code_utils

content_str

```
def content_str(content: Union[str, List, None]) -> str
```

Converts `content` into a string format.

This function processes content that may be a string, a list of mixed text and image URLs, or None, and converts it into a string. Text is directly appended to the result string, while image URLs are represented by a placeholder image token. If the content is None, an empty string is returned.

Arguments:

- content (Union[str, List, None]): The content to be processed. Can be a string, a list of dictionaries representing text and image URLs, or None.

Returns:

- `str` - A string representation of the input content. Image URLs are replaced with an image token.

Notes:

- The function expects each dictionary in the list to have a "type" key that is either "text" or "image_url". For "text" type, the "text" key's value is appended to the result. For "image_url", an image token is appended.
- This function is useful for handling content that may include both text and image references, especially in contexts where images need to be represented as placeholders.

infer_lang

```
def infer_lang(code)
```

infer the language for the code. TODO: make it robust.

extract_code

```
def extract_code(
        text: Union[str, List],
        pattern: str = CODE_BLOCK_PATTERN,
        detect_single_line_code: bool = False) -> List[Tuple[str, str]]
```

Extract code from a text.

Arguments:

- `text` *str or List* - The content to extract code from. The content can be a string or a list, as returned by standard GPT or multimodal GPT.
- `pattern` *str, optional* - The regular expression pattern for finding the code block. Defaults to CODE_BLOCK_PATTERN.
- `detect_single_line_code` *bool, optional* - Enable the new feature for extracting single line code. Defaults to False.

Returns:

- `list` - A list of tuples, each containing the language and the code. If there is no code block in the input text, the language would be "unknown". If there is code block but the language is not specified, the language would be "".

generate_code

```
def generate_code(pattern: str = CODE_BLOCK_PATTERN,
                  **config) -> Tuple[str, float]
```

(openai<1) Generate code.

Arguments:

- `pattern` *Optional, str* - The regular expression pattern for finding the code block. The default pattern is for finding a code block in a markdown file.
- `config` *Optional, dict* - The configuration for the API call.

Returns:

- `str` - The generated code.
- `float` - The cost of the generation.

improve_function

```
def improve_function(file_name, func_name, objective, **config)
```

(openai<1) Improve the function to achieve the objective.

improve_code

```
def improve_code(files, objective, suggest_only=True, **config)
```

(openai<1) Improve the code to achieve a given objective.

Arguments:

- `files` *list* - A list of file names containing the source code.
- `objective` *str* - The objective to achieve.
- `suggest_only` *bool* - Whether to return only the suggestions or the improved code.
- `config` *Optional, dict* - The configuration for the API call.

Returns:

- `str` - The improved code if suggest_only=False; a list of suggestions if suggest_only=True (default).
- `float` - The cost of the generation.

execute_code

```
def execute_code(code: Optional[str] = None,
                 timeout: Optional[int] = None,
                 filename: Optional[str] = None,
                 work_dir: Optional[str] = None,
                 use_docker: Optional[Union[List[str], str, bool]] = None,
                 lang: Optional[str] = "python") -> Tuple[int, str, str]
```

Execute code in a docker container. This function is not tested on MacOS.

Arguments:

- `code` *Optional, str* - The code to execute. If None, the code from the file specified by filename will be executed. Either code or filename must be provided.
- `timeout` *Optional, int* - The maximum execution time in seconds. If None, a default timeout will be used. The default timeout is 600 seconds. On Windows, the timeout is not enforced when use_docker=False.
- `filename` *Optional, str* - The file name to save the code or where the code is stored when `code` is None. If None, a file with a randomly generated name will be created. The randomly generated file will be deleted after execution. The file name must be a relative path. Relative paths are relative to the working directory.
- `work_dir` *Optional, str* - The working directory for the code execution. If None, a default working directory will be used. The default working directory is the "extensions" directory under "path_to_autogen".
- `use_docker` *Optional, list, str or bool* - The docker image to use for code execution. If a list or a str of image name(s) is provided, the code will be executed in a docker container with the first image successfully pulled. If None, False or empty, the code will be executed in the current environment. Default is None, which will be converted into an empty list when docker package is available. Expected behaviour:
  - If `use_docker` is explicitly set to True and the docker package is available, the code will run in a Docker container.
  - If `use_docker` is explicitly set to True but the Docker package is missing, an error will be raised.
  - If `use_docker` is not set (i.e., left default to None) and the Docker package is not available, a warning will be displayed, but the code will run natively. If the code is executed in the current environment, the code must be trusted.
- `lang` *Optional, str* - The language of the code. Default is "python".

Returns:

- `timeout`0 - 0 if the code executes successfully.
- `timeout`1 - The error message if the code fails to execute; the stdout otherwise.

- `timeout` 2 - The docker image name after container run when docker is used.

generate_assertions

```
def generate_assertions(definition: str, **config) -> Tuple[str, float]
```

(openai<1) Generate assertions for a function.

Arguments:

- `definition` *str* - The function definition, including the signature and docstr.
- `config` *Optional, dict* - The configuration for the API call.

Returns:

- `str` - The generated assertions.
- `float` - The cost of the generation.

eval_function_completions

```
def eval_function_completions(responses: List[str],
                              definition: str,
                              test: Optional[str] = None,
                              entry_point: Optional[str] = None,
                              assertions: Optional[Union[str, Callable[
                                  [str], Tuple[str, float]]]] = None,
                              timeout: Optional[float] = 3,
                              use_docker: Optional[bool] = True) -> Dict
```

(openai<1) Select a response from a list of responses for the function completion task (using generated assertions), and/or evaluate if the task is successful using a gold test.

Arguments:

- `responses` *list* - The list of responses.
- `definition` *str* - The input definition.
- `test` *Optional, str* - The test code.
- `entry_point` *Optional, str* - The name of the function.
- `assertions` *Optional, str or Callable* - The assertion code which serves as a filter of the responses, or an assertion generator. When provided, only the responses that pass the assertions will be considered for the actual test (if provided).
- `timeout` *Optional, float* - The timeout for executing the code.

Returns:

- `dict` - The success metrics.

PassAssertionFilter Objects

```
class PassAssertionFilter()
```

pass_assertions

```
def pass_assertions(context, response, **_)
```

(openai<1) Check if the response passes the assertions.

implement

```
def implement(
    definition: str,
    configs: Optional[List[Dict]] = None,
    assertions: Optional[Union[str,
                               Callable[[str],
                                        Tuple[str,
                                              float]]]] = generate_assertions
) -> Tuple[str, float]
```

(openai<1) Implement a function from a definition.

Arguments:

- `definition` *str* - The function definition, including the signature and docstr.
- `configs` *list* - The list of configurations for completion.
- `assertions` *Optional, str or Callable* - The assertion code which serves as a filter of the responses, or an assertion generator.

Returns:

- `str` - The implementation.
- `float` - The cost of the implementation.
- `int` - The index of the configuration which generates the implementation.

Edit this page

Previous
« agent_utils

Next
function_utils »

Community

Discord

Twitter