ChatMetamaterials

Source code manual

1. Developer contribution

Conceptualization: Zhenyang Gao
ChatMetamaterials implementation: Zhenyang Gao
Supervision: Hongze Wang, Yi Wu, Kun Zhou, Yang Lu, Minh-Son Pham

2. Getting started

This documentation enables users to deploy the ChatMetamaterials source code locally on their laptops without needing to connect to the server. For specific meanings and interpretations of the passed values and data, please refer to our website: MetaVerse Labs. This document serves as a concise handbook for using the local deployment version of our source code. Although it is local deployment, users must ensure they have the correct VPN and network connection settings, similar to those required by OpenAI. To use the source code, first establish the Python programming environment as outlined on our website. Note that the local deploy is ONLY supported by python 3.10.11 for decryption. Once the environment is set up, open the terminal, navigate to the program's directory, and use the following commands to install the required packages in .venv:

```
pip install rhino3dm
pip install numpy
pip install Flask
pip install openai==0.28.0
pip install requests
pip install scipy
pip install cryptography
pip install scikit-learn
```

The source codes and dataset can also be downloaded in https://github.com/ChatMetamaterials/SourceCode/. After installing the required packages, utilize Git Large File Storage (LFS) to transfer the source code and dataset to your programming directory, including the following files:

```
database_official_0303.bin
Access_database.pyc
DP_algorithms.pyc
Gyroid_algorithms.pyc
Irrational_algorithms.pyc
ChatMetamaterials_front_official_API.pyc
structure_analysis_official_API.pyc
pyarmor_runtime_000000
```

It is important to note that `database_official_0303.bin` is a large GitHub file. To download it correctly, Git LFS should be used; a direct download via the website results in only a 1KB pointer file, not the actual database, which can lead to a fatal error (e.g., `InvalidToken cryptography.fernet.InvalidToken` message indicating decryption failure). For Git and LFS installation instructions on Windows, please refer to https://git-lfs.com/ and https://git-scm.com/downloads/win. Clone the repository using:

```
git clone https://github.com/ChatMetamaterials/SourceCode.git
```

Create your own .py file within the directory, naming it local.py as an example. Afterwards, import the two major interfaces from our source codes to begin programming. Here's a basic setup:

```
from ChatMetamaterials_front_official_API import *
from structure_analysis_official_API import *
```

3. Classification API

We developed the classification API to categorize metamaterial design prompts into one of three types: (1) basic cell properties design, (2) deformation recognition, and (3) research and innovation tasks. To use this API, you can write the following code:

```
Input = "Can you design a unit cell with modulus at 10 MPa, another one with modulus at 25 MPa."
file_name = "Example_classification.txt"
json_data = API_classification (Input, failure_try = 3)
write_json_to_txt(json_data,file_name)
```

where `Input` is the prompt string to be classified, and `file_name` is the filename where JSON information will be stored. The data returned from the classification can either be stored in a .txt file or passed directly as a json.dumps() string object from the function. To further process this data, you will use the `extract_data_json` and `extract_data_txt` functions. Here's an example of extracting the data:

```
data = extract_data_json (json_data)
```

or

```
data = extract_data_txt ("Example_classification.txt")
```

Note that the `data` returned here represents the category values "1", "2", or "3", corresponding to the categories (1) basic cell properties design, (2) deformation recognition, and (3) research and innovation, as previously discussed.

4. Basic cell property design API

For prompts classified under basic properties of unit cells, users can utilize the basic cell property design API to access the metadata list for those cells as interpreted by ChatMetamaterials. Here's an example of how you can implement this:

```
Input = "Can you design a unit cell with modulus at 10 MPa, another one with modulus at 25 MPa."
Image = ""
file_name = "Example_basic_cells.txt"
json_data = API_basic_cells (Input, Image)
```

```
write_json_to_txt(json_data, file_name)
```

where `Image` represents a potential image URL that can be attached for the API to process. The `json_data` is the returned JSON string as previously described. For a detailed interpretation of the data structure processed here, please visit our website. To pass and use the data, please follow the detailed instructions given in **Section 2**.

5. Deformation recognition API

For prompts that involve recognizing complex deformation patterns based on raw human inputs like text and images, you can use the deformation recognition API to obtain the relevant deformation pattern metadata. Here's an example of how to implement this:

```
Input = "Please design a metamaterial (default size), that fails first with the letter S indicated in the image."
Image = "https://s3.bmp.ovh/imgs/2025/02/19/46a35e29cf2b3385.png"
file_name = "Example_deformation_recognition.txt"
json_data = API_deformation_pattern_recognition (Input, Image)
write_json_to_txt(json_data, file_name)
```

where the `Image` parameter would be a hand-drawn image of the deformation pattern. For a detailed interpretation of the data structure processed here, please refer to our website. To pass and utilize the data effectively, please follow the instructions provided in Section 2.

