Implementing ChatGPT in MA35D1 Buildroot

Example Code Introduction for 64-bit NuMicro® Family

Document Information

Application	This example program is implemented on the Nuvoton Buildroot platform using the OpenAI API to enable ChatGPT, including chat functionality and image generation capabilities.
BSP Version	MA35D1 Series Buildroot 2024
Hardware	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. Overview

This example program utilizes the OpenAI API to communicate with the OpenAI server and provide ChatGPT functionality, including chat and image generation capabilities. It offers a simple and convenient way for users to interact with various AI models from OpenAI through a command-line interface.

The purpose of this program is to implement powerful AI capabilities on embedded platforms, providing users with a natural and fluent conversational experience and image generation functionality. It can be applied in various application scenarios such as smart homes, robots, etc., to bring a more intelligent interactive experience to devices.

This example program is developed based on the Nuvoton Buildroot 2024 environment and requires the establishment of the Nuvoton Buildroot 2024 development environment (https://github.com/OpenNuvoton/buildroot_2024) to run.

Although this article uses the NuMaker-IoT-MA35D16F90 evaluation board as an example, the content of this article is applicable to all evaluation boards that support Nuvoton Buildroot 2024, including NUC970, NUC980, N9H30, MA35D0, MA35D1, and MA35H0 series.

1.1 Principle

The OpenAI API is an open application programming interface used to access OpenAI's language models. ChatGPT is a specific model within the OpenAI API that is designed for chat and conversational applications.

ChatGPT is characterized by its ability to engage in interactive conversations. It can understand context and generate appropriate responses. This is achieved by providing user inputs and previous dialogue as input to the model and using a generative approach to generate responses. ChatGPT also exhibits a certain level of autonomy and can demonstrate creativity in generating responses.

The OpenAI API supports various libraries for communication with the OpenAI server, including curl, python, node.js, and csharp. This program uses the curl library to communicate with the OpenAI server. When a user enters a command in the command-line interface, the program uses the curl library to create an HTTP request, sending the command as the request content to the API endpoint of the OpenAI server. Upon receiving the request, the server processes it using ChatGPT or another model specified by the user and generates the corresponding response or image. The server then returns the generated result as an HTTP response to the program, which parses and displays the response on the command-line interface for the user to view. In this way, users can communicate with the OpenAI server using the curl method and utilize the ChatGPT functionality through this program.

1.2 Environment Setup

This example program is primarily written in bash syntax and utilizes various Linux tools. Therefore, before running the program, it is necessary to ensure that these tools are available and functioning properly. Buildroot makes it easy to support these required tools. Hence, in this article, Buildroot is used as the development environment.

To set up the MA35D1 environment and generate image files, you can refer to the *MA35D1 Buildroot Project User Manual.*

- **Buildroot**

In Buildroot, enable the packages required by this example program by following these steps in the menuconfig:

1. Open the configuration menu of Buildroot by executing the command "make menuconfig". In the package selection menu, choose the appropriate options and click the spacebar to enable each package. You can also use the "/" key to search for packages by their names.
2. The example program is primarily written in bash syntax and utilizes features like readline and Unicode for text input processing. Therefore, it is necessary to enable the relevant packages.
   - 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. The example program uses curl to communicate with the OpenAI server. Communication between both parties is conducted in JSON format. It is also essential to ensure that the system time is accurate; otherwise, curl may encounter issues. Therefore, it is necessary to enable relevant packages such as curl, jq, and ntp.
   - 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. For the output text, gawk is used for formatting and processing. Therefore, it is necessary to enable the gawk package.
   - Target packages ---> Development tools ---> [*] gawk

It is important to note that if you want to handle Unicode, the C library environment needs to support UTF-8. By default, Buildroot uses the glibc library, but you can choose to switch to the musl library. Since the musl library has built-in support for UTF-8, it can directly handle Unicode processing.

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

If you want to maintain the original setting using the glibc library, you need to establish a locale environment for the glibc library. The following is an example using C.UTF-8.

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

5. In addition to the example program, it is also necessary to place the relevant system configuration files onto the device. Therefore, in the configuration menu, users must specify a root filesystem overlay directory. Any files added or modified later can be directly placed in this directory.

