M5Module-LLM Arduino API

M5Module-LLM Arduino驱动库API文档。

M5ModuleLLM Class

`M5ModuleLLM` 用于初始化 Module LLM, 并且提供内部成员用于快速初始化 LLM 的各个单元, 方便根据自己的需求构建应用。

```
class M5ModuleLLM {
public:
    bool begin(Stream * targetPort);
    bool checkConnection();
    void update();

    m5_module_llm::ApiSys sys;
    m5_module_llm::ApiLlm llm;
    m5_module_llm::ApiAudio audio;
    m5_module_llm::ApiTts tts;
    m5_module_llm::ApiTts melotts;
    m5_module_llm::ApiKws kws;
    m5_module_llm::ApiAsr asr;
    m5_module_llm::ApiAsr yolo;
    m5_module_llm::ApiVad vad;
    m5_module_llm::ApiWhisper whisper;
    m5_module_llm::ApiDepthAnything depthanything;
    m5_module_llm::ModuleMsg msg;
    m5_module_llm::ModuleComm comm;
private:
};
```

begin

函数原型:

```
bool begin(Stream* targetPort);
```

功能说明:

- 初始化 Module LLM UART 接口配置

传入参数:

- Stream* targetPort:
  - 传入 Serial 指针

返回值:

- bool:

- true: 初始化成功
- false: 初始化失败

checkConnection

函数原型:

```
bool checkConnection();
```

功能说明:

- 发送 `sys.ping` 指令, 检查 Module LLM 连接状态

传入参数:

- null

返回值:

- bool:
  - true: 模组响应
  - false: 模组无响应

update

函数原型:

```
void update();
```

功能说明:

- 拉取 Module LLM UART 响应数据, 该 API 需包含在 Loop 中循环执行。

传入参数:

- null

返回值:

- null

ApiSys Class

`M5ModuleLLM` 的内部成员 `ApiSys sys` 用于控制 SYS 单元实现系统复位等操作。

ping

函数原型:

```
int ping();
```

功能说明:

- 发送sys.ping指令, 检查 Module LLM 连接状态

传入参数:

- null

返回值:

- int:
  - MODULE_LLM_OK / Error Code

reset

函数原型:

```
int reset(bool waitResetFinish = true);
```

功能说明:

- 发送 sys.reset 指令, 复位软件服务。

传入参数:

- bool waitResetFinish:
  - true:阻塞等待复位
  - false:非阻塞执行复位

返回值:

- int:
  - MODULE_LLM_OK / Error Code

reboot

函数原型:

```
int reboot();
```

功能说明:

- 发送 sys.reboot 指令, 复位系统。

传入参数:

- null

返回值:

- int:
  - MODULE_LLM_OK / Error Code

ApiAudio Class

注意：此函数在 1.3 及之后版本已经弃用，改为内部自动配置。

`M5ModuleLLM` 的内部成员 `ApiAudio audio` 用于控制 Audio 单元的初始化和配置。

setup

函数原型:

```
String setup(ApiAudioSetupConfig_t config = ApiAudioSetupConfig_t(), String request_id = "au
```

功能说明:

- 初始化 Audio 单元, 开启系统声卡。(使用 KWS 和 TTS 前需开启该功能)

传入参数:

ApiAudioSetupConfig_t config:

- LLM单元初始化配置:
- String request_id:
  - 会话id, 使用默认即可。

```
struct ApiAudioSetupConfig_t {
    int capcard      = 0;
    int capdevice    = 0;
    float capVolume  = 0.5;
    int playcard     = 0;
    int playdevice   = 1;
    float playVolume = 0.15;
};
```

参数	描述	输入值
capcard	麦克风声卡的索引	系统默认声卡:0
capdevice	麦克风设备索引	板载硅麦:0
capVolume	输入的音量	0.0 ~ 10.0 (1 <volume将增益, 默认值为0.5)
playcard	扬声器声卡的索引	系统默认声卡:0
playdevice	扬声器设备索引	板载扬声器:1
playVolume	输出的音量	0.0 ~ 10.0 (1 <volume将增益, 默认值为0.5)

返回值:

- String:
  - audio_work_id: audio单元work_id

ApiCamera Class

`M5ModuleLLM` 的内部成员 `ApiCamera camera` 用于控制 Camera 单元的初始化和配置。

setup

函数原型:

```
String setup(ApiCameraSetupConfig_t config = ApiCameraSetupConfig_t(), String request_id = "
```

