在 MA35D1 Buildroot 上实现 ChatGPT

NuMicro® 64 位系列微控制器范例代码介绍

文件信息

应用简述	本程序在 Nuvoton Buildroot 平台上，利用 OpenAI API 来实现 ChatGPT，包括聊天功能和图像生成功能等。
BSP 版本	MA35D1 Series Buildroot 2024
开发平台	NuMaker-IoT-MA35D16F90 V2.4

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. 概述

此范例程序利用 OpenAI API 与 OpenAI 服务器进行通讯，进而提供 ChatGPT 功能，包括聊天和图像生成等。它提供了一个简单而方便的方式，让使用者可以通过命令行界面与 OpenAI 的众多 AI 模型进行互动。

这个程序的目的是在嵌入式平台上实现强大的 AI 能力，为使用者提供自然流畅的对话体验和图像生成功能。它可以应用于各种应用场景，如智能家居、机器人等，为设备带来更智能的互动体验。

此范例程序是基于 Nuvoton Buildroot 2024 的环境进行开发，需先建立 Nuvoton Buildroot 2024 开发环境 (https://github.com/OpenNuvoton/buildroot_2024)，才能够执行此范例程序。

虽然在这篇文章中，是以 NuMaker-IoT-MA35D16F90 V2.4 开发板来举例。但是这篇文章的内容适用于所有支持 Nuvoton Buildroot 2024 的开发板，包括 NUC970、NUC980、N9H30、MA35D0、MA35D1 和 MA35H0 等系列。

1.1 原理

OpenAI API 是一个开放的应用程序接口，用于访问 OpenAI 的语言模型。而 ChatGPT 是 OpenAI API 中的一个特定模型，专门用于聊天和对话式应用。

ChatGPT 的特点是可以进行对话式的互动，它能够理解上下文并生成相应的回应。这是通过将用户的输入和之前的对话作为模型的输入，并使用生成式的方法来生成回应。ChatGPT 还具有一定的自主性，可以在生成回应时表现出一定的创造性。

OpenAI API 支持多种不同方式的函数库，用以与 OpenAI 服务器进行通讯，其中包括 curl、python、node.js 和 csharp。本程序是使用 curl 方式来与 OpenAI 服务器进行通讯。当用户在命令行界面上输入指令时，程序会使用 curl 库来建立 HTTP 请求，将指令作为请求的内容发送到 OpenAI 服务器的 API 端点。服务器在接收到请求后，使用 ChatGPT 或其他用户指定的模型进行处理，并生成相应的回应或图像。然后，服务器将生成的结果作为 HTTP 响应返回给程序，程序再将响应解析并显示在命令行界面上供用户查看。这样，用户就可以通过这个程序使用 curl 方式与 OpenAI 服务器进行通讯，实现 ChatGPT 功能。

1.2 环境设定

此范例程序主要是以 bash 的语法所撰写，在程序中会使用到许多 Linux 工具。因此在执行前，必须要确保这些工具是可以正常运行的。因为在 Buildroot 中，可以很容易地支持这些所需的工具。所以在本篇文章中，就以 Buildroot 为开发环境。

MA35D1 环境架设，生成映像文件可参考新唐 MA35D1 Buildroot Project User Manual。

- **Buildroot**

于 Buildroot 的 menuconfig 中启用此范例程序所需的套件，请按照以下步骤进行：

1. 通过执行 make menuconfig 命令来打开 Buildroot 的配置菜单。在套件选择菜单中，可以选择相应的选项并按下空格键来启用套件。也可以直接使用「/」键，用套件的名称来搜索套件。

2. 范例程序主要是以 bash 的语法所编写，而且在输入的文字处理上会使用到 readline 和 Unicode 等功能。所以需要启用相关套件。

   - Target packages ---> [*] Show packages that are also provided by busybox
   - Target packages ---> Shell and utilities ---> [*] bash
   - Target packages ---> Libraries ---> Text and terminal handling ---> [*] ncurses
   - Target packages ---> Libraries ---> Text and terminal handling ---> [*] enable wide char support

3. 范例程序是使用 curl 的方式来与 OpenAI 服务器进行沟通。双方在进行沟通时，会使用 JSON 格式。而且必须确保系统的时间是准确的，否则 curl 会出现异常。所以需要启用 curl、jq 和 ntp 等相关套件。

   - Target packages ---> Libraries ---> Networking ---> [*] libcurl
   - Target packages ---> Libraries ---> Networking ---> [*] curl binary
   - Target packages ---> Libraries ---> Networking ---> SSL/TLS library to use (OpenSSL) --->
   - Target packages ---> Libraries ---> Crypto ---> [*] CA Certificates
   - Target packages ---> Development tools ---> [*] jq
   - Target packages ---> Networking applications ---> [*] ntp