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

6. After enabling all the required packages, exit the menuconfig and save the configuration.

7. Copy the entire *rootfs_overlay* directory to the *board/nuvoton/ma35d1/* directory to match the previously defined root filesystem overlay directory. This ensures that the example program and system configuration files are placed in the correct location within the Buildroot system.

8. Use the "make" command to build the Buildroot system with the updated configuration.

Once the compilation is complete, use NuWriter to program the image files to the NuMaker-IoT-MA35D16F90 evaluation board.

1.3 Demo Result

The execution result of compiling the Buildroot image files for this example program are shown in Figure 1-1. After the compilation is complete, you can find the generated image files in the *output/images/* directory.

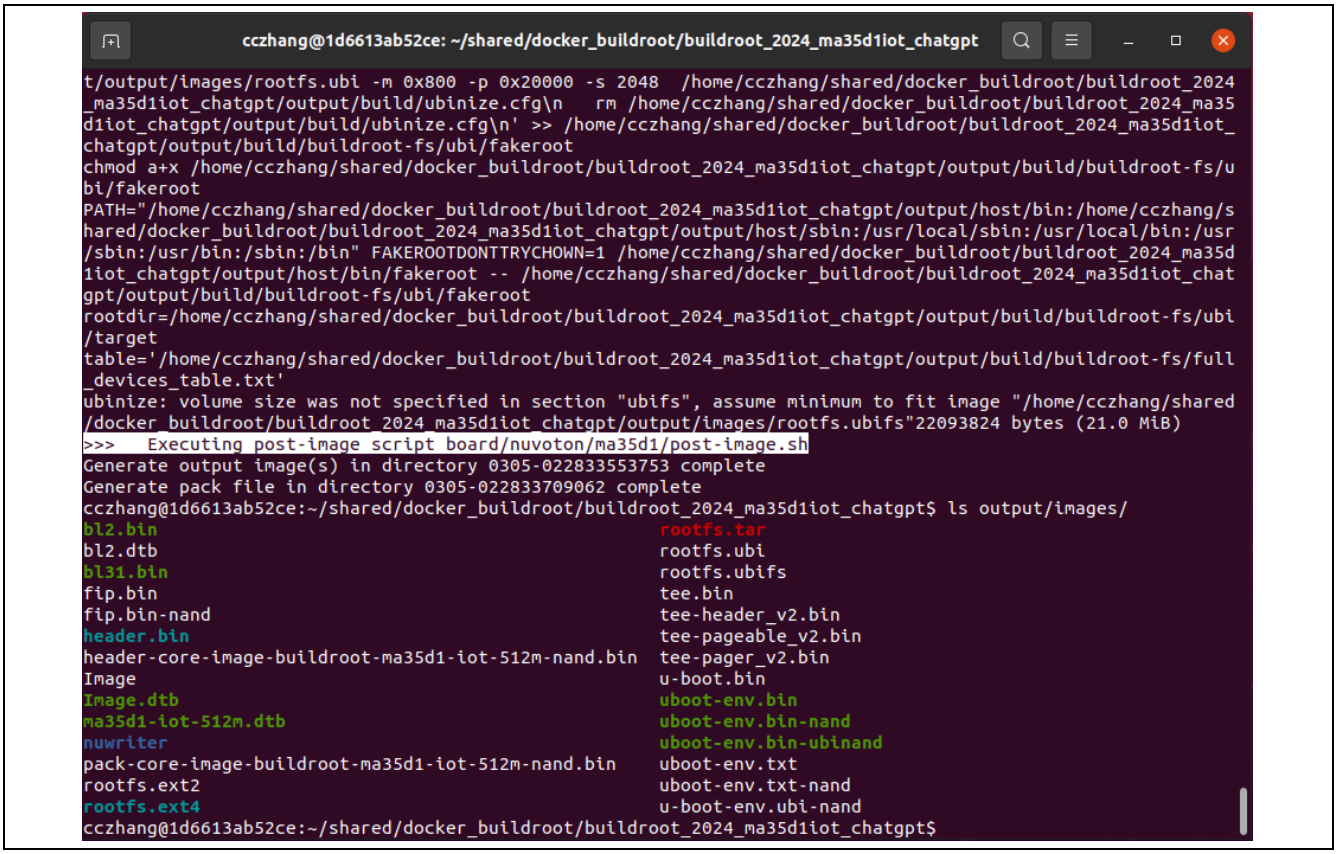

Figure 1-1 Compilation Result of Compiling Buildroot Image Files

The execution result of programming the Buildroot image files for this example program are shown in Figure 1-2.

