Llettuce

Release 0.1

Reza Omidvar, James Mitchell-White, Grazziela Figueredo, Philip

Jul 18, 2024

CONTENTS:

Llettuce is an application for medical researchers that matches the informal medicine names supplied by the user to concepts in the Observational Health Data Sciences and Informatics standardised vocabularies.

The application can be used as an API, or run with a graphical user interface (GUI).

Note: This project is under active development

CHAPTER

ONE

API DIAGRAM

This is the rough process that the Llettuce API follows. Subject to change

1.1 Usage

1.1.1 Installation

To use Llettuce, you must first clone the repository

```
$ git clone <url>
```

Then install the dependencies, either using pip

```
$ pip install -r requirements.txt
```

or conda

```
$ conda create -f environment.yml
```

1.1.2 Connecting to a database

Llettuce works by querying a database with the OMOP schema, so you should have access to one. Your database access credentials should be kept in *.env*. An example of the format can be found in */Carrot-Assistant/.env.example*:

```
DB_HOST="Your Host"
DB_USER="Your User"
DB_PASSWORD="Your Password"
DB_NAME="Your Database Name"
DB_PORT="Your Port, default is 5432"
DB_SCHEMA="Your Schema"
```

1.1.3 Running the API

The simplest way to get a formal name from an informal name is to use the API. To start a Llettuce server:

```
$ uvicorn app:app --host 0.0.0.0 --port 8000
```

Then a response can be produced using curl

```
$ curl -X POST "http://127.0.0.1:8000/run" -H "Content-Type: application/json" -d '{
↪"informal_name": "Betnovate Scalp Application"}'
```

1.1.4 Running the GUI

An alternative to using curl to get Llettuce output is to first start the API as before, and then start the GUI

```
$ streamlit run ui.py
```

The GUI makes calls to the API equivalent to the curl request above.

1.2 API Reference

This page contains auto-generated API reference documentation[1].

1.2.1 ui

Attributes

`informal_name`
`result_stream`

Functions

`stream_message`(→ None)	Stream a message to the user, rendering HTML
`capitalize_words`(→ str)	Capitalize each word in a string
`make_api_call`(→ sseclient.SSEClient)	Make a call to the Llettuce API to retrieve OMOP concepts.
`display_concept_info`(→ None)	Display the concept information.

[1] Created with sphinx-autoapi

Module Contents

`ui.`**`informal_name:  streamlit.text_input`**

`ui.`**`stream_message`**`(`*message: str*`)` → None

Stream a message to the user, rendering HTML

Parameters

message: str
The message to stream

`ui.`**`capitalize_words`**`(`*s: str*`)` → str

Capitalize each word in a string

Parameters

s: str
The string to capitalize

Returns

str
The capitalized string

`ui.`**`make_api_call`**`(`*name: str*`)` → sseclient.SSEClient

Make a call to the Llettuce API to retrieve OMOP concepts.

Parameters

name: str
The informal name to send to the API

Returns

sseclient.SSEClient
The stream of events from the API

`ui.`**`display_concept_info`**`(`*concept: dict*`)` → None

Display the concept information. An OMOP concept is formatted as HTML to be streamed to the user.

Parameters

concept: dict
The concept information

`ui.result_stream: sseclient.SSEClient`

1.2.2 app

Attributes

`logger`	
`app`	

Classes

`InformalNameRequest`	This class is used to represent the request for the informal name of a medication

Functions

`generate_events`(→ collections.abc.AsyncGenerator[str])	Generate LLM output and OMOP results for an informal medication name
`run_pipeline`(→ sse_starlette.sse.EventSourceRespons	Call generate_events to run the pipeline

Module Contents

`app.logger`

`app.app`

`class app.InformalNameRequest`

Bases: `pydantic.BaseModel`

This class is used to represent the request for the informal name of a medication

`informal_name: str = None`

`async app.generate_events`(*request: InformalNameRequest*) → collections.abc.AsyncGenerator[str]

Generate LLM output and OMOP results for an informal medication name

The first event is the reply from the LLM The second event fetches relevant concepts from the OMOP database using the LLM output

The function yields results as they become available, allowing for real-time streaming.

Parameters

request: InformalNameRequest
The request containing the informal name of the medication

Yields

str
JSON encoded strings of the event results. Two types are yielded: 1. "llm_output": The result from the language model processing. 2. "omop_output": The result from the OMOP database matching.

Each yielded string has the format: {

"event": "<event_type>", "data": <event_data>

}

`async app.run_pipeline(`*request: InformalNameRequest*`)` → sse_starlette.sse.EventSourceResponse

Call generate_events to run the pipeline

Parameters

request: InformalNameRequest
The request containing the informal name of the medication

Returns

EventSourceResponse
The response containing the events

1.2.3 utils

Functions

`get_informal_name`(→ str)	Gets the informal name from the response

Module Contents

`utils.get_informal_name(`*json_arr: str*`)` → str

Gets the informal name from the response

Parameters

json_arr: str
The json response

Returns

str
The informal name

1.2.4 models

Functions

`get_model`(→ object)	Get the model

Module Contents

`models.get_model`(*model_name: str*, *temperature: float = 0.7*, *logger: logging.Logger | None = None*) → object
Get the model

Parameters

model_name: str
The name of the model

temperature: float
The temperature for the model

logger: logging.Logger|None
The logger for the model

Returns

object
The model

1.2.5 prompt

Classes

`Prompts`	This class is used to generate prompts for the models.

