Flashcard Generator

Release 1.0.0

Christina Schlager

Sep 20, 2024

TABLE OF CONTENTS:

CHAPTER

ONE

DATA PREPARATION

This section covers functions related to preparing data for the RAG pipeline.

tools.pipeline.**load_wikipedia_dataset**(*language: str = 'simple'*, *date: str = '20220301'*) → Dataset

Load the Wikipedia dataset from the Hugging Face Hub. This function loads the dataset specified by language and date.

Hugging Face Dataset URL: https://huggingface.co/datasets/wikipedia

Parameters

- **language** – Language version of Wikipedia to load. Defaults to 'simple' for Simple English.
- **date** – The snapshot date of the Wikipedia dataset to load. Format: YYYYMMDD. Defaults to "20220301".

Returns

A dataset object that allows accessing the data as required.

tools.pipeline.**process_documents**(*data: Dict[str, Any]*, *chunk_size: int = 5000*, *chunk_overlap: int = 100*, *min_valid_length: int = 200*) → Tuple[List[Any], List[int], int, int]

Process a dataset of documents by splitting each document into chunks, filtering these chunks based on a minimum valid length, and annotating them with metadata.

This function uses a RecursiveCharacterTextSplitter to break down documents into manageable chunks. Chunks that meet the minimum length requirement are kept and annotated with metadata from the original document. The function then returns a list of valid chunks and statistics about the processing.

Parameters

- **data** – A dictionary containing a 'train' key with a list of document dictionaries. Each document should have 'text', 'id', 'url', and 'title' keys.
- **chunk_size** – The size of each chunk in characters.
- **chunk_overlap** – The number of characters to overlap between chunks.
- **min_valid_length** – The minimum number of characters a chunk must have to be considered valid.

Returns

A tuple containing four elements: - A list of valid chunks, where each chunk is an object with 'page_content' and 'metadata', which contains ID, url, and title. - A list containing the count of chunks created from each document. - Total number of chunks created across all documents. - Total number of valid chunks that met the length requirement.

CHAPTER

TWO

DATA STORAGE

This section explains how data is stored in ChromaDB for efficient retrieval and processing.

tools.pipeline.**load_vectorstore_nomic**(*path: str = './chroma_db_nomic'*, *collection_name: str = 'opensource_rag_wikipedia'*) → Tuple[Chroma, Any]

Loads a vector store configured to use the Nomic embeddings with a specified collection.

This function initializes an embeddings object and a persistent client, retrieves the specified collection from the database, and finally initializes and returns a *Chroma* vector store object configured with the Nomic embeddings.

Parameters

- **path** – The file path where the vector store data is located. Defaults to "./chroma_db_nomic".
- **collection** – The name of the collection to be used in the vector store.

Defaults to "opensource_rag_wikipedia".

Returns

A configured instance of the *Chroma* vector store.

tools.pipeline.**load_vectorstore_openai**(*path: str = './chroma_db_openai'*, *collection_name: str = 'openai_rag_chroma_wikipedia'*) → Tuple[Chroma, Any]

Loads a vector store configured to use OpenAI embeddings with a specified collection.

This function initializes an OpenAI embeddings object and a persistent client, retrieves the specified collection from the database, and initializes a *Chroma* vector store object configured with the OpenAI embeddings.

Parameters

- **path** – The file path where the vector store data is located. Defaults to "./chroma_db_openai".
- **collection_name** – The name of the collection to be used in the vector store. Defaults to "openai_rag_chroma_wikipedia".

Returns

A configured instance of the *Chroma* vector store containing the documents and the *OpenAIEmbeddings* used.

tools.pipeline.**setup_vectorstore_nomic**(*texts: List[Any]*, *path: str = './chroma_db_nomic'*, *collection_name: str = 'opensource_rag_wikipedia'*) → Tuple[Chroma, Any]

Sets up a vector store using Nomic embeddings (model "nomic-embed-text-v1.5") with documents provided. The function initializes Nomic embeddings, sets up a persistent client, ensures the collection exists (or creates it), and populates a vector store with the documents.

Parameters

- **texts** – A list of documents (texts) to be stored in the vector store.
- **path** – The file path where the vector store data will be located. Defaults to "./chroma_db_nomic".
- **collection** – The name of the collection to be used or created in the vector store. Defaults to "opensource_rag_wikipedia".

Returns

A configured instance of the *Chroma* vector store containing the documents and 'Nomic Embeddings' used.

tools.pipeline.**setup_vectorstore_openai**(*texts: List[Any]*, *path: str = './chroma_db_openai'*, *collection_name: str = 'openai_rag_chroma_wikipedia'*) → Tuple[Chroma, Any]

Sets up a vector store using OpenAI embeddings (model "text-embedding-3-large") with the provided documents. The function initializes OpenAI embeddings, sets up a persistent client, checks for the existence of the specified collection (or creates it), and populates a vector store with the documents.

Parameters

- **texts** – A list of documents (texts) to be stored in the vector store.
- **path** – The file path where the vector store data will be located. Defaults to "./chroma_db_openai".
- **collection_name** – The name of the collection to be used or created in the vector store. Defaults to "openai_rag_chroma_wikipedia".

Returns

A tuple containing a configured instance of the *Chroma* vector store and the *OpenAIEmbeddings* used.

CHAPTER

THREE

RETRIEVER

This section outlines the functionality of the retriever component, responsible for fetching relevant embeddings based on query vectors.

tools.pipeline.**format_documents**(*docs: List[Any]*) → List[Dict[str, Any]]

Formats a list of document objects into a standardized dictionary format.

This function iterates through each document in the provided list, extracting relevant metadata namely, page content, ID, title, and source (URL). Each document is then transformed into a dictionary with keys for 'context', 'id', 'title', and 'source', based on the respective properties and metadata from the document objects.

Parameters

docs – A list of document objects, each containing content and metadata.

Returns

A list of dictionaries where each dictionary represents a document with formatted data.

tools.pipeline.**retrieve_documents**(*topic: str*, *vectorstore: Chroma*, *threshold: float = 0.5*) → List[Any]

Retrieves documents from the vector store based on a similarity score threshold.

This function initializes a retriever with the specified threshold for similarity scoring, using the 'similarity_score_threshold' search type. It then invokes the retriever with a given topic to fetch documents that meet or exceed the specified similarity score threshold.

Parameters

- **topic** – The topic query as a string which is used to retrieve relevant documents.
- **vectorstore** – The vector store instance which contains the document embeddings and provides the retrieval mechanism.
- **threshold** – The minimum similarity score threshold. Documents with a similarity score above this threshold will be considered relevant.

Returns

Returns a list of documents or relevant entries that match the topic based on the specified similarity score threshold.

CHAPTER

FOUR

GENERATION

This section details the generation component, which focuses on producing question-answer pairs based on the retrieved documents.

tools.pipeline.**calculate_num_pairs**(*text: str*) → int

Calculates the number of sections a text can be divided into using the *split_text_into_sections* function.

This function first divides the text into sections based on predefined criteria in *split_text_into_sections*. It then calculates the number of these sections (pairs), returning the total count.

Parameters
text – The text for which to calculate the number of dividable sections.

Returns
The number of sections into which the text was divided.

tools.pipeline.**generate_question_answer_pairs_open_ai_json**(*topic: str*, *vectorstore: Chroma*, *threshold: float = 0.5*) → str

Generates unique question-answer pairs from documents retrieved based on a topic.

This function first retrieves documents related to the specified topic with a similarity score above the given threshold. It then formats these documents and splits them into smaller sections suitable for generating question-answer pairs. Each section is processed to produce unique and contextually relevant question-answer pairs, using the model 'gpt-4-turbo' via the OpenAI API.

Dependencies:

- **OpenAI API**: Used to generate question-answer pairs using a model-based approach.
- **Model**: 'gpt-4-turbo'. This model is specified for its ability to generate detailed and contextually accurate question-answer pairs.

Parameters

- **topic** – The topic query to fetch related documents.
- **vectorstore** – The vector store instance used to retrieve document embeddings.
- **threshold** – The similarity score threshold for document retrieval. Defaults to 0.5.

Returns
A JSON string containing a list of unique question-answer pairs generated from the documents.

tools.pipeline.**generate_question_answer_pairs_open_source_json**(*topic: str*, *vectorstore: Chroma*, *threshold: float = 0.5*) → str

Generates unique question-answer pairs from documents retrieved based on a topic.

This function first retrieves documents related to the specified topic with a similarity score above the given threshold. It then formats these documents and splits them into smaller sections suitable for generating question-answer pairs. Each section is processed to produce unique and contextually relevant question-answer pairs, using the model 'meta-llama/Meta-Llama-3.1-70B-Instruct-Turbo' via the Together API.

Dependencies:

- **Together API**: Used to generate question-answer pairs using a model-based approach.
- **Model**: 'meta-llama/Meta-Llama-3.1-70B-Instruct-Turbo'. This model is specified to generate question-answer pairs that are unique and tailored to the educational content based on the document's context.

Parameters

- **topic** – The topic query to fetch related documents.
- **vectorstore** – The vector store instance used to retrieve document embeddings.
- **threshold** – The similarity score threshold for document retrieval. Defaults to 0.5.

Returns

A JSON string containing a list of unique question-answer pairs generated from the documents.

tools.pipeline.**split_text_into_sections**(*text: str*, *max_length: int = 150*, *min_length: int = 50*) → List[str]

Splits a given text into sections based on specified maximum and minimum lengths.

The function divides the input text into paragraphs, and sequentially adds paragraphs to a current section until adding another paragraph would exceed the maximum length. If the current section meets the minimum length requirement, it is saved as a separate section. This continues until all paragraphs are processed.

Parameters

- **text** – The text to be split into sections.
- **max_length** – The maximum length of each section in characters, defaults to 150.
- **min_length** – The minimum length a section must have to be considered valid, defaults to 50.

Returns

A list of text sections that meet the length requirements.

CHAPTER

FIVE

EVALUATION

This section covers the evaluation methods for the RAG pipeline, utilizing Ragas, DeepEval, and Haystack metrics to assess performance and effectiveness.

tools.pipeline.**deepeval_evaluate_data**(*deepeval_data: List[LLMTestCase]*, *ground_truth: bool = True*, *model: str = 'gpt-4'*) → None

Evaluates a prepared dataset using a suite of DeepEval's metrics designed to assess the performance of language models. It outputs a structured report summarizing the results of each metric evaluation based on whether ground truth is considered.

Parameters

- **deepeval_data** – The dataset containing test cases prepared for evaluation.
- **ground_truth** – Specifies whether metrics that require ground truth data should be included.
- **model** – The name of the model to be used for evaluation, defaults to "gpt-4".

Returns

Outputs a structured report summarizing the metric evaluations.

tools.pipeline.**deepeval_prepare_data**(*dataset: DataFrame*, *ground_truth: bool = True*) → List[LLMTestCase]

Prepares a dataset for using DeepEval evaluation metrics by creating test cases for each entry in the dataset. Each test case is structured to include a question, the actual and expected answers (if ground truth is provided), and the retrieval context.

Parameters

- **dataset** – The dataset to be prepared, typically containing 'question', 'answer', and 'contexts' columns, and optionally a 'ground_truth' column if ground_truth is True.
- **ground_truth** – Specifies whether the 'ground_truth' column should be included in the test cases.

Returns

A list of test cases, each designed for evaluation purposes.

tools.pipeline.**evaluate_retriever**(*topic: str*, *vectorstore: Chroma*, *k: int = 10*) → DataFrame

Evaluates the performance of a vector store's retrieval capabilities by conducting searches using both cosine similarity and cosine distance metrics, and then compares the results.

This function retrieves the top 'k' documents based on the cosine similarity and cosine distance metrics for a given topic. It constructs a DataFrame containing the results, including each document's ID, URL, context, similarity score, and distance score.

Parameters

- **topic** – The topic string based on which the documents are retrieved.
- **vectorstore** – An instance of a vector store configured with embeddings used for retrieving documents.
- **k** – The number of top documents to retrieve, defaults to 10.

Returns

A pandas DataFrame containing the document details and their respective retrieval scores.

tools.pipeline.**haystack_evaluate_data**(*data: List[Dict[str, Any]]*, *ground_truth: bool = True*) → DataFrame

Evaluates the prepared dataset by running it through a series of Haystack's evaluation metrics within a pipeline. Metrics include context relevance, faithfulness of the answers, and optionally semantic answer similarity if ground_truth is True.

Parameters

- **data** – A list of dictionaries where each dictionary contains 'questions', 'contexts', 'predicted_answers', and optionally 'ground_truth_answers' if ground_truth is True.
- **ground_truth** – A flag to determine whether semantic similarity evaluation should be performed using the ground truth answers provided in the data.

Returns

A pandas DataFrame containing the evaluation results for each data point, including scores for context relevance, faithfulness, and optionally semantic similarity along with the overall scores for these metrics.

tools.pipeline.**haystack_prepare_data**(*dataset: DataFrame*, *ground_truth: bool = True*) → List[Dict[str, Any]]

Prepares a dataset for using Haystack's evaluation metrics by formatting the input data into a specific structure.

Parameters

- **dataset** – The dataset to be prepared, typically containing 'question', 'answer', and 'contexts' columns, and optionally a 'ground_truth' column if ground_truth is True.
- **ground_truth** – Specifies whether metrics that require ground truth data should be included.

Returns

A list of dictionaries, each containing data for a single instance. Each dictionary includes the question, predicted answers, and context(s). If *ground_truth* is True, it also includes ground truth answers.

tools.pipeline.**ragas_evaluate_data**(*data: Dataset*, *ChatOpenAI_model_name: str = 'gpt-4'*, *temperature: float = 0*, *ground_truth: bool = True*) → Dict[str, Any]

Evaluates the prepared dataset using various evaluadtion metrics with a specified ChatGPT model as additional large language model. It wraps the ChatGPT model in a LangchainLLMWrapper and calculates different metrics based on whether ground truth is considered.

Parameters

- **data** – The prepared dataset to evaluate.
- **ChatOpenAI_model_name** – The model name of the ChatGPT to use for evaluation.
- **temperature** – The sampling temperature to use in the model inference, defaults to 0.
- **ground_truth** – Specifies whether metrics that require ground truth data should be included.

Returns
The evaluation results with specified metrics.

tools.pipeline.**ragas_prepare_data**(*dataset: DataFrame*, *ground_truth: bool = True*) → Dataset

Prepares a dataset for use of the Ragas evaluation metrics by transforming and ensuring proper formatting of its columns. This function resets the dataset index, ensures 'contexts' are lists of strings, optionally adds a 'ground_truth' field, removes unnecessary columns, and casts the dataset to specified features.

Parameters

- **dataset** – The dataset to prepare, which should contain at least the 'question', 'answer', 'contexts' and 'source' field.
- **ground_truth** – If True, includes the 'ground_truth' field in the final dataset.

Returns
The transformed dataset with the specified features format.

Raises
ValueError – If the data type of the 'contexts' field is neither a string nor a list.

CHAPTER

SIX

UTILS

This section includes various utility functions that support the operation and management of the RAG pipeline.

tools.pipeline.**load_qa_pairs**(*folder_name: str*, *topic: str*, *pipeline_name: str*, *content: str = 'generated_qas_gt'*) → DataFrame

Loads question-answer pairs from a CSV file into a pandas DataFrame. The CSV file must be named according to a specific naming convention and located in the specified folder.

Parameters

- **folder_name** – The folder where the CSV file is stored.
- **topic** – The main topic of the question-answer pairs, used in the file name. The spaces in the topic will be replaced with underscores and converted to lowercase.
- **pipeline_name** – The name of the pipeline used to generate the question-answer pairs, used in the file name.
- **content** – An optional descriptor for the file name, defaults to 'generated_qas_gt'.

Returns

A pandas DataFrame containing the loaded question-answer pairs. The 'contexts' column is processed to remove square brackets and single quotes.

tools.pipeline.**prepare_dataset**(*flashcard_set: str*) → Dataset

Converts a JSON string of flashcard data into a structured dataset.

Parameters

flashcard_set – A JSON string that represents a set of flashcards. Each flashcard is a dictionary with keys 'question', 'answer', 'context', and 'source'.

Returns

A dataset object with structured fields for questions, answers, contexts, and sources. The dataset has features defined for each field to ensure correct data types.

tools.pipeline.**save_qa_pairs**(*dataset: Any*, *folder_name: str*, *topic: str*, *pipeline_name: str*, *content: str = 'generated_qas'*) → None

Saves the provided dataset as a CSV file formatted for question-answer pairs, organized by topic, pipeline name and content.

Parameters

- **dataset** – The dataset containing the question-answer pairs. It can be a pandas DataFrame or any object that has a *to_pandas()* method.
- **folder_name** – The name of the folder where the CSV file will be saved.
- **topic** – The main topic of the question-answer pairs, used in the file name. The spaces in the topic will be replaced with underscores and converted to lowercase.

- **pipeline_name** – The name of the pipeline used to generate the question-answer pairs.
- **content** – An optional descriptor that precedes the file naming convention. Defaults to 'generated_qas'.

Returns
None

Raises
Exception – If the dataset cannot be converted to a pandas DataFrame.

Upon successful saving of the file, this function prints "File saved as CSV." to the standard output. The file is saved with a semicolon as the delimiter and commas as the decimal separator.

PYTHON MODULE INDEX

t

INDEX