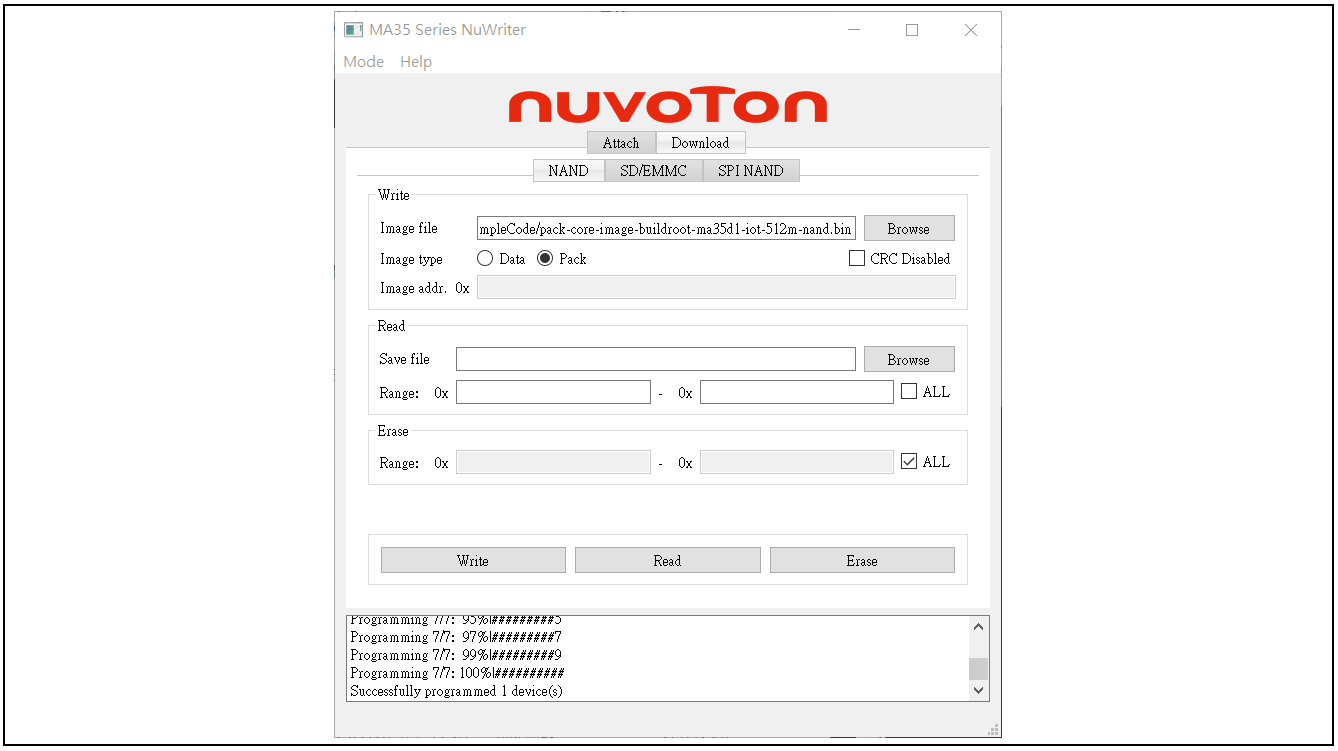

Figure 1-2 Use NuWriter to Program Buildroot Image Files

The execution results of the chat functionality of this example program in the terminal software, using Putty as an example, are shown in Figure 1-3 and Figure 1-4. During the chat process, you can interact with ChatGPT.

