Knowledge-and-Language 2025/2026
Project Report

A Bilingual Recipe Assistant Combining a Knowledge Graph, Lightweight NLP, and LLMs

Miguel Castela[1] , **Miguel Martins**[1]
[1] Universidade de Coimbra
{uc2022212972, uc2022213951}@student.uc.pt

1 Abstract

We present a bilingual recipe assistant that combines a curated recipe Knowledge Graph (KG), intent classification, entity extraction and SPARQL query retrieval. This ends in an LLM generating the summary of the results in the user's language. In this report, we describe the data curation, approach, implementation, and experiments, highlighting strengths, limitations, and future directions.

2 Introduction

Motivation.
Recipe search is often unstructured and specific. Users can query for ingredients, dietary tags, or time constraints. Most systems either rely on pure keyword search or naive large language model answers. We aim to combine structured knowledge graph retrieval with natural language generation, ensuring factual retrieval and usability.

Problem Description. Given a user query in Portuguese or English, the system must: Detect intent; extract and link entities to the KG via fuzzy matching; retrieve candidate results via SPARQL; perform a RAG similarity search over recipe descriptions; generate summaries based on all the information, the user's query and the recent history; present results through an interactive interface.

Goal and Approach Our approach is building the following pipeline: A curated recipe knowledge graph in RDF/Turtle; an intent classifier trained with BERT; an NLP pipeline for entity extraction and fuzzy linking; a FAISS embedding index for recipe descriptions; LLM-based summarization; a React/Vite-based frontend.

3 Related Work

Research on integrating symbolic knowledge representations with neural language models has expanded significantly in recent years, particularly in the context of improving factual accuracy and domain-specific reasoning. A central issue identified by Agrawal et al. [Agrawal *et al.*, 2024] is that Large Language Models (LLMs) often struggle with domain grounding due to limitations in context comprehension, factual knowledge, and reliable recall. This has motivated work on hybrid systems that augment neural architectures with structured knowledge resources such as RDF/OWL Knowledge Graphs (KGs) and external retrieval modules. For this effect RDF encodes information as subject–predicate–object triples, enabling flexible graph-based modelling [KL2, 2025]. RDFS adds schema semantics such as class hierarchies and domain/range constraints, while OWL introduces richer logical constructs including cardinality restrictions, equivalence, and disjointness [KL2, 2025].

In these hybrid systems, SPARQL querying allows precise retrieval of RDF data [KL2, 2025], while either NLP rule based systems or more complex tools such as spaCy [Honnibal *et al.*, 2020] and BERT [Devlin *et al.*, 2019] enable entity recognition, intent classification, and mapping between natural language and the KG entities.

A major line of research addressing LLM factuality is Retrieval-Augmented Generation (RAG). Lewis et al. [Lewis *et al.*, 2020] introduced RAG as a hybrid architecture combining a parametric generator (BART [Lewis *et al.*, 2019]) with non-parametric memory built from Dense Passage Retrieval (