4. 对于输出的文字，我们会使用 gawk 来对其进行排版处理。所以需要启用 gawk 套件。

   - Target packages ---> Development tools ---> [*] gawk

   需要注意的是，若要处理 Unicode，则 C 函数库环境需要支持 UTF-8。在 Buildroot 默认是使用 glibc 函数库，你可以选择改用 musl 函数库。因为 musl 函数库默认就支持 UTF-8，所以直接就能够支持处理 Unicode。

   - Toolchain ---> C library (musl) --->

   若要维持原先的设置使用 glibc 函数库，则需要为 glibc 函数库建立 locale 环境。以下是以 C.UTF-8 为例。

   - System configuration ---> [*] Purge unwanted locales
   - System configuration ---> (C.UTF-8) Locales to keep
   - System configuration ---> (C.UTF-8) Generate locale data

5. 除了范例程序之外，我们还要将相关的系统配置文件放到设备之中。所以在配置菜单内，我们必须指定一个根文件系统覆盖目录。而之后我们所新增或修改的文件，就可以直接放到这个目录中。

   - System configuration ---> (board/nuvoton/ma35d1/rootfs_overlay) Root filesystem overlay directories

6. 启用所有需要的套件后，退出 menuconfig 并保存配置。

7. 将 rootfs_overlay 整个目录拷贝到 board/nuvoton/ma35d1/目录下，以符合前面定义的根文件系统覆盖目录。这样能够确保范例程序和系统配置文件，都放置在 Buildroot 系统的正确位置。

