Conversational Task Agent

Francisco Barreiras nº67208
Francisco Silva nº57824
Tomás Carvalho nº67209

NOVA SST

Abstract. This project involves developing a task-oriented bot designed to assist users through complex, multi-step tasks like baking a birthday cake or repairing a car scratch, with a unique capability to dynamically adjust instructions based on the user's available resources and tools. Should a user encounter a situation where they lack a necessary ingredient or tool, the taskbot is engineered to reconfigure the task plan on-the-fly, offering alternative solutions to ensure the task can still be completed successfully.

Keywords: task-oriented bot · dynamically adjust · alternative solutions.

1 Introduction

This project consists in the development of a task-oriented bot with the main goal of helping the user to perform a task like baking a cake, in case of adversity (like not having the required ingredients for cooking the recipe), it should provide alternative paths. For the implementation of such project, we have used OpenSearch due to its capabilities in realms of full-text search and analytics, and its ability to scale to billion size documents.

2 Architecture

The architecture of the system is comprised by 3 modules and a main python notebook, used to control the indexing ans search modules.
The Indexing.py module's purpose is to handle the index operations, might these be to calculate the embeddings and interact with the pickle files used to store the embeddings, to create the index and mappings or to parse and store the recipes from the base file to OpenSearch.
The Search Builder API is meant to be used as a tool to easily and in an iterative manner build simple or complex search OpenSearch queries. This module avoids repetitive code, also as a way to ensure good application scaling and coding practices and maintaining a developer-friendly interface.
The Search.py module is simply an abstraction of the SearchBuilder.py module, made to simplify the search operations for a user. It's easily scalable thanks to the SearchBuilder layer and was made to be used as a simple interface for the system.

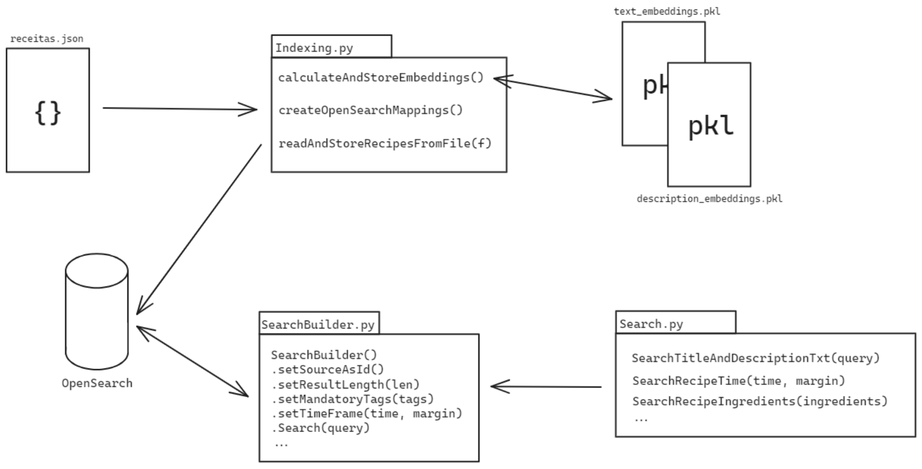
receitas.json
{}
Indexing.py
calculateAndStoreEmbeddings()
createOpenSearchMappings()
readAndStoreRecipesFromFile(f)
text_embeddings.pkl
pkl
pkl
description_embeddings.pkl
OpenSearch
SearchBuilder.py
SearchBuilder()
.setSourceAsId()
.setResultLength(len)
.setMandatoryTags(tags)
.setTimeFrame(time, margin)
.Search(query)
...
Search.py
SearchTitleAndDescriptionTxt(query)
SearchRecipeTime(time, margin)
SearchRecipeIngredients(ingredients)
...

Fig. 1. Architecture of the system.

3 Algorithms and Implementation

3.1 Mappings and reasons behind them

Mappings define how data is stored, indexed, and searched within an index. They are essentially the schema definition, specifying the data types for each field in your documents and how those fields should be treated by OpenSearch. By specifying the data type for each field (such as text, keyword, date, integer, etc.), OpenSearch can store data more efficiently, improving both storage usage and search performance. In our work we implemented the following mappings:

– **doc_id: keyword** - used for fields that to be indexed as a whole, without breaking them down into separate searchable terms ,in this specific case, the id of the document.

– **tags: keyword** - used to tag each document according to its 'diets', 'courses' and 'cuisines'. Will be used to search these tags in a deterministic way, meaning the tag is to be exactly the ones used in the initial document, useful to filter recipes

– **title: text** - chosen for fields that contain full text, such as descriptions, article content, or any other form of unstructured text that benefits from full-text search capabilities.

– **title_embedding: knn vector** - performs searches to find documents that are "closest" to a given vector, created specifically for title embedding representations.

- **description: text** - enables complex search queries, including matching on individual terms within the field.
- **description_embedding: knn vector** - used to find the "closest" description embedding vectors to the given query.
- **ingredients: text** - used to perform searches by similarity, meaning a score will be calculated to match the documents in which its ingredients best match the given query.
- **time: integer** - used to perform queries based on numeric ranges, such as finding all documents where a certain numeric value falls between two bounds. Used to find recipes that take a given time with a given range
- **difficultyLevel: keyword** - used to search for recipes that have a specific difficulty, its a keyword in order to be used as a filter.
- **contents: text** - a non-indexable field, used to store recipe information that is not to be searched by, but can be retrieved.

3.2 Supported Searches

OpenSearch excels in conducting searches within datasets, offering a robust set of features tailored for both simple and advanced search queries. This capability is foundational to its utility as a search and analytics engine.

We opted to join the recipe name and description when searching a recipe by query to have the best possible results, since we considered that some searches might not appear on a recipe's name, but rather be specified in the description.

Since we implemented the SearchBuilder module, it's possible to make a broad amount of queries just by assembling the builder as pleased, this means that any combination of parameters can be used to search for a recipe. In order to give use-case examples of our SearchBuilder as well as a straight forward way to test the system, we implemented 9 methods in the Search module that exemplify possible and useful queries for a searching system like the one implemented

- **SearchTitleAndDescriptionTxt** - provides the capability of searching for a recipe by title and description by text matching.
- **SearchRecipeTime** - provides the capability of searching for recipe by its time within a range, not considering the name or description.

- **SearchRecipeNameTime** - provides the capability of searching for recipe by its time within a range, having in account text matching by the name and description.
- **SearchRecipeIngredients** - provides the capability of searching for a recipe by ingredients.
- **SearchRecipeExcludeIngredients** - provides the capability of searching for a recipe that does not have the given ingredients.
- **SearchRecipeDifficulty** - provides the capability of searching for a recipe by its difficulty.
- **SearchRecipeNameDifficulty** - provides the capability of searching for a recipe by its difficulty, matching the query to its name and description.
- **SearchTitleEmbeddings** - provides the capability of searching for a recipe by title embeddings.
- **SearchDescriptionEmbeddings** - provides the capability of searching for a recipe by description embedding.

4 Dataset Description

This dataset is comprised of 993 elements (recipes), each one of them is constituted by several fields but in our work we will work only with the following:

- displayName - also known as the title of the recipe.
- description - the overview of the recipe and how you can use it.
- totalTimeMinutes - the time it takes to prepare it.
- ingredients - a list of the requirements for the recipe.
- difficultyLevel - how hard it is to prepare the dish.
- cuisines - a list of the kind of gastronomies in which this recipe is inserted in.
- courses - a list of the kind of meals in which it is usually eaten.
- diets - a list of the diets for which this recipe is fit for.

5 References

References

1. OpenSearch. "OpenSearch Compound Query DSL - Bool." https://opensearch.org/docs/latest/query-dsl/compound/bool/ (Accessed: April 8, 2024).
2. OpenSearch. "OpenSearch Full-text Query DSL - Match." https://opensearch.org/docs/latest/query-dsl/full-text/match/ (Accessed: April 8, 2024).
3. OpenSearch. "OpenSearch Full-text Query DSL - Multi-match." https://opensearch.org/docs/latest/query-dsl/full-text/multi-match/ (Accessed: April 8, 2024).
4. OpenSearch. "OpenSearch Term Query DSL - Range." https://opensearch.org/docs/latest/query-dsl/term/range/ (Accessed: April 8, 2024).
5. Python. "Python pickle — Python 3 documentation." https://docs.python.org/3/library/pickle.html (Accessed: April 8, 2024).
6. OpenSearch. "OpenSearch Field Types." https://opensearch.org/docs/latest/field-types/ (Accessed: April 8, 2024).