeAIRepository

```
override suspend fun generateContentStream(  
    prompt: String,  
    fileUri: String?,  
    mimeType: String?  
) {  
    buildContent(prompt, fileUri, mimeType)  
  
    llmSession?.generateResponseAsync() { result, done ->  
        updateGenerationStream(  
            text = if (done) "" else result,  
            status = if (done) Status.DONE else Status.RUNNING  
        )  
    }  
}
```



# MediaPipeAIRepository

```
override fun close() {  
    super.close()  
  
    llmSession?.close()  
    llmInference?.close()  
}
```



只有0.10.22  
可以正确关  
闭且支持  
Gemma3

# 让我们开始吧！

## Code Lab of Gemini on Android

<https://github.com/dailystudio/gemini-codelab-bwa-04-12>

## Gemini API

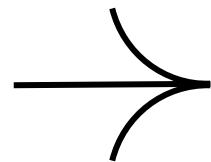
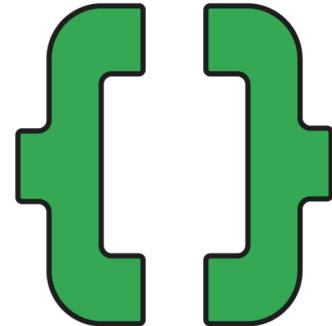
<https://ai.google.dev/gemini-api/docs?hl=zh-cn>

## MediaPipe LLM

[https://ai.google.dev/edge/mediapipe/solutions/genai/llm\\_inference/android](https://ai.google.dev/edge/mediapipe/solutions/genai/llm_inference/android)

## LLM Models

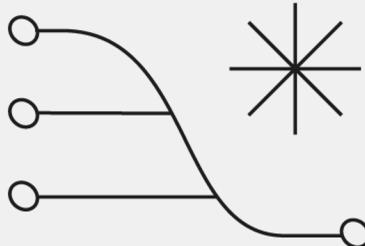
[https://ai.google.dev/edge/mediapipe/solutions/genai/llm\\_inference/index - models](https://ai.google.dev/edge/mediapipe/solutions/genai/llm_inference/index-models)





Google Developer Group

Editable Location



谢谢大家！



{ Build with AI }