8. 通过 make 命令使用更新的配置来构建 Buildroot 系统。

编译完成后，使用 NuWriter 将映像文件烧录到 NuMaker-IoT-MA35D16F90 开发板。

1.3 执行结果

本范例程序的 Buildroot 映像文件执行编译结果，如图 1-1 所示。在编译完成后，可以在 *output/images/* 目录下找到生成的映像文件。

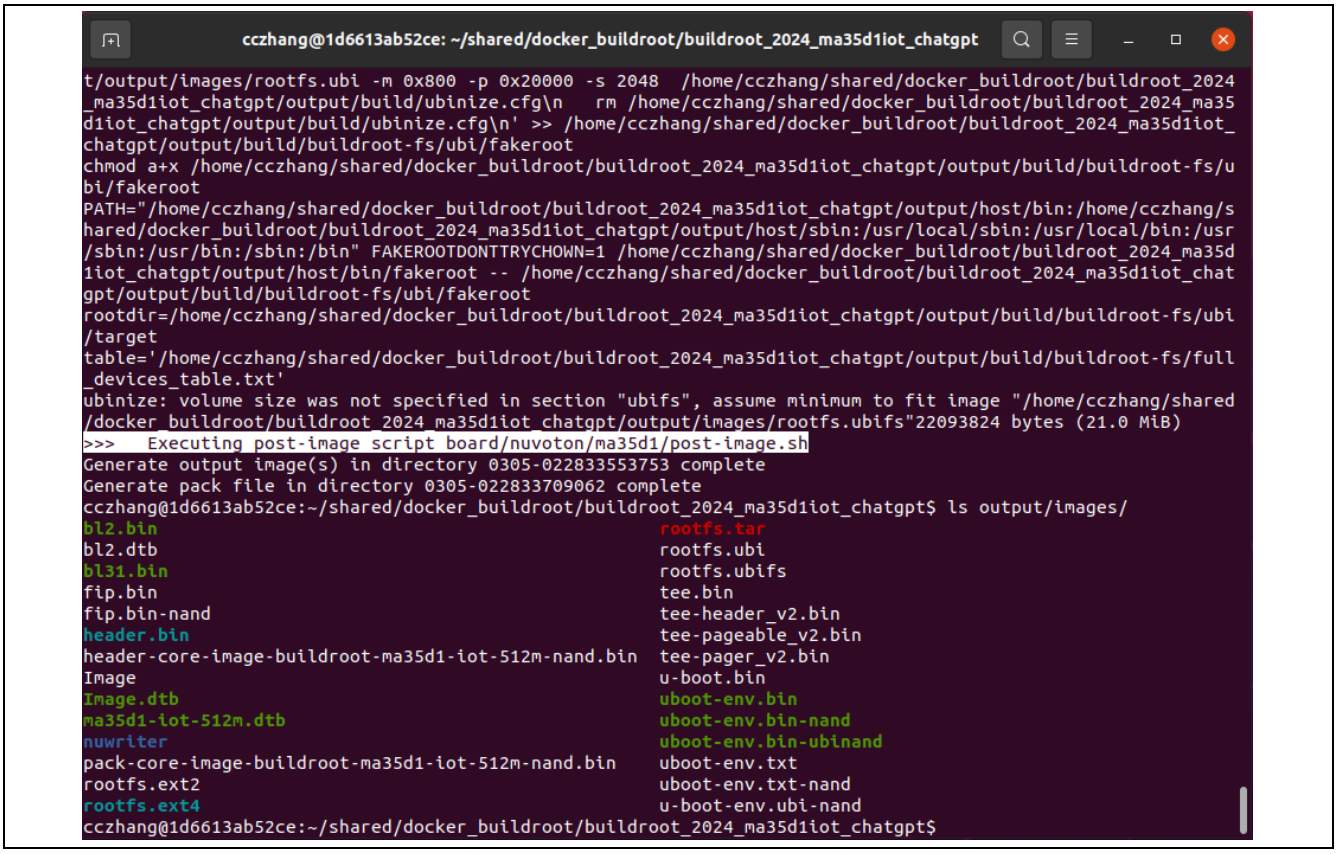

图 1-1 Buildroot 执行编译结果

本范例程序的 Buildroot 映像文件执行烧录结果，如图 1-2 所示。

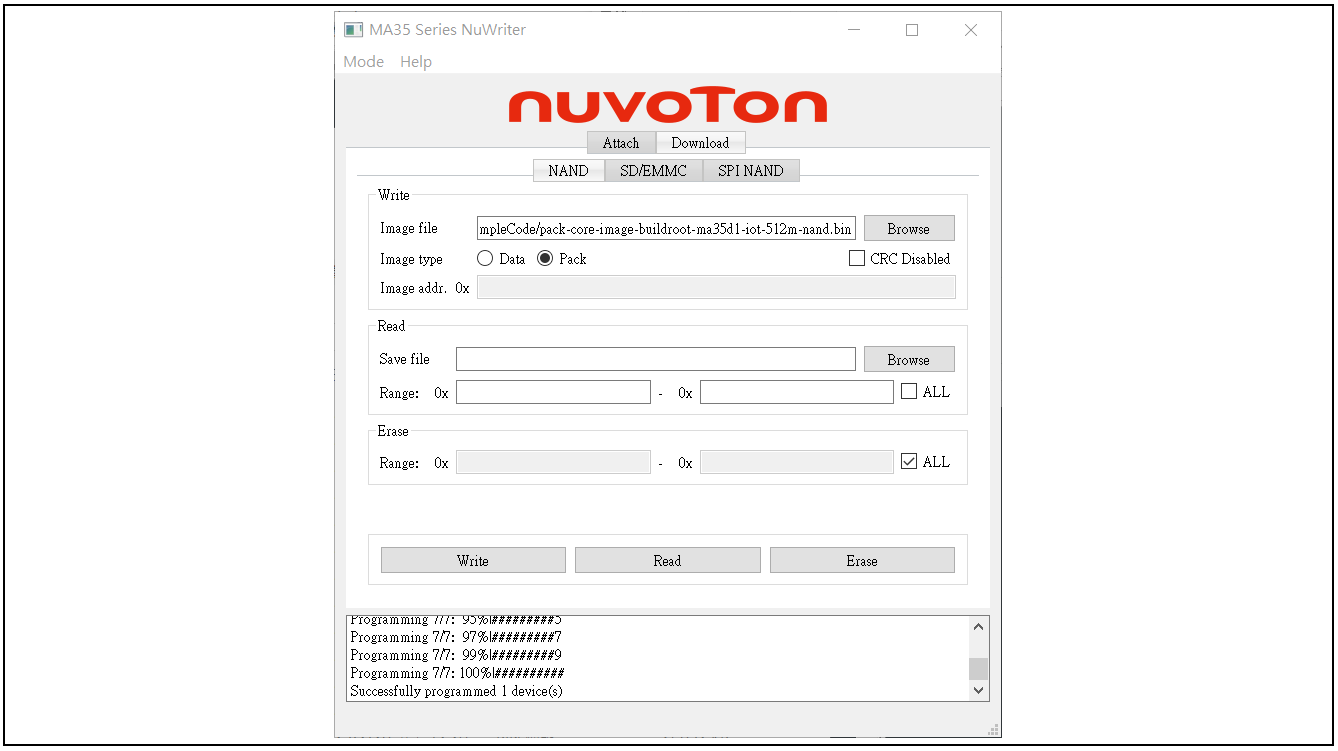

图 1-2 使用 NuWriter 烧录 Buildroot 映像文件