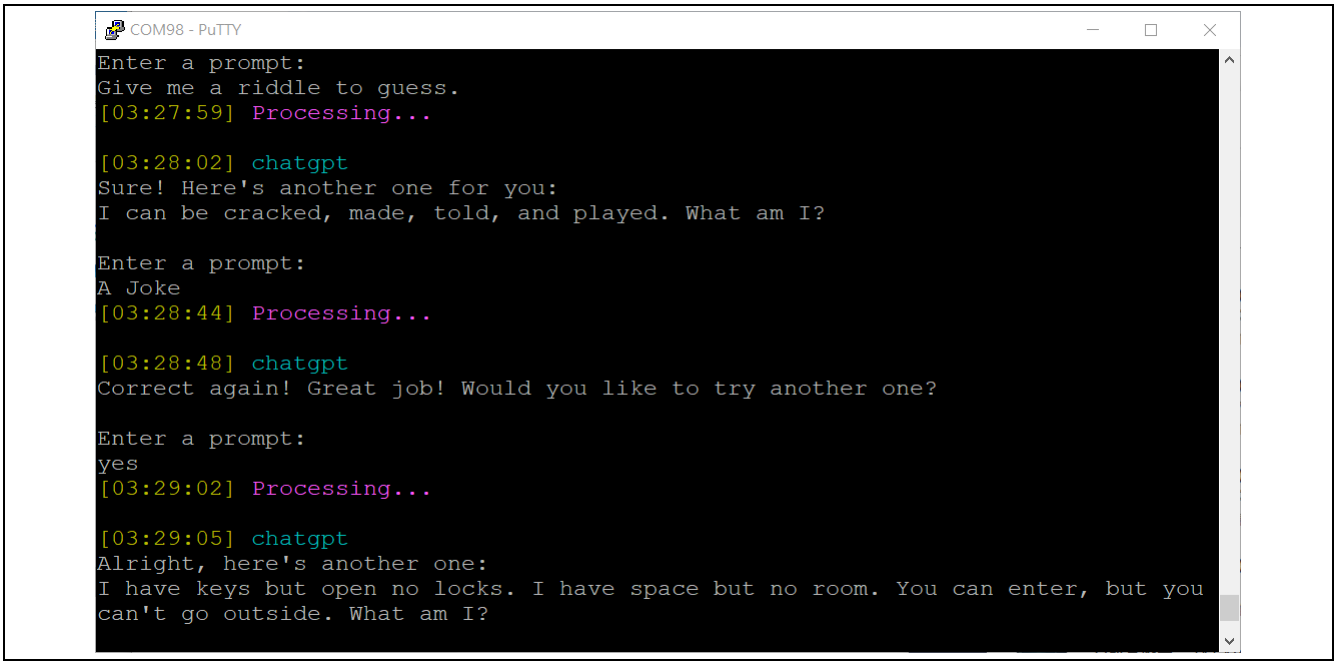

Figure 1-3 ChatGPT Chat Functionality Example 1

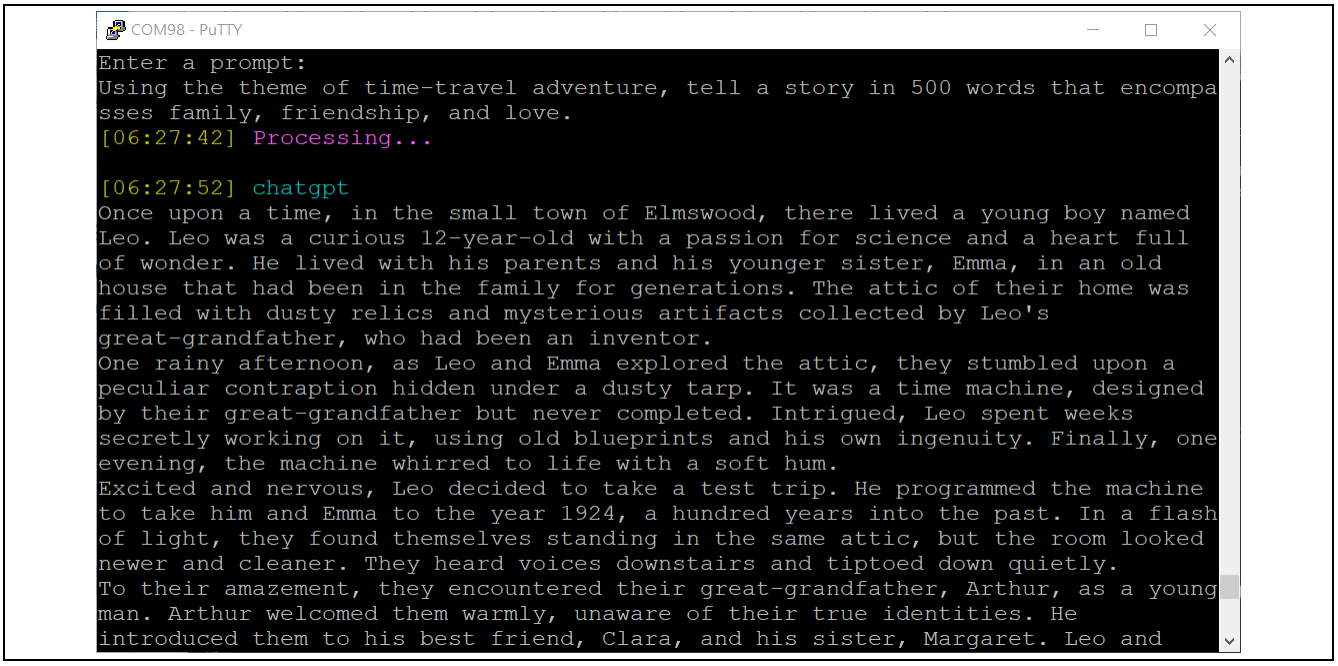

Figure 1-4 ChatGPT Chat Functionality Example 2

The execution results of the image generation functionality of this example program are shown in Figure 1-5. The instruction for image generation is "image:", and after that, you input the prompt for image generation. ChatGPT will then generate the corresponding image. Once the generation process is complete, the URL of the generated image will be displayed in the window. The image will also be downloaded locally with the file name *image.png*.