功能说明:

- 初始化 Camera 单元, 开启摄像头输入。(使用 UVC 前需开启该功能)

传入参数:

ApiCameraSetupConfig_t config:

- Camera 单元初始化配置:
- String request_id:
  - 会话id, 使用默认即可。

```
struct ApiCameraSetupConfig_t {
    String response_format = "camera.raw";
    String input           = "/dev/video0";
    bool enoutput          = false;
    int frame_width        = 320;
    int frame_height       = 320;
};
```

参数	描述	输入值
input	UVC 的索引	"/dev/video0"
enoutput	是否串口输出图像数据	启用: true 禁用: false
frame_width	采集图像的宽	320
frame_height	采集图像的高	320

返回值:

- String:
  - camera_work_id: camera 单元 work_id

ApiKws Class

`M5ModuleLLM` 的内部成员 `ApiKws kws` 用于控制 KWS 单元的初始化和配置。

setup

函数原型:

```
String setup(ApiKwsSetupConfig_t config = ApiKwsSetupConfig_t(),
             String request_id = "kws_setup",
             String language = "en_US");
```

功能说明:

- 初始化KWS单元, 并配置唤醒关键字。

传入参数:

ApiKwsSetupConfig_t config:

- KWS单元初始化配置:
- String request_id:
  - 会话id, 使用默认即可。

```
struct ApiKwsSetupConfig_t {
    String kws             = "HELLO";
    String model           = "sherpa-onnx-kws-zipformer-gigaspeech-3.3M-2024-01-01";
    String response_format = "kws.bool";
    String input           = "sys.pcm";
    bool enoutput          = true;
};
```

参数	描述	输入值
model	转换模型	英文模型: "sherpa-onnx-kws-zipformer-gigaspeech-3.3M-2024-01-01" 中文模型: "sherpa-onnx-kws-zipformer-wenetspeech-3.3M-2024-01-01"
kws	KWS唤醒词文本设置	不允许中文/英文混合, 英文要求全大写
enoutput	启用UART输出	启用: true 禁用: false

返回值:

- String:
  - kws_work_id: kws单元work_id

ApiVad Class

`M5ModuleLLM` 的内部成员 `ApiVad vad` 用于控制 VAD 单元的初始化和配置。

setup

函数原型:

```
String setup(ApiVadSetupConfig_t config = ApiVadSetupConfig_t(), String request_id = "vad_se
```

功能说明:

- 初始化 VAD 单元。

传入参数:

ApiVadSetupConfig_t config:

- VAD 单元初始化配置:
- String request_id:
  - 会话id, 使用默认即可。

```
struct ApiKwsSetupConfig_t {
    String model           = "silero-vad";
    String response_format = "vad.bool";
    String input           = {"sys.pcm", "kws.1000"};
    bool enoutput          = true;
};
```

参数	描述	输入值
model	转换模型	模型: "silero-vad"
input	输入	KWS唤醒输入: "kws.xxx"(输入kws单元的work_id) 板载麦克风输入: "sys.pcm" UART流式输入: "vad.wav.stream.base64"
enoutput	启用UART输出	启用: true 禁用: false

返回值:

- String:
  - vad_work_id: vad 单元 work_id

ApiAsr Class

`M5ModuleLLM` 的内部成员 `ApiAsr asr` 用于控制 ASR 单元的初始化和配置。

setup

函数原型:

```
String setup(ApiAsrSetupConfig_t config = ApiAsrSetupConfig_t(), String request_id = "asr_se
             String language = "en_US");
```

功能说明:

- 初始化 ASR 单元, 开启语音转文本功能。

传入参数:

ApiAsrSetupConfig_t config:

- ASR单元初始化配置:
- String request_id:
  - 会话id, 使用默认即可。

```
struct ApiAsrSetupConfig_t {
    String model           = "sherpa-ncnn-streaming-zipformer-20M-2023-02-17";
    String response_format = "asr.utf-8.stream";
    String input           = ["sys.pcm", "kws.1000"];
    bool enoutput          = true;
    float rule1            = 2.4;
    float rule2            = 1.2;
    float rule3            = 30.0;
};
```