本范例程序在终端软件下的聊天功能执行结果，以 Putty 为例，如图 1-3 和图 1-4 所示。在聊天的过程中，可以与 ChatGPT 进行互动。

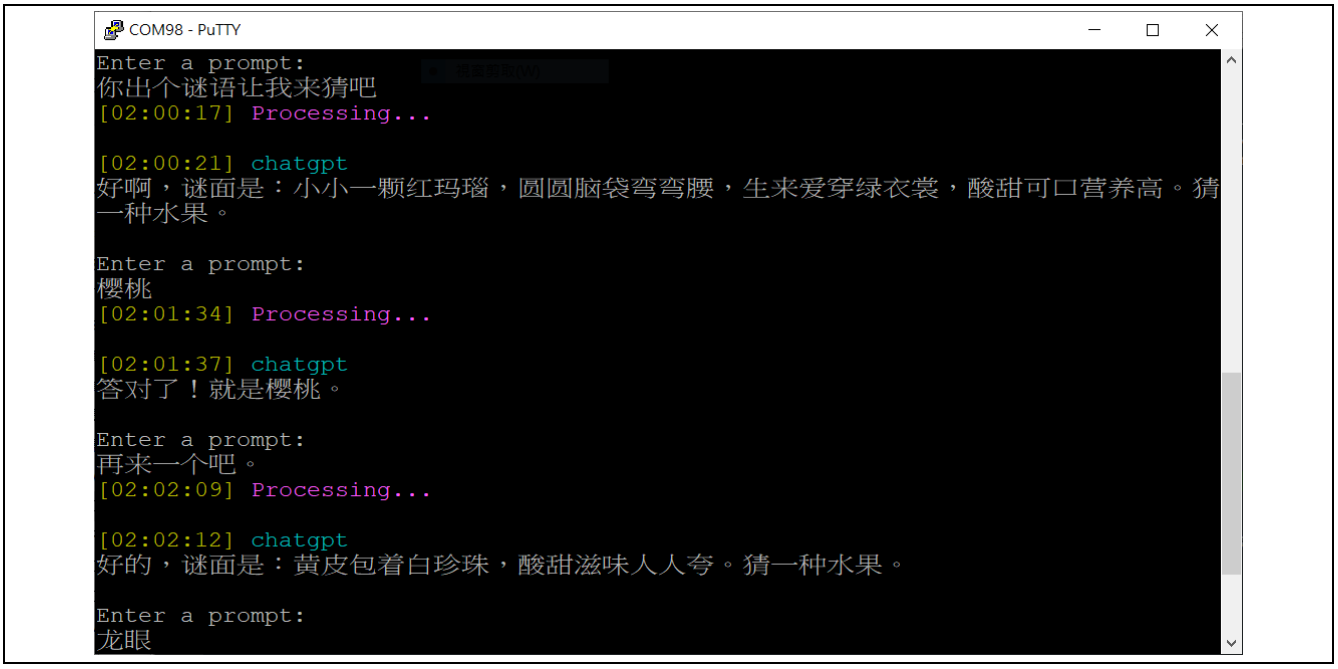

图 1-3 ChatGPT 聊天功能示例 1

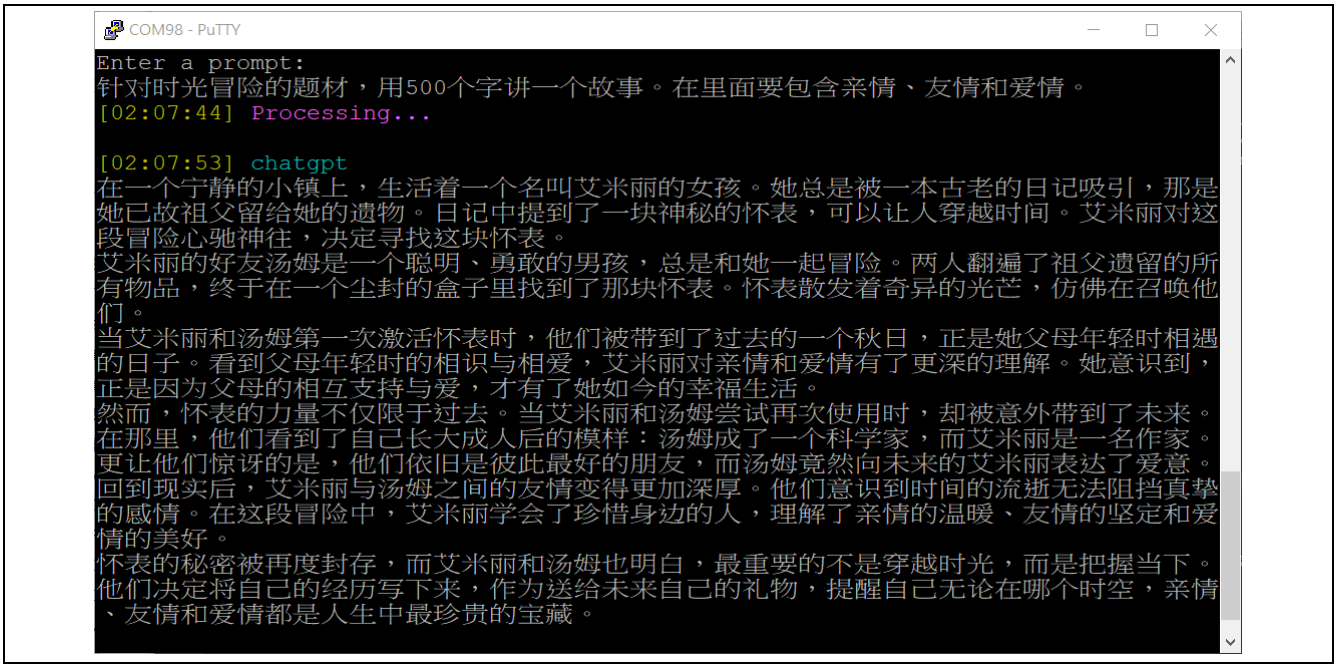

图 1-4 ChatGPT 聊天功能示例 2

本范例程序的图像生成功能执行结果，如图 1-5 所示。图像生成的指令为「image:」，在其之后输入图像生成的提示，ChatGPT 就会生成对应的图像。在生成的过程结束时，在窗口上会显示生成图像的 URL。之后也会将此图像下载至本地，图文件的名称为 *image.png*。

