Documentation for `utils.py`

A comprehensive guide to understanding the utility functions within `utils.py`. This file contains various helper functions for tasks such as caching, loading prompts, processing files, and setting up a vector store. Let's dive in!

Overview

The `utils.py` file provides a collection of utility functions designed to support various operations within the application. These functions handle tasks ranging from managing a cache to loading prompts from YAML files and processing job descriptions and resumes. By centralizing these functionalities, the file promotes code reusability and maintainability.

`CacheManager` Class

The `CacheManager` class is a singleton class that manages a simple in-memory cache. It's designed to store and retrieve data quickly, reducing the need to recompute or reload frequently accessed information. The class uses a thread lock to ensure thread safety.

Methods:

`__new__(cls)`

The `__new__` method is overridden to implement the singleton pattern. It ensures that only one instance of the `CacheManager` class is created.

Functionality: Creates a single instance of the CacheManager, using a lock to ensure thread safety.

`set(self, key: str, value: Any) -> None`

Sets a variable in the cache with the given key and value.

Parameters:

- `key` (str): The key to identify the cached value.
- `value` (Any): The value to be stored in the cache.

Example:

If you want to store the user's name in the cache:

```
cache_manager.set("user_name", "John Doe")
```

`get(self, key: str, default=None) -> Any`

Retrieves a variable from the cache based on the given key. If the key is not found, it returns the default value.

Parameters:

- `key` (str): The key to identify the cached value.
- `default` (Any, optional): The default value to return if the key is not found. Defaults to `None`.

Example:

To retrieve the user's name from the cache:

```
user_name = cache_manager.get("user_name", "Guest")
```

`has(self, key: str) -> bool`

Checks if a variable exists in the cache based on the given key.

Parameters:

- `key` (str): The key to check for in the cache.

Example:

To check if the user's name is in the cache:

```
if cache_manager.has("user_name"): ...
```

`clear(self, key: str = None) -> None`

Clears a specific key or the entire cache.

Parameters:

- `key` (str, optional): The key to clear from the cache. If `None`, the entire cache is cleared. Defaults to `None`.

`key` (str, optional): The key to clear from the cache. If `None`, the entire cache is cleared. Defaults to `None`.

Example:

To clear the user's name from the cache:

```
cache_manager.clear("user_name")
```

To clear the entire cache:

```
cache_manager.clear()
```

`append_to_list(self, key: str, value: Any) -> bool`

Appends a value to a list in the cache. If the list doesn't exist, it creates a new list with the given value.

Parameters:

`key` (str): The key to identify the list in the cache.
`value` (Any): The value to append to the list.

Returns: `True` if the value was successfully appended, `False` otherwise.

Example:

To append a new skill to the user's skills list:

```
cache_manager.append_to_list("user_skills", "Python")
```

`remove_from_list(self, key: str, value: Any) -> bool`

Removes a value from a list in the cache.

Parameters:

`key` (str): The key to identify the list in the cache.
`value` (Any): The value to remove from the list.

Returns: `True` if the value was successfully removed, `False` otherwise.

Example:

To remove a skill from the user's skills list:

```
cache_manager.remove_from_list("user_skills", "Java")
```

Function: `load_prompts(path: Path) -> dict`

Loads prompts from a YAML file specified by the given path.

Parameters:

`path` (Path): The path to the YAML file containing the prompts.

Returns: A dictionary containing the loaded prompts.

Example:

To load prompts from a file named `prompts.yaml`:

```
prompts = load_prompts(Path("prompts.yaml"))
```

Function: `process_directory(directory_path, file_content)`

Processes files within a specified directory based on the file content type ("job_description" or "resume").

Parameters:

`directory_path` (str): The path to the directory containing the files.
`file_content` (str): Specifies whether to process job descriptions (".txt" files) or resumes (".pdf" files).

Returns: A list of dictionaries containing processed file information.

Example for Job Descriptions:

```
job_descriptions = process_directory("/path/to/job_descriptions", "job_description")
```

This will return a list where each element contains the name and content of each .txt file in the directory.

Example for Resumes:

Example for Resumes:

```
resumes = process_directory("/path/to/resumes", "resume")
```

This will return a list of Document objects, each representing a processed .pdf file, with content and metadata.

Function: `flatten(all_rankings: Dict[AnyStr, List[ResumeFeedback]], jobs: List[Dict[AnyStr, AnyStr]])`

Flattens resume feedback rankings and job descriptions into a single string for easy readability and processing.

Parameters:

- `all_rankings` (Dict[AnyStr, List[ResumeFeedback]]): A dictionary containing resume rankings for each job.
- `jobs` (List[Dict[AnyStr, AnyStr]]): A list of dictionaries, where each dictionary represents a job with its name and content.

Returns: A flattened string containing job descriptions and resume rankings.

Example:

To flatten the rankings and job descriptions:

```
flattened_string = flatten(all_rankings, jobs)
```

Function: `setup_vector_store(cache_manager: CacheManager)`

Sets up a Chroma vector store for storing and retrieving resume embeddings. It uses the `HuggingFaceInferenceAPIEmbeddings` model for generating embeddings and caches both the embedding model and the vector store for efficiency.

Parameters:

- `cache_manager` (CacheManager): An instance of the `CacheManager` class for caching the embedding model and vector store.

Returns: The Chroma vector store instance.

Example:

To set up the vector store:

```
vector_store = setup_vector_store(cache_manager)
```

Function: `process_txt(txt_file)`

Processes an individual TXT file to extract job description content.

Parameters:

- `txt_file`: The TXT file to be processed.

Returns: A list containing a dictionary with the file name and content.

Example:

To process a TXT file:

```
job_descriptions = process_txt(txt_file)
```

Function: `process_pdfs(pdf_files)`

Processes multiple PDF files to extract text content.

Parameters:

- `pdf_files`: A list of PDF files to be processed.

Returns: A list of `Document` objects, each containing the text content and metadata of a PDF file.

Example:

To process a list of PDF files:

```
documents = process_pdfs(pdf_files)
```