On this page

oai.openai_utils

get_key

```
def get_key(config: Dict[str, Any]) -> str
```

Get a unique identifier of a configuration.

Arguments:

- `config` *dict or list* - A configuration.

Returns:

- `tuple` - A unique identifier which can be used as a key for a dict.

get_config_list

```
def get_config_list(api_keys: List,
                    base_urls: Optional[List] = None,
                    api_type: Optional[str] = None,
                    api_version: Optional[str] = None) -> List[Dict]
```

Get a list of configs for OpenAI API client.

Arguments:

- `api_keys` *list* - The api keys for openai api calls.
- `base_urls` *list, optional* - The api bases for openai api calls. If provided, should match the length of api_keys.
- `api_type` *str, optional* - The api type for openai api calls.
- `api_version` *str, optional* - The api version for openai api calls.

Returns:

- `list` - A list of configs for OepnAI API calls.

Example:

```
# Define a list of API keys
api_keys = ['key1', 'key2', 'key3']

# Optionally, define a list of base URLs corresponding to each API key
base_urls = ['https://api.service1.com', 'https://api.service2.com', 'https://api.service3.com']

# Optionally, define the API type and version if they are common for all keys
api_type = 'azure'
api_version = '2023-12-01-preview'

# Call the get_config_list function to get a list of configuration dictionaries
config_list = get_config_list(api_keys, base_urls, api_type, api_version)
```

config_list_openai_aoai

```
def config_list_openai_aoai(
        key_file_path: Optional[str] = ".",
        openai_api_key_file: Optional[str] = "key_openai.txt",
        aoai_api_key_file: Optional[str] = "key_aoai.txt",
        openai_api_base_file: Optional[str] = "base_openai.txt",
        aoai_api_base_file: Optional[str] = "base_aoai.txt",
        exclude: Optional[str] = None) -> List[Dict]
```

Get a list of configs for OpenAI API client (including Azure or local model deployments that support OpenAI's chat completion API).

This function constructs configurations by reading API keys and base URLs from environment variables or text files. It supports configurations for both OpenAI and Azure OpenAI services, allowing for the exclusion of one or the other. When text files are used, the environment variables will be overwritten. To prevent text files from being used, set the corresponding file name to None. Or set key_file_path to None to disallow reading from text files.

Arguments:

- `key_file_path` *str, optional* - The directory path where the API key files are located. Defaults to the current directory.
- `openai_api_key_file` *str, optional* - The filename containing the OpenAI API key. Defaults to 'key_openai.txt'.
- `aoai_api_key_file` *str, optional* - The filename containing the Azure OpenAI API key. Defaults to 'key_aoai.txt'.
- `openai_api_base_file` *str, optional* - The filename containing the OpenAI API base URL. Defaults to 'base_openai.txt'.
- `aoai_api_base_file` *str, optional* - The filename containing the Azure OpenAI API base URL. Defaults to 'base_aoai.txt'.
- `exclude` *str, optional* - The API type to exclude from the configuration list. Can be 'openai' or 'aoai'. Defaults to None.

Returns:

- `List[Dict]` - A list of configuration dictionaries. Each dictionary contains keys for 'api_key', and optionally 'base_url', 'api_type', and 'api_version'.

Raises:

- `FileNotFoundError` - If the specified key files are not found and the corresponding API key is not set in the environment variables.

Example:

To generate configurations excluding Azure OpenAI:

configs = config_list_openai_aoai(exclude='aoai')

File samples:

- key_aoai.txt

```
aoai-12345abcdef67890ghijklmnopqr
aoai-09876zyxwvuts54321fedcba
```

- base_aoai.txt

```
https://api.azure.com/v1
https://api.azure2.com/v1
```

Notes:

- The function checks for API keys and base URLs in the following environment variables: 'OPENAI_API_KEY', 'AZURE_OPENAI_API_KEY', 'OPENAI_API_BASE' and 'AZURE_OPENAI_API_BASE'. If these are not found, it attempts to read from the specified files in the 'key_file_path' directory.
- The API version for Azure configurations is set to DEFAULT_AZURE_API_VERSION by default.
- If 'exclude' is set to 'openai', only Azure OpenAI configurations are returned, and vice versa.
- The function assumes that the API keys and base URLs in the environment variables are separated by new lines if there are multiple entries.

config_list_from_models

```
def config_list_from_models(
        key_file_path: Optional[str] = ".",
        openai_api_key_file: Optional[str] = "key_openai.txt",
        aoai_api_key_file: Optional[str] = "key_aoai.txt",
        aoai_api_base_file: Optional[str] = "base_aoai.txt",
        exclude: Optional[str] = None,
        model_list: Optional[list] = None) -> List[Dict]
```

Get a list of configs for API calls with models specified in the model list.