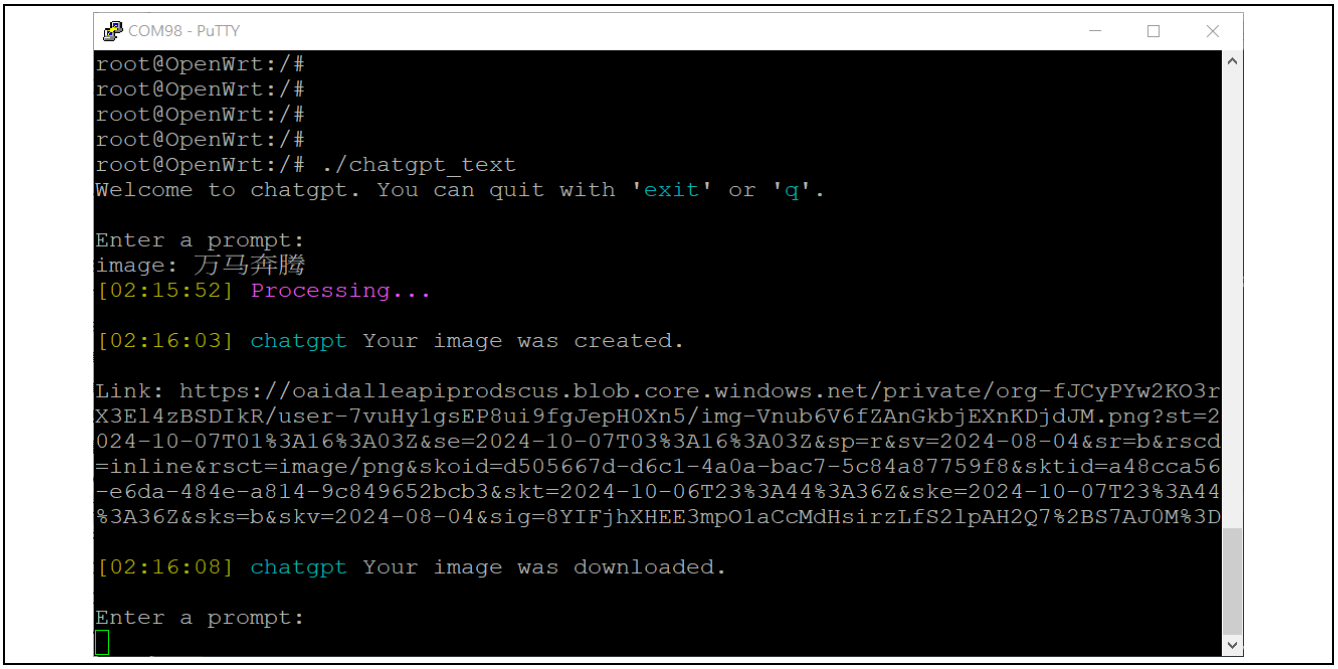

图 1-5 ChatGPT 图像生成功能示例

在图像生成时，提示可以是一个简单的词语。比如：

万马奔腾

其结果如图 1-6 所示。

图 1-6 ChatGPT 生成图像 1

提示也可以是一整段较复杂的文段，比如：

在这场激烈的战争中，精灵、矮人和人类联合起来，面对邪恶的魔族和黑暗的半兽人。彼此的勇士奋勇作战，剑与斧相交，箭矢如雨。精灵的弓箭手在树梢上射下死亡的雨幕，矮人的战斧劈开敌人的盾牌，人类的勇士冲锋陷阵。火焰和魔法交织，为这片战场增添了惊心动魄的景象。种族间的血战持续着，但他们的团结和勇气使他们能够抵抗黑暗势力，保卫着自由和和平的希望。

其结果如图 1-7 所示。

图 1-7 ChatGPT 生成图像 2

提示也可以是一段对图像生成的描述，其中包含着许多不同的条件要求，比如：

以小白兔和狐狸在打架为主题，画一个 3D 风格的四格漫画。

其结果如图 1-8 所示。

图 1-8 ChatGPT 生成图像 3

2. 代码介绍

因为在生成式AI的文本中，常常会使用到一些特殊字符。而这些特殊字符，在使用OpenAI API时会出现格式解析的错误。因此必须要对其进行事先处理，以确保不会影响到程序运行。

```
# escapes special characters to ensure the string can be safely used
# in a JSON-compatible format
# $1 should be the message(s) to do escape
escape() {
    echo "$1" | jq -Rrs 'tojson[1:-1]'
}
```