参数	描述	输入值
model	转换模型	英文模型: "sherpa-ncnn-streaming-zipformer-20M-2023-02-17" 中文模型: "sherpa-ncnn-streaming-zipformer-zh-14M-2023-02-23"
response_format	输出格式	普通输出: "asr.utf-8" 流式输出: "asr.utf-8.stream"
input	输入	KWS唤醒输入: "kws.xxx"(输入kws单元的work_id) 板载麦克风输入: "sys.pcm" UART流式输入: "asr.wav.stream.base64"
rule1	唤醒到未识别到内容超时时间	单位:秒
rule2	识别最大间隔时间	单位:秒
rule3	识别最长超时时间	单位:秒
enoutput	启用UART输出	启用: true 禁用: false

返回值:

- String:
  - asr_work_id: asr 单元 work_id

ApiWhisper Class

`M5ModuleLLM` 的内部成员 `ApiWhisper whisper` 用于控制 Whisper 单元的初始化和配置。

setup

函数原型:

```
String setup(ApiWhisperSetupConfig_t config = ApiWhisperSetupConfig_t(), String request_id =
```

功能说明:

- 初始化 Whisper 单元, 开启语音转文本功能。

传入参数:

ApiWhisperSetupConfig_t config:

- Whisper 单元初始化配置:
- String request_id:

- 会话id, 使用默认即可。

```
struct ApiAsrSetupConfig_t {
    String model           = "whisper-tiny";
    String response_format = "asr.utf-8";
    String input           = [ "sys.pcm", "kws.1000", "vad.1001" ];
    String language        = "en";
    bool enoutput          = true;
};
```

参数	描述	输入值
model	转换模型	模型: "whisper-tiny"
response_format	输出格式	普通输出: "asr.utf-8"
input	输入	KWS唤醒输入: "kws.xxx"(输入kws单元的work_id) 板载麦克风输入: "sys.pcm" UART流式输入: "asr.wav.stream.base64"
language	用于语言识别的语言	默认 “en” 可选 “zh”, "ja"
enoutput	启用UART输出	启用: true 禁用: false

返回值:

- String:
  - whisper_work_id: whisper 单元 work_id

ApiLlm Class

`M5ModuleLLM` 的内部成员 `ApiLlm llm` 用于控制 LLM 单元的初始化和配置。

setup

函数原型:

```
String setup(ApiLlmSetupConfig_t config = ApiLlmSetupConfig_t(), String request_id = "llm_se
```

功能说明:

- 初始化 LLM 单元, 支持配置 LLM 单元输入输出数据方式。

传入参数:

- ApiLlmSetupConfig_t config:
  - LLM单元初始化配置:
- String request_id:
  - 会话id, 使用默认即可。

```
struct ApiLlmSetupConfig_t {
    String prompt;
    String model           = "qwen2.5-0.5B-prefill-20e";
    String response_format = "llm.utf-8.stream";
    String input           = ["llm.utf-8", "kws.1000"];
    bool enoutput          = true;
    int max_token_len      = 127;
};
```

参数	描述	输入值
response_format	输出格式	普通输出: "llm.utf-8" 流式输出: "llm.utf-8.stream"
enoutput	启用UART输出	启用: true 禁用: false
model	转换模型	预置模型 "qwen2.5-0.5B-prefill-20e"
max_length	配置最大输出token(最大返回推理文本长度)	最大值: 1023
input	输入	ASR输入: "asr.xxx"(输入asr单元的work_id) UART输入: "llm.utf-8" KWS唤醒打断: "kws.xxx"(输入kws单元的work_id)
prompt	模型初始化系统提示词	String

返回值:

- String:
  - llm_work_id: llm单元work_id

inference

函数原型:

```
int inference(String work_id, String input, String request_id = "llm_inference");
```

功能说明:

- 输入数据, 开始推理。返回结果内容将进入 `M5ModuleLLM.msg` 中的 `responseMsgList` 列表容器中。

传入参数:

- String work_id:
  - 调用的LLM单元work_id
- String input:
  - 输入文本
- String request_id:
  - 会话ID, 当同时存在多个会话的时候用于区分。

返回值:

- int:
  - MODULE_LLM_OK / Error Code

inferenceAndWaitResult

函数原型:

```
int inferenceAndWaitResult(String work_id, String input, std::function<void(String&)> onResu
```

功能说明:

- 输入数据, 开始推理。并阻塞等待返回结果, 然后调用 callback 函数。

传入参数:

- String work_id:
  - 调用的 LLM 单元 work_id
- String input:
  - 输入文本
- void onResult(String&)
  - 推理结果 callback 函数
- uint32_t timeout:
  - 等待推理超时时间
- String request_id:
  - 会话ID, 当同时存在多个会话的时候用于区分。

返回值:

- int:
  - MODULE_LLM_OK / Error Code

ApiVlm Class

`M5ModuleLLM` 的内部成员 `ApiVlm vlm` 用于控制 VLM 单元的初始化和配置。

setup

函数原型:

```
String setup(ApiVlmSetupConfig_t config = ApiVlmSetupConfig_t(), String request_id = "vlm_se
```

功能说明:

- 初始化 VLM 单元, 支持配置 VLM 单元输入输出数据方式。

传入参数:

- ApiLlmSetupConfig_t config:
  - LLM单元初始化配置:
- String request_id:
  - 会话id, 使用默认即可。

```
struct ApiVlmSetupConfig_t {
    String prompt;
    String model           = "internvl2.5-1B-ax630c";
    String response_format = "vlm.utf-8.stream";
    String input           = ["vlm.utf-8", "kws.1000"];
    bool enoutput          = true;
    int max_token_len      = 1023;
};
```

参数	描述	输入值
model	转换模型	预置模型 "internvl2.5-1B-ax630c"
response_format	输出格式	普通输出: "vlm.utf-8" 流式输出: "vlm.utf-8.stream"
input	输入	ASR输入: "asr.xxx"(输入asr单元的work_id) UART输入: "llm.utf-8" KWS唤醒打断: "kws.xxx"(输入kws单元的work_id)
max_length	配置最大输出token(最大返回推理文本长度)	最大值: 1023
prompt	模型初始化系统提示词	String
enoutput	启用UART输出	启用: true 禁用: false

返回值:

- String:
  - vlm_work_id: vlm 单元 work_id

inference

函数原型:

```
int inference(String work_id, String input, String request_id = "vlm_inference");
```

功能说明:

- 输入数据, 开始推理。返回结果内容将进入 `M5ModuleLLM.msg` 中的 `responseMsgList` 列表容器中。

传入参数:

- String work_id:
  - 调用的LLM单元work_id
- String input:
  - 输入文本
- String request_id:
  - 会话ID, 当同时存在多个会话的时候用于区分。

返回值:

- int:
  - MODULE_LLM_OK / Error Code

inferenceAndWaitResult

函数原型:

```
int inferenceAndWaitResult(String work_id, String input, std::function<void(String&)> onResu
                           uint32_t timeout = 5000, String request_id = "vlm_inference");
```

功能说明:

- 输入数据, 开始推理。并阻塞等待返回结果, 然后调用 callback 函数。

传入参数:

- String work_id:
  - 调用的 VLM 单元 work_id
- String input:
  - 输入文本
- void onResult(String&)
  - 推理结果 callback 函数
- uint32_t timeout:
  - 等待推理超时时间
- String request_id:

- 会话 ID, 当同时存在多个会话的时候用于区分。

返回值:

- int:
  - MODULE_LLM_OK / Error Code

ApiTts Class

`M5ModuleLLM` 的内部成员 `ApiTts tts` 用于控制 TTS 单元的初始化和配置。

setup

函数原型:

```
String setup(ApiTtsSetupConfig_t config = ApiTtsSetupConfig_t(), String request_id = "tts_se
```

功能说明:

- 初始化TTS单元, 开启文本转语音功能。

传入参数:

ApiTtsSetupConfig_t config:

- LLM单元初始化配置:
- String request_id:
  - 会话id, 使用默认即可。

```
struct ApiTtsSetupConfig_t {
    String model           = "single_speaker_english_fast";
    String response_format = "sys.pcm";
    String input           = ["tts.utf-8.stream", "kws.1000"];
    bool enoutput          = false;
    bool enaudio           = true;
};
```

参数	描述	输入值
model	转换模型	英文模型: "single_speaker_english_fast" 中文模型: "single_speaker_fast"
input	输入	LLM输入: "llm.xxx"(输入llm单元的work_id) UART输入: "tts.utf-8" UART流式输入: "tts.utf-8.stream" KWS唤醒打断: "kws.xxx"(输入kws单元的work_id)
enoutput	启用UART输出	启用: true 禁用: false
enaudio	启用扬声器播放	启用: true 禁用: true

返回值:

- String:
  - tts_work_id: tts单元work_id

inference

函数原型:

```
int inference(String work_id, String input, uint32_t timeout = 0, String request_id = "tts_i
```

功能说明:

- 输入数据, 开始推理转换, 完成后扬声器将自动播放。

传入参数:

- String work_id:
  - 调用的TTS单元work_id
- String input:
  - 输入文本
- uint32_t timeout:
  - 等待推理超时时间
- String request_id:
  - 会话ID, 当同时存在多个会话的时候用于区分。

返回值:

- int:

- MODULE_LLM_OK / Error Code

ApiMelotts Class

`M5ModuleLLM` 的内部成员 `ApiMelotts melotts` 用于控制 Melotts 单元的初始化和配置。

setup

函数原型:

```
String setup(ApiMelottsSetupConfig_t config = ApiMelottsSetupConfig_t(), String request_id =
             String language = "en_US");
```

功能说明:

- 初始化 Melotts 单元, 开启文本转语音功能。

传入参数:

ApiMelottsSetupConfig_t config:

- Melotts单元初始化配置:
- String request_id:
  - 会话id, 使用默认即可。

```
struct ApiMelottsSetupConfig_t {
    String model              = "melotts_zh-cn";
    String response_format    = "sys.pcm";
    std::vector<String> input = {"tts.utf-8.stream"};
    bool enoutput             = false;
    bool enaudio              = true;
};
```

参数	描述	输入值
model	转换模型	中英文模型: "melotts_zh-cn" 中文模型: "single_speaker_fast"
input	输入	LLM输入: "llm.xxx"(输入llm单元的work_id) UART输入: "melotts.utf-8" UART流式输入: "melotts.utf-8.stream"
enoutput	启用UART输出	启用: true 禁用: false
enaudio	启用扬声器播放	启用: true 禁用: true

返回值:

- String:
  - melotts_work_id: melotts 单元 work_id

inference

函数原型:

```
int inference(String work_id, String input, uint32_t timeout = 0, String request_id = "tts_i
```

功能说明:

- 输入数据, 开始推理转换, 完成后扬声器将自动播放。

传入参数:

- String work_id:
  - 调用的 Melotts 单元work_id
- String input:
  - 输入文本
- uint32_t timeout:
  - 等待推理超时时间
- String request_id:
  - 会话ID, 当同时存在多个会话的时候用于区分。

返回值:

- int:
  - MODULE_LLM_OK / Error Code

ApiYolo Class

`M5ModuleLLM` 的内部成员 `ApiYolo yolo` 用于控制 Yolo 单元的初始化和配置。

setup

函数原型:

```
String setup(ApiYoloSetupConfig_t config = ApiYoloSetupConfig_t(), String request_id = "yolo
```

功能说明:

- 初始化 Yolo 单元, 开启图像检测功能。

传入参数:

ApiYoloSetupConfig_t config:

- Yolo 单元初始化配置:
- String request_id:
  - 会话id, 使用默认即可。

```
struct ApiYoloSetupConfig_t {
    String model               = "yolo11n";
    String response_format     = "yolo.box.stream";
    std::vector<String> input = {"yolo.jpeg.base64"};
    bool enoutput              = true;
};
```

参数	描述	输入值
model	转换模型	检测模型: "yolo11n" 姿态模型: "yolo11n-pose" 手部姿态模型: "yolo11n-hand-pose"
response_format	输出格式	检测输出: "yolo.box.stream" 姿态输出: "yolo.pose.stream"
input	输入	UVC 输入: "camera.xxx"(输入 camera 单元的 work_id) UART流式输入: "yolo.jpeg.base64.stream"
enoutput	启用UART输出	启用: true 禁用: false

返回值:

- String:
  - yolo_work_id: yolo 单元 work_id

ModuleMsg Class

`M5ModuleLLM` 的内部成员 `ModuleMsg msg` 提供了 `responseMsgList` 容器用于用于缓存接收 Module LLM 返回的各种信息。参考以下案例，在主循环中遍历获取返回结果。

```
void loop()
{
    module_llm.update();

    // Handle response msg
    for (auto& msg : module_llm.msg.responseMsgList) {
        // KWS msg
        if (msg.work_id == kws_work_id) {
            Serial.printf(">> Keyword detected\n");
        }

        // ASR msg
        if (msg.work_id == asr_work_id) {
            if (msg.object == "asr.utf-8.stream") {
                // Parse and get asr result
                JsonDocument doc;
                deserializeJson(doc, msg.raw_msg);
                String asr_result = doc["data"]["delta"].as<String>();
                Serial.printf(">> %s\n", asr_result.c_str());
            }
        }
    }
    module_llm.msg.responseMsgList.clear();
}
```