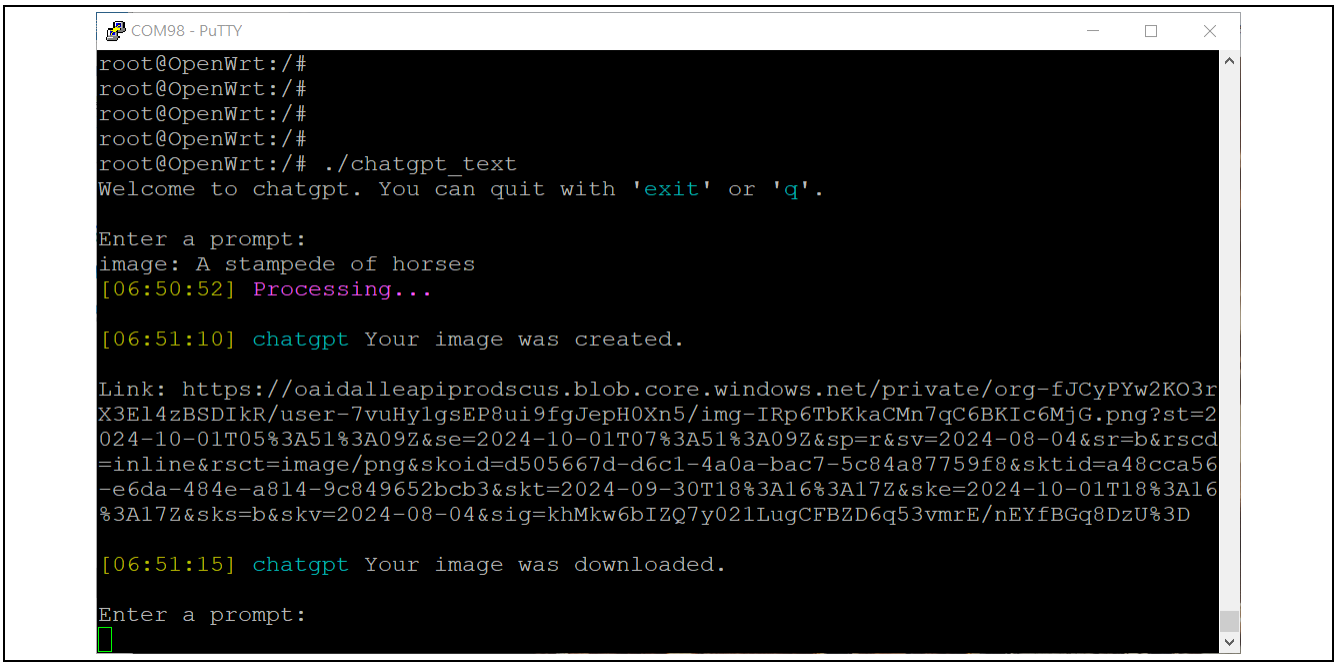

Figure 1-5 Code Length and CRC32 Checksum

During image generation, the prompt can be a simple word or phrase, for example,

A stampede of horses

as the prompt. The resulting image is shown in Figure 1-6.

Figure 1-6 ChatGPT Image Generation 1

The prompt can also be a longer and more complex paragraph, for example,

In this fierce war, elves, dwarves, and humans united, facing the evil forces of the dark creatures and the vile orcs. Warriors from each race fought valiantly, swords clashing with axes, arrows raining down like a storm. Elven archers perched in the treetops unleashed a deadly barrage, dwarven axes cleaved through enemy shields, and human warriors charged into the fray. Flames and magic intertwined, creating a breathtaking spectacle on the battlefield. The bloodshed between races persisted, but their unity and courage allowed them to resist the dark forces, defending the hope for freedom and peace.

as the prompt. The resulting image is shown in Figure 1-7.

Figure 1-7 ChatGPT Image Generation 2

The prompt can also be a description for image generation, including various specific requirements, for example,

Create a four-panel comic in a 3D style, depicting a fight between a white rabbit and a fox.

as the prompt. The resulting image is shown in Figure 1-8.

Figure 1-8 ChatGPT Image Generation 3

2. Code Description

In generative AI text, special characters are often used. However, these special characters can cause formatting and parsing errors when using the OpenAI API. Therefore, it is necessary to preprocess the text in advance to ensure that it does not affect the program's execution.

```
# 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]'
}
```

When the user inputs a message to ChatGPT, it needs to be formatted into the standard format of the ChatGPT model before calling the OpenAI API. Additionally, the interactive response of ChatGPT is related to the previous conversation. Therefore, this new user message will be added to the end of the conversation message list and together serve as the input text for the model.

```
# 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
}
```