当使用者输入消息给 ChatGPT 后，在调用 OpenAI API 之前，必须要将其先打包成 ChatGPT 模型的标准格式。另外 ChatGPT 的互动回应也会跟之前的对话相关，所以这段新的使用者消息会加在对话消息列表的最后，再一起作为模型的输入文本。

```
# build user chat message function for /chat/completions (gpt models)
# builds chat message before request,
# $1 should be the escaped request prompt,
# it extends $chat_message
build_user_chat_message() {
    local escaped_request_prompt="$1"
    if [ -z "$chat_message" ]; then
        chat_message="{\"role\": \"user\", \"content\": \"$escaped_request_prompt\"}"
    else
        chat_message="$chat_message, {\"role\": \"user\", \"content\":
\"$escaped_request_prompt\"}"
    fi
}
```

ChatGPT 的互动回应跟之前的对话相关，这里面除了使用者的输入消息之外，也包含了 ChatGPT 之前的回应。所以当 ChatGPT 有了回应之后，这段回应也要加在对话消息列表的最后。

另外在 OpenAI API 中，token 是指文本的最小单位，而 API 的价格和限制是基于 token 数量计算的。在程序中会设定一个最大 token 数量的限制，如果输入文本的 token 数量超过了这个限制，在这里也会进行缩减。其规则是从最旧的数据开始删除，直到符合限制值为止。这个限制值预设是 4096，使用者可以根据自己的需求来进行调整。

```
# adds the assistant response to the message in (chatml) format
# for /chat/completions (gpt models)
# keeps messages length under max token limit,
# $1 should be the escaped response data
# it extends and potentially shrinks $chat_message
add_assistant_response_to_chat_message() {
    local escaped_response_data="$1"
```

```
    # add response to chat context as answer
    chat_message="$chat_message, {\"role\": \"assistant\", \"content\":
\"$escaped_response_data\"}"

    # check prompt length, 1 word =~ 1.3 tokens
    # reserving 100 tokens for next user prompt
    while (($(echo "$chat_message" | wc -c) * 13 / 10 > (MAX_TOKENS - 100))); do
        # transform to json array to parse with jq
        local chat_message_json="[ $chat_message ]"
        # remove first/oldest QnA from prompt
        local chat_message_removed=$(echo "$chat_message_json" | jq -c '.[2:]')
        # remove the json array format
        chat_message=${chat_message_removed:1:-1}
    done
}
```

下列为聊天功能的代码，其调用 OpenAI API 的 curl 函数库，向 OpenAI 服务器请求使用 ChatGPT 模型并给定包含对话的消息列表，之后模型将会传回回应。在 OpenAI API 中，参数 model 为 ChatGPT 模型，默认为 gpt-4o。参数 messages 是迄今为止的对话消息列表，也就是此函数的输入参数再经由一些处理，以确保符合 OpenAI API 的参数格式。参数 max_completion_tokens 指定了生成文本的最大 token 数量，用于控制生成结果的长度，其默认值为 4096。参数 temperature 的作用，是控制生成文本的随机性和多样性。其值的可设定范围介于 0 到 2 之间，默认值为 0.7。当 temperature 设置为较高的值时，生成的文本会更加随机和多样化。这意味着模型有可能会选择概率较低的选项，并产生一些更加出人意料的回应。相反，当 temperature 设置为较低的值时，生成的文本会更加重视确定性和一致性。若要进一步了解 OpenAI API 内的所有参数，可以参考 OpenAI API reference。

```
# request to OpenAPI API chat completion endpoint function
# $1 should be the message(s) formatted with role and content
request_to_chat() {
    local message="$1"
    escaped_system_prompt=$(escape "$SYSTEM_PROMPT")

    curl https://api.openai.com/v1/chat/completions \
     -sS \
     -H 'Content-Type: application/json' \
     -H "Authorization: Bearer $OPENAI_KEY" \
     -d '{
         "model": "'"$MODEL"'",
         "messages": [
             {"role": "system", "content": "'"$escaped_system_prompt"'"},
             '"$message"'
         ],
         "max_completion_tokens": '$MAX_COMPLETION_TOKENS',
         "temperature": '$TEMPERATURE'
     }'
}
```

因为 ChatGPT 模型的回应文本，可能会包含多国语言。这边针对一些常见的语言，例如英文、中文、日文等，在显示回应文本到终端窗口时会进行换行的处理，以利于阅读。

```
# function to wrap text considering different languages
# $1 should be the response body
text_display() {
echo "$1" | gawk -v max_width=$MAX_WIDTH '
{
    # initialize variables
    line = ""  # current line content
    len = 0  # current line length

    # split the line into words by spaces
    n = split($0, words, " ")

    # loop through each word
    for (i = 1; i <= n; i++) {
     word = words[i]  # extract the current word
     word_len = 0  # length of the current word (consider ASCII and non-ASCII characters)

     # calculate the length of the word (English words and non-ASCII characters)
     for (j = 1; j <= length(word); j++) {
          char = substr(word, j, 1)

          # check if the character is an ASCII character (1 space)
          if (char ~ /[ -~]/) {
               word_len += 1
          } else {
               # non-ASCII characters (Chinese, Japanese) take 2 spaces
               word_len += 2
          }
     }
    …
    …
}
```