VoiceAssistant Class

`M5ModuleLLM_VoiceAssistant` 用于快速创建 LLM 语音助手实例, 快速实现 KWS(语音唤醒)->ASR(语音转文本)->LLM(大模型推理)-> TTS(文本转语音)。

- 初始化时候只需要将 `M5ModuleLLM` 实例传入构造函数, 并注册对应事件的回调函数即可完成语音助手创建。

```
/*
 * SPDX-FileCopyrightText: 2024 M5Stack Technology CO LTD
 *
 * SPDX-License-Identifier: MIT
 */
#include <Arduino.h>
#include <M5Unified.h>
#include <M5ModuleLLM.h>

M5ModuleLLM module_llm;
M5ModuleLLM_VoiceAssistant voice_assistant(&module_llm);

/* On ASR data callback */
void on_asr_data_input(String data, bool isFinish, int index)
{
    M5.Display.setTextColor(TFT_GREEN, TFT_BLACK);
    M5.Display.printf(">> %s\n", data.c_str());

    /* If ASR data is finish */
    if (isFinish) {
        M5.Display.setTextColor(TFT_YELLOW, TFT_BLACK);
        M5.Display.print(">> ");
    }
};

/* On LLM data callback */
void on_llm_data_input(String data, bool isFinish, int index)
{
    M5.Display.print(data);

    /* If LLM data is finish */
    if (isFinish) {
        M5.Display.print("\n");
    }
};

void setup()
{
    M5.begin();
    M5.Display.setTextSize(2);
    M5.Display.setTextScroll(true);

    /* Init module serial port */
    Serial2.begin(115200, SERIAL_8N1, 16, 17);  // Basic
    // Serial2.begin(115200, SERIAL_8N1, 13, 14);  // Core2
    // Serial2.begin(115200, SERIAL_8N1, 18, 17);  // CoreS3

    /* Init module */
```

```
    module_llm.begin(&Serial2);

    /* Make sure module is connected */
    M5.Display.printf(">> Check ModuleLLM connection..\n");
    while (1) {
        if (module_llm.checkConnection()) {
            break;
        }
    }

    /* Begin voice assistant preset */
    M5.Display.printf(">> Begin voice assistant..\n");
    int ret = voice_assistant.begin("HELLO");
    if (ret != MODULE_LLM_OK) {
        while (1) {
            M5.Display.setTextColor(TFT_RED);
            M5.Display.printf(">> Begin voice assistant failed\n");
        }
    }

    /* Register on ASR data callback function */
    voice_assistant.onAsrDataInput(on_asr_data_input);

    /* Register on LLM data callback function */
    voice_assistant.onLlmDataInput(on_llm_data_input);

    M5.Display.printf(">> Voice assistant ready\n");
}

void loop()
{
    /* Keep voice assistant preset update */
    voice_assistant.update();
}
```

Error Code

```
enum ModuleLLMErrorCode_t {
    MODULE_LLM_OK                                = 0,
    MODULE_LLM_RESET_WARN                        = -1,
    MODULE_LLM_JSON_FORMAT_ERROR                 = -2,
    MODULE_LLM_ACTION_MATCH_FAILED               = -3,
    MODULE_LLM_INFERENCE_DATA_PUSH_FAILED        = -4,
    MODULE_LLM_MODEL_LOADING_FAILED              = -5,
    MODULE_LLM_UNIT_NOT_EXIST                    = -6,
    MODULE_LLM_UNKNOWN_OPERATION                 = -7,
    MODULE_LLM_UNIT_RESOURCE_ALLOCATION_FAILED = -8,
    MODULE_LLM_UNIT_CALL_FAILED                  = -9,
    MODULE_LLM_MODEL_INIT_FAILED                 = -10,
    MODULE_LLM_MODEL_RUN_FAILED                  = -11,
    MODULE_LLM_MODULE_NOT_INITIALISED            = -12,
    MODULE_LLM_MODULE_ALREADY_WORKING            = -13,
    MODULE_LLM_MODULE_NOT_WORKING                = -14,
    MODULE_LLM_NO_UPDATEABLE_MODULES             = -15,
    MODULE_LLM_NO_MODULES_AVAILABLE_FOR_UPDATE = -16,
    MODULE_LLM_FILE_OPEN_FAILED                  = -17,
    MODULE_LLM_WAIT_RESPONSE_TIMEOUT             = -97,
    MODULE_LLM_RESPONSE_PARSE_FAILED             = -98,
    MODULE_LLM_ERROR_NONE                        = -99,
};
```