When it comes to the interactive responses of ChatGPT, they are related to the previous conversation. This includes both the user's input messages, and the responses generated by ChatGPT. Therefore, after ChatGPT generates a response, it should be added to the end of the conversation message list.

Additionally, in the OpenAI API, a token refers to the smallest unit of text. The pricing and limitations of the API are based on the number of tokens used. In the code, there is usually a maximum token limit set. If the number of tokens in the input text exceeds this limit, it needs to be truncated. The rule is to remove data starting from the oldest until it fits within the limit. By default, the limit is set to 4096 tokens, but users can adjust it according to their needs.

```
# 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
}
```

The following is the code for the chat functionality, which calls the curl library to make a request to the OpenAI server using the ChatGPT model and provides a message list containing the conversation. The model will then return a response. In the OpenAI API, the "model" parameter refers to the ChatGPT model, which is set to "gpt-4o" by default. The "messages" parameter represents the conversation message list so far, which is the input parameter for this function. It undergoes some processing to ensure it conforms to the parameter format required by the OpenAI API. The "max_completion_tokens" parameter specifies the maximum number of tokens for the generated text, controlling the length of the generated result. Its default value is 4096. The "temperature" parameter is used to control the randomness and diversity of the generated text. Its value can range from 0 to 2, with a default value of 0.7. When set to a higher value, the generated text will be more random and diverse. This means that the model may choose options with lower probabilities and produce more unexpected responses. Conversely, when set to a lower value, the generated text will prioritize determinism and consistency. For a comprehensive understanding of all parameters within the OpenAI API, please refer to the 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'
     }'
}
```

Because the response text from the ChatGPT model may contain multiple languages, handling line breaks when displaying the response text in the terminal window is done for some common languages such as English, Chinese, Japanese, etc. This is done to facilitate readability.