如果在使用 OpenAI API 时，OpenAI 服务器有报错的话，这边会将错误的相关信息显示在终端窗口上。

```
# error handling function
# $1 should be the response body
handle_error() {
    if echo "$1" | jq -e '.error' >/dev/null; then
     echo -e "Your request to Open AI API failed: \033[0;31m$(echo "$1" | jq -r
'.error.type')\033[0m"
     echo "$1" | jq -r '.error.message'
     exit 1
    fi
}
```

下列为图像生成功能的代码，其调用 OpenAI API 的 curl 函数库，向 OpenAI 服务器请求使用 dall-e-3 模型并给定提示，之后模型将会生成新图像。在 OpenAI API 中，参数 model 为图像生

成模型，默认为 dall-e-3。参数 prompt 为图像生成的提示，此函数的输入参数为图像生成的指令「image:」再加上图像生成的提示。在将图像生成的指令移除后，仅保留图像生成的提示来作为此项参数设置。参数 n 为生成图像的数量，默认值为 1。参数 size 为生成图像的尺寸，默认值为 1024x1024。

```
# request to OpenAI API image generations endpoint function
# $1 should be the prompt
request_to_image() {
    local prompt="$1"
    image_response=$(curl https://api.openai.com/v1/images/generations \
    -sS \
    -H 'Content-Type: application/json' \
    -H "Authorization: Bearer $OPENAI_KEY" \
    -d '{
        "model": "dall-e-3",
        "prompt": "'"${prompt#*image:}"'",
        "n": 1,
        "size": "'"$SIZE"'"
    }')
}
```

除了 dall-e-3 模型之外，另外还可以选择旧版的 dall-e-2 模型，以下表格为上述参数在两者间的差别。使用者可以根据自己的需求，来选择图像生成的模型。

OpenAI API 参数	dall-e-2	dall-e-3
prompt	最大为 1000 字符	最大为 4000 字符
n	1~10	1
size	256x256 512x512 1024x1024	1024x1024 1792x1024 1024x1792

表 2-1 dall-e-2 与 dall-e-3 模型之参数比较

下列是一些 OpenAI 模型的参数默认值，设置用户可以进行修改以符合其需求。其中只有最后一项 SIZE 是跟图像生成功能有关，其余的参数皆是跟聊天功能有关。需要注意到的是，参数 MAX_COMPLETION_TOKENS 只有在 GPT 4 以后的模型，才会有支持。

```
# set defaults
MODEL=${MODEL:-gpt-4o}
MAX_TOKENS=${MAX_TOKENS:-4096}
MAX_COMPLETION_TOKENS=${MAX_COMPLETION_TOKENS:-4096}
TEMPERATURE=${TEMPERATURE:-0.7}
MAX_WIDTH=${MAX_WIDTH:-80}
SIZE=${SIZE:-1024x1024}
```

在程序最下方的 while 循环，则是整个聊天功能和图像生成功能的主要代码。在提示可输入时，若用户输入「exit」或「q」，则结束程序运行。若用户输入以「image:」开头的字符串，则进行图像生成功能。除此之外，皆进行聊天功能。在进行图像生成功能时，会将生成图像结果的远程 URL 显示在窗口中，也会将图像下载至本地目录。

```
while $running; do

    …
    …

    if [[ $prompt =~ ^(exit|q)$ ]]; then
         running=false
    elif [[ "$prompt" =~ ^image: ]]; then
         request_to_image "$prompt"
         handle_error "$image_response"
         image_url=$(echo "$image_response" | jq -r '.data[0].url')
         if [ "$time_info" = true ]; then
             echo -ne "\n\033[33m["`date "+%H:%M:%S"`"] \033[0m"
         fi
         echo -e "${CHATGPT_CYAN_LABEL}Your image was created. \n\nLink: ${image_url}\n"

         if [ -f image.png ]; then
             rm image.png
         fi
         curl -sS $image_url -o image.png
         if [ "$time_info" = true ]; then
             echo -ne "\033[33m["`date "+%H:%M:%S"`"] \033[0m"
         fi
         echo -e "${CHATGPT_CYAN_LABEL}Your image was downloaded."
    else
         # escape quotation marks, new lines, backslashes...
         request_prompt=$(escape "$prompt")

         build_user_chat_message "$request_prompt"
         response=$(request_to_chat "$chat_message")
         handle_error "$response"
         response_data=$(echo "$response" | jq -r '.choices[].message.content')

         if [ "$time_info" = true ]; then
             echo -ne "\n\033[33m["`date "+%H:%M:%S"`"] \033[0m"
         fi
         echo -e "${CHATGPT_CYAN_LABEL}"
         text_display "${response_data}"
         echo -e "${CHATGPT_CYAN_LABEL}${response_data}" > response.txt
         add_assistant_response_to_chat_message "$(escape "$response_data")"

         timestamp=$(date +"%Y-%m-%d %H:%M")
         echo -e "$timestamp $prompt \n$response_data \n" >>~/.chatgpt_history
    fi
done
```