6. Research and innovation API

For prompts that aim to emulate human-like research and innovation, you can use the specified API to obtain relevant metadata. Here's an example of how to structure the code for such an interaction:

```
Input = "Create a metamaterial containing precipitates triggering precipitation toughening."
file_name = "Example_research_innovation.txt"
Image = ""
json_data = API_research_innovation (Input, Image)
write_json_to_txt(json_data,file_name)
```

In this scenario, it's important to note that using an image is not recommended. For a detailed interpretation of the data structure processed by this API, please refer to our website. To effectively pass and utilize the data, follow the instructions provided in **Section 2**.

7. Shoe sole design API

Currently, ChatMetamaterials' product design API is limited to shoe sole designs. However, this capability can be quickly expanded to meet industrial needs or through academic collaborations. If you have potential interest in expanding the range of products or collaborating, please don't hesitate to contact the developers. To design a customized shoe sole using the API, here is an example:

```
Input = "Please design a shoe sole according to the shoe sole design properties shown in the image."
file_name = "Example_shoe_sole.txt"
Image = "https://s3.bmp.ovh/imgs/2025/02/21/4e3654d36137abb1.png"
json_data = API_design_shoe (Input, Image)
write_json_to_txt(json_data, file_name)
```

where the Image parameter would include a shoe sole drawing attached to the API call. The product data follows the same format as the research and innovation metadata to ensure cell package flexibility for multiple products in the future. For a detailed interpretation of the data structure processed by this API, please refer to our website. To effectively pass and utilize the data, follow the instructions provided in **Section 2**.

8. Universal metamaterial design API

One of the most distinctive features of ChatMetamaterials is its universal metamaterial design API. Currently, we support designs for damage-programmable metamaterials, irrational (point-cloud based) metamaterials, and hierarchical TPMS gyroid metamaterials. However, we are fully prepared to expand this offering based on your specific research needs. We eagerly invite collaborations to broaden the scope of our API, as we aim to learn from, contribute to, and accelerate advancements across the broad spectrum of metamaterial communities. Please don’t hesitate to contact us for future collaborations to explore how we can work together on new metamaterials. Here is an example of usage:

```
Input = "Design an irrational metamaterial that has 100 random nodes uniformly spread within the default design space."
file_name = "Example_universal_metamaterial.txt"
Type = 1
json_data = API_special_agents(Input, Type)
write_json_to_txt(json_data,file_name)
```

In this context, the Type parameter within the ChatMetamaterials knowledge base specifies the category of metamaterials being referenced. Here, the values "0" to "2" correspond to the different types of designs supported: "0" represents damage-programmable architectures; "1" refers to irrational (point-cloud) architectures; "2" denotes hierarchical TPMS gyroid architectures. For a detailed interpretation of the data structure processed by this API and how these types are implemented, please refer to our website. To effectively pass and utilize the data, ensure you follow the instructions provided in **Section 2** of our documentation.

9. Architecture generation APIs

To generate architectures for metamaterials, it’s essential to preprocess the metadata obtained from various APIs into a consistent format suitable for structural calculation algorithms. Here's how the preprocessing is conducted across different types of metamaterial design data:

(1) For unit cell properties design:

```
# Example metadata for basic cells properties
metadata_basic_cells = extract_data_txt ("Example_basic_cells.txt")
# Preprocess the metadata to make it suitable for structural calculations
architectural_metadata =
metadata_for_multi_single_cells(metadata_basic_cells)
```

Here, `metadata_basic_cells` represents the metadata obtained from the `API_basic_cells`, and `architectural_metadata` is the processed metadata ready for further computations.

(2) For deformation recognition:

```
# Example metadata for deformation recognition
metadata_deformation_recognition = extract_data_txt
("Example_deformation_recognition.txt")
# Convert the raw metadata into a format usable by deformation pattern
algorithms
architectural_metadata =
metadata_for_deformation_pattern(metadata_deformation_recognition)
```

In this case, `metadata_deformation_recognition` is the raw data obtained from the `API_deformation_pattern_recognition`.

(3) For research and innovation and shoe sole design:

```
# Example metadata for research and innovation or shoe sole design
metadata_research_innovation = extract_data_txt
("Example_research_innovation.txt")
# Adapt the metadata for research and innovation algorithms
architectural_metadata =
metadata_for_research_and_innovation(metadata_research_innovation)
```

where `metadata_research_innovation` represents data from either `API_research_innovation` or `API_design_shoe`, processed to be compatible with the respective algorithms.

Once `architectural_metadata` is prepared for each category, the `save_rhino_metamaterials_object` can then be applied to generate the actual architectures' 3D model from the basic cells design, deformation recognition, research and innovation, and product design APIs:

```
pass_architectural_metadata = extract_data_json(architectural_metadata)
file_path = "your_path/your_filename.3dm"
save_rhino_metamaterials_object(pass_architectural_metadata)
```

For universal architectures, considering the potential of different modeling software to be used, we provide the design architecture data instead. These data can be easily converted to model information and used for generating the 3D models under different software. Here is an example of usage:

```
file_name = "irrational_architectural_data.txt"
metadata_extended = extract_data_txt ("Example_universal_metamaterial.txt")
```

```
Type = "irrational"
architectures = lines_for_special_agents(metadata_extended, Type)
write_json_to_txt(architectures, file_name)
```