This function extends `config_list_openai_aoai` by allowing to clone its' out for each of the models provided. Each configuration will have a 'model' key with the model name as its value. This is particularly useful when all endpoints have same set of models.

Arguments:

- `key_file_path` *str, optional* - The path to the key files.
- `openai_api_key_file` *str, optional* - The file name of the OpenAI API key.
- `aoai_api_key_file` *str, optional* - The file name of the Azure OpenAI API key.
- `aoai_api_base_file` *str, optional* - The file name of the Azure OpenAI API base.
- `exclude` *str, optional* - The API type to exclude, "openai" or "aoai".
- `model_list` *list, optional* - The list of model names to include in the configs.

Returns:

- `list` - A list of configs for OpenAI API calls, each including model information.

Example:

```
    # Define the path where the API key files are located
    key_file_path = '/path/to/key/files'

    # Define the file names for the OpenAI and Azure OpenAI API keys and bases
    openai_api_key_file = 'key_openai.txt'
    aoai_api_key_file = 'key_aoai.txt'
    aoai_api_base_file = 'base_aoai.txt'

    # Define the list of models for which to create configurations
    model_list = ['gpt-4', 'gpt-3.5-turbo']

    # Call the function to get a list of configuration dictionaries
    config_list = config_list_from_models(
        key_file_path=key_file_path,
        openai_api_key_file=openai_api_key_file,
        aoai_api_key_file=aoai_api_key_file,
        aoai_api_base_file=aoai_api_base_file,
        model_list=model_list
    )

    # The `config_list` will contain configurations for the specified models, for example:
    # [
    #     {'api_key': '...', 'base_url': 'https://api.openai.com', 'model': 'gpt-4'},
    #     {'api_key': '...', 'base_url': 'https://api.openai.com', 'model': 'gpt-3.5-turbo'}
    # ]
```

config_list_gpt4_gpt35

```
def config_list_gpt4_gpt35(
        key_file_path: Optional[str] = ".",
        openai_api_key_file: Optional[str] = "key_openai.txt",
        aoai_api_key_file: Optional[str] = "key_aoai.txt",
        aoai_api_base_file: Optional[str] = "base_aoai.txt",
        exclude: Optional[str] = None) -> List[Dict]
```

Get a list of configs for 'gpt-4' followed by 'gpt-3.5-turbo' API calls.

Arguments:

- `key_file_path` *str, optional* - The path to the key files.
- `openai_api_key_file` *str, optional* - The file name of the openai api key.
- `aoai_api_key_file` *str, optional* - The file name of the azure openai api key.
- `aoai_api_base_file` *str, optional* - The file name of the azure openai api base.
- `exclude` *str, optional* - The api type to exclude, "openai" or "aoai".

Returns:

- `list` - A list of configs for openai api calls.

filter_config

```
def filter_config(config_list, filter_dict)
```

This function filters `config_list` by checking each configuration dictionary against the criteria specified in `filter_dict`. A configuration dictionary is retained if for every key in `filter_dict`, see example below.

Arguments:

- `config_list` *list of dict* - A list of configuration dictionaries to be filtered.
- `filter_dict` *dict* - A dictionary representing the filter criteria, where each key is a field name to check within the configuration dictionaries, and the corresponding value is a list of acceptable values for that field. If the configuration's field's value is not a list, then a match occurs when it is found in the list of acceptable values. If the configuration's field's value is a list, then a match occurs if there is a non-empty intersection with the acceptable values.

Returns:

list of dict: A list of configuration dictionaries that meet all the criteria specified in `filter_dict`.

Example:

```
    # Example configuration list with various models and API types
    configs = [
        {'model': 'gpt-3.5-turbo'},
        {'model': 'gpt-4'},
        {'model': 'gpt-3.5-turbo', 'api_type': 'azure'},
        {'model': 'gpt-3.5-turbo', 'tags': ['gpt35_turbo', 'gpt-35-turbo']},
    ]

    # Define filter criteria to select configurations for the 'gpt-3.5-turbo' model
    # that are also using the 'azure' API type
    filter_criteria = {
        'model': ['gpt-3.5-turbo'],  # Only accept configurations for 'gpt-3.5-turbo'
        'api_type': ['azure']       # Only accept configurations for 'azure' API type
    }

    # Apply the filter to the configuration list
    filtered_configs = filter_config(configs, filter_criteria)

    # The resulting `filtered_configs` will be:
    # [{'model': 'gpt-3.5-turbo', 'api_type': 'azure', ...}]


    # Define a filter to select a given tag
    filter_criteria = {
        'tags': ['gpt35_turbo'],
    }

    # Apply the filter to the configuration list
    filtered_configs = filter_config(configs, filter_criteria)

    # The resulting `filtered_configs` will be:
    # [{'model': 'gpt-3.5-turbo', 'tags': ['gpt35_turbo', 'gpt-35-turbo']}]
```