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
  - MA35D1 Series Buildroot 2024

3.2 硬件需求

- 电路组件
  - NuMaker-IoT-MA35D16F90 V2.4

4. 目录信息

目录结构如下图所示。

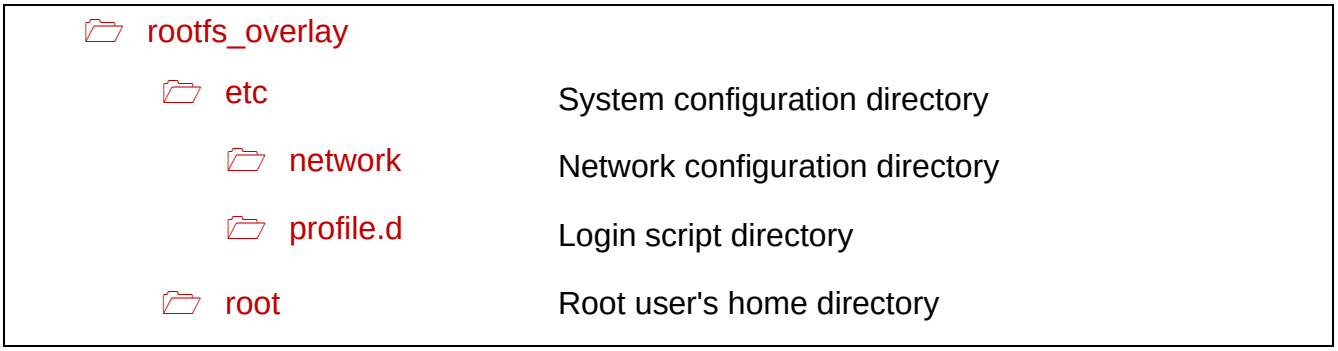

图 4-1 目录信息

5. 范例程序执行

1. 用终端软件连接上 NuMaker-IoT-MA35D16F90 开发板的 VCOM。

2. 在 Buildroot 开机后，需要确保设备是可以正常连接上网络，并且系统时间是准确的，这样才能够正常执行 ChatGPT 范例程序。可以利用 ping 和 date 命令来进行确认。

```
# ping -c 1 api.openai.com
PING api.openai.com (172.66.0.243): 56 data bytes
64 bytes from 172.66.0.243: seq=0 ttl=56 time=28.304 ms

--- api.openai.com ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 28.304/28.304/28.304 ms
# date
Thu Mar  6 05:43:21 UTC 2025
```

3. 在系统环境中，设置自行申请的 OpenAI API key。

```
# export OPENAI_KEY=your_key_here
```

4. 执行 ChatGPT 范例程序。关于这个部分，可以参考下列描述。

   - 在执行范例程序之前，可以使用「-h」参数来查看帮助信息。

```
# ./chatgpt_text -h
A simple, lightweight shell script to use OpenAI's Language Models and DALL-E from the
terminal.

Commands:
  image: - To generate an image and store it named "image.png" in local, you also can open
the image in your browser.

Options:
  -p, --prompt              Provide prompt instead of starting chat

  -l, --list                List available openAI models

  -m, --model               Model to use

  -t, --temperature         Temperature

  -s, --size                Image size. (The sizes that are accepted by the OpenAI API is
                            dall-e-3 model: 1024x1024, 1792x1024 or 1024x1792,
                            dall-e-2 model: 256x256, 512x512 or 1024x1024)
```

   - 可以直接执行范例程序进入交谈模式，如同前述章节之执行结果。在对话中输入一般文字时，会调用聊天功能。但是如果在文字前面加上「image:」时，就会调用图像生成功能。

```
# ./chatgpt_text
```

- 在执行范例程序时，也可以直接指定参数的设置值，用以取代在代码中之预设值。以下为改成使用聊天模型 gpt-3.5-turbo 的范例。要注意到的是，许多的聊天模型其信息格式与 gpt-4o 并不相容。若要使用这些模型，必须要先分析其信息格式，并且修改范例程序的相关内容。

```
# ./chatgpt_text -m gpt-3.5-turbo
```

- 在执行时使用「-l」参数，可以列出官方目前所有支持的模型清单。

```
# ./chatgpt_text -l
```

- 在执行时使用「-p」参数，可以直接指定聊天提示并且得到回应，不用进入到交谈模式。要注意到的是，此种方法只适合一次性的独立提问，所以其回应不会与之前的文本有所互动。

```
# ./chatgpt_text -p "地球上总共有多少海水?"

[07:24:55]
地球上的海水总量约为13.86亿立方千米，占地球水资源的绝大部分。
#
```

6. 修订纪录

Date	Revision	Description
2025.04.25	1.00	初始发布。

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.