```
# 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
          }
     }
    …
    …
}
```

If there is an error reported by the OpenAI server while using the OpenAI API, the relevant error message will be displayed in the terminal window.

```
# 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
}
```

The following is the code for the image generation functionality, which calls the curl library to make a request to the OpenAI server using the dall-e-3 model and provides a prompt. The model will then generate a new image. In the OpenAI API, the "model" parameter refers to the image generation model, which is set to "dall-e-3" by default. The "prompt" parameter is the

prompt for image generation. The input parameter for this function is the instruction for image generation, which is "image:", followed by the prompt for image generation. After removing the instruction for image generation, only the prompt for image generation is retained as the setting for this parameter. The "n" parameter specifies the number of generated images, with a default value of 1. The "size" parameter specifies the size of the generated images, with a default value of 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"'"
    }')
}
```

In addition to the dall-e-3 model, there is also an option to choose the older version, dall-e-2 model. The table below shows the differences in the aforementioned parameters between the two models. Users can choose the image generation model based on their specific needs.

OpenAI API parameter	dall-e-2	dall-e-3
prompt	Maximum 1000 characters	Maximum 4000 characters
n	1~10	1
size	256x256 512x512 1024x1024	1024x1024 1792x1024 1024x1792

Table 2-1 Comparison of Parameters between dall-e-2 and dall-e-3 Models

The following are default parameter values for OpenAI models, which can be modified by users according to their needs. Only the last parameter, “SIZE”, is related to image generation, while the rest of the parameters are related to chat functionality. It should be noted that the “MAX_COMPLETION_TOKENS” parameter is only supported in models GPT-4 and later.

```
# 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}
```

The while loop at the bottom of the program contains the main code for both the chat and image generation functionalities. When prompted for input, if the user enters "exit" or "q", the program will terminate. If the user enters a string starting with "image:", the image generation

functionality will be triggered. Otherwise, the program will proceed with the chat functionality. During image generation, the remote URL of the generated image result will be displayed in the window, and the image will also be downloaded to the local directory.

```
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. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - MA35D1 Series Buildroot 2024

3.2 Hardware Requirements

- Circuit components
  - NuMaker-IoT-MA35D16F90 V2.4

4. Directory Information

The directory structure is shown below.

rootfs_overlay	
etc	System configuration directory
network	Network configuration directory
profile.d	Login script directory
root	Root user's home directory

Figure 4-1 Directory Information

5. Example Code Execution

1. Connect to the VCOM of the NuMaker-IoT-MA35D16F90 evaluation board using terminal software.
2. After booting Buildroot, ensure that the device can successfully connect to the network and that the system time is accurate. This is necessary for the proper execution of the ChatGPT example program. You can use the “ping” and “date” commands to verify these conditions.

```
# 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. In the system environment, configure the self-obtained OpenAI API key.

```
# export OPENAI_KEY=your_key_here
```

4. Run the ChatGPT example program. For this part, you can refer to the following description.
   - Before running the example program, you can use the "-h" parameter to view the help message.

```
# ./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)
```

   - You can directly run the example program to enter the interactive mode, as shown in the previous section's execution result. When entering regular text during the

conversation, the chat functionality will be invoked. However, if you prefix the text with "image:", the image generation functionality will be called.

```
# ./chatgpt_text
```

- When running the example program, you can also directly specify the values of the parameters to replace the default values in the code. Here is an example of using the chat model gpt-3.5-turbo. It should be noted that many chat models have message formats that are not compatible with gpt-4o. To use these models, you need to analyze their message formats and modify the relevant content of the example program.

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

- When running, using the "-l" parameter will list all the currently supported models by the official.

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

- When running, using the "-p" parameter allows you to directly specify a chat prompt and receive a response without entering the interactive mode. It is important to note that this method is only suitable for one-time standalone queries, so the response will not interact with previous text.

```
# ./chatgpt_text -p "How much seawater is there on Earth in total?"

[07:47:35]
There are approximately 1.332 billion cubic kilometers of seawater on Earth.
#
```

6. Revision History

Date	Revision	Description
2025.04.25	1.00	Initial version.

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.