Notes:

- If `filter_dict` is empty or None, no filtering is applied and `config_list` is returned as is.
- If a configuration dictionary in `config_list` does not contain a key specified in `filter_dict`, it is considered a non-match and is excluded from the result.
- If the list of acceptable values for a key in `filter_dict` includes None, then configuration dictionaries that do not have that key will also be considered a match.

config_list_from_json

```
def config_list_from_json(
    env_or_file: str,
    file_location: Optional[str] = "",
    filter_dict: Optional[Dict[str, Union[List[Union[str, None]],
                                          Set[Union[str, None]]]]] = None
) -> List[Dict]
```

Retrieves a list of API configurations from a JSON stored in an environment variable or a file.

This function attempts to parse JSON data from the given `env_or_file` parameter. If `env_or_file` is an environment variable containing JSON data, it will be used directly. Otherwise, it is assumed to be a filename, and the function will attempt to read the file from the specified `file_location`.

The `filter_dict` parameter allows for filtering the configurations based on specified criteria. Each key in the `filter_dict` corresponds to a field in the configuration dictionaries, and the associated value is a list or set of acceptable values for that field. If a field is missing in a configuration and `None` is included in the list of acceptable values for that field, the configuration will still be considered a match.

Arguments:

- `env_or_file` *str* - The name of the environment variable, the filename, or the environment variable of the filename that containing the JSON data.
- `file_location` *str, optional* - The directory path where the file is located, if `env_or_file` is a filename.
- `filter_dict` *dict, optional* - A dictionary specifying the filtering criteria for the configurations, with keys representing field names and values being lists or sets of acceptable values for those fields.

Example:

`env_or_file`0

Returns:

- `env_or_file`1 - A list of configuration dictionaries that match the filtering criteria specified in `filter_dict`.

Raises:

- `env_or_file`3 - if env_or_file is neither found as an environment variable nor a file

get_config

```
def get_config(api_key: str,
               base_url: Optional[str] = None,
               api_type: Optional[str] = None,
               api_version: Optional[str] = None) -> Dict
```

Constructs a configuration dictionary for a single model with the provided API configurations.

Example:

```
config = get_config(
    api_key="sk-abcdef1234567890",
    base_url="https://api.openai.com",
    api_version="v1"
)
# The 'config' variable will now contain:
# {
#     "api_key": "sk-abcdef1234567890",
#     "base_url": "https://api.openai.com",
#     "api_version": "v1"
# }
```

Arguments:

- `api_key` *str* - The API key for authenticating API requests.
- `base_url` *Optional[str]* - The base URL of the API. If not provided, defaults to None.
- `api_type` *Optional[str]* - The type of API. If not provided, defaults to None.
- `api_version` *Optional[str]* - The version of the API. If not provided, defaults to None.

Returns:

- `Dict` - A dictionary containing the provided API configurations.

config_list_from_dotenv

```
def config_list_from_dotenv(
    dotenv_file_path: Optional[str] = None,
    model_api_key_map: Optional[dict] = None,
    filter_dict: Optional[dict] = None
) -> List[Dict[str, Union[str, Set[str]]]]
```

Load API configurations from a specified .env file or environment variables and construct a list of configurations.

This function will:

- Load API keys from a provided .env file or from existing environment variables.
- Create a configuration dictionary for each model using the API keys and additional configurations.
- Filter and return the configurations based on provided filters.

model_api_key_map will default to `{"gpt-4": "OPENAI_API_KEY", "gpt-3.5-turbo": "OPENAI_API_KEY"}` if none

Arguments:

- `dotenv_file_path` *str, optional* - The path to the .env file. Defaults to None.
- `model_api_key_map` *str/dict, optional* - A dictionary mapping models to their API key configurations. If a string is provided as configuration, it is considered as an environment variable name storing the API key. If a dict is provided, it should contain at least 'api_key_env_var' key, and optionally other API configurations like 'base_url', 'api_type', and 'api_version'. Defaults to a basic map with 'gpt-4' and 'gpt-3.5-turbo' mapped to 'OPENAI_API_KEY'.
- `filter_dict` *dict, optional* - A dictionary containing the models to be loaded. Containing a 'model' key mapped to a set of model names to be loaded. Defaults to None, which loads all found configurations.

Returns:

List[Dict[str, Union[str, Set[str]]]]: A list of configuration dictionaries for each model.

Raises:

- `FileNotFoundError` - If the specified .env file does not exist.
- `TypeError` - If an unsupported type of configuration is provided in model_api_key_map.

retrieve_assistants_by_name

```
def retrieve_assistants_by_name(client: OpenAI, name: str) -> List[Assistant]
```

Return the assistants with the given name from OAI assistant API

Previous
« completion

Next
agent_utils »