Module Contents

class prompt.**Prompts**(*model_name: str*, *prompt_type: str | None = 'simple'*)

This class is used to generate prompts for the models.

get_prompt() → haystack.components.builders.PromptBuilder

Get the prompt for the model

Returns

PromptBuilder
The prompt for the model

_simple_prompt() → haystack.components.builders.PromptBuilder

Get a simple prompt

Returns

PromptBuilder
The simple prompt

1.2.6 pipeline

Classes

`llm_pipeline`	This class is used to generate a pipeline for the model

Module Contents

class pipeline.**llm_pipeline**(*opt: argparse.Namespace*, *logger: logging.Logger | None = None*)

This class is used to generate a pipeline for the model

get_simple_assistant() → haystack.Pipeline

Get a simple assistant pipeline

Returns

Pipeline
The pipeline for the assistant

1.2.7 assistant

Attributes

`opt`

Functions

`run`(→ dict | None)	Run the LLM assistant to suggest a formal drug name for an informal medicine name

Module Contents

`assistant.run`(*opt: argparse.Namespace = None*, *informal_name: str = None*, *logger: utils.logging_utils.Logger | None = None*) → dict | None

Run the LLM assistant to suggest a formal drug name for an informal medicine name

Parameters

opt: argparse.Namespace
The options for the assistant

informal_name: str
The informal name of the medication

logger: Logger
The logger to use

Returns

dict or None
A dictionary containing the assistant's output - 'reply': str, the formal name suggested by the assistant - 'meta': dict, metadata from an LLM Generator Returns None if no informal_name is provided

`assistant.opt`

1.2.8 preprocess

Functions

`preprocess_search_term`(→ str)	Preprocess a search term for use in a full-text search query.

Module Contents

`preprocess.`**`preprocess_search_term`**(*term*) → str

Preprocess a search term for use in a full-text search query.

This function performs the following operations:

1. Converts the input term to lowercase.
2. Splits the term into individual words.
3. Removes common stop words.
4. Joins the remaining words with ' | ' for use in PostgreSQL's to_tsquery function.

Args:
term (str): The original search term.

Returns:
str: A preprocessed string ready for use in a full-text search query.

Example:

```
>>> preprocess_search_term("The quick brown fox")
"quick | brown | fox"
```

1.2.9 OMOP_match

Classes

`OMOPMatcher`	OMOPMatcher class to calculate best OMOP matches for a given search term

Functions

`run`(opt, search_term, logger)

Module Contents

`class` `OMOP_match.`**`OMOPMatcher`**(*logger: utils.logging_utils.Logger | None = None*)

OMOPMatcher class to calculate best OMOP matches for a given search term

`close`()

Close the engine connection.

`calculate_best_matches`(*search_terms*, *vocabulary_id=None*, *concept_ancestor='y'*, *concept_relationship='y'*, *concept_synonym='y'*, *search_threshold=None*, *max_separation_descendant=1*, *max_separation_ancestor=1*)

Calculate best OMOP matches for a given search term

fetch_OMOP_concepts(*search_term*, *vocabulary_id*, *concept_ancestor*, *concept_relationship*, *concept_synonym*, *search_threshold*, *max_separation_descendant*, *max_separation_ancestor*)

Fetch OMOP concepts for a given search term

fetch_concept_ancestor(*concept_id*, *max_separation_descendant*, *max_separation_ancestor*)

Fetch concept ancestor for a given concept_id

fetch_concept_relationship(*concept_id*)

Fetch concept relationship for a given concept_id

OMOP_match.**run**(*opt*, *search_term*, *logger*)

1.2.10 base_options

Classes

`BaseOptions`	This class defines options used during all types of experiments.

Module Contents

class base_options.**BaseOptions**

This class defines options used during all types of experiments. It also implements several helper functions such as parsing, printing, and saving the options.

initialize() → None

Initializes the BaseOptions class

Parameters

None

Returns

None

parse() → argparse.Namespace

Parses the arguments passed to the script

Parameters

None

Returns

opt: argparse.Namespace
The parsed arguments

_print(*args: Dict*) → None

Prints the arguments passed to the script

Parameters

args: dict
The arguments to print

Returns

None

1.2.11 logging_utils

Classes

`Logger`	logger preparation

Module Contents

class `logging_utils.`**Logger**(*logging_level='INFO'*, *console_logger=True*, *multi_module=True*)

Bases: `object`

logger preparation

Parameters

log_dir: string
path to the log directory

logging_level: string
required Level of logging. INFO, WARNING or ERROR can be selected. Default to 'INFO'

console_logger: bool
flag if console_logger is required. Default to False

Returns

logger: logging.Logger
    logger object

`_make_level()`

`make_logger()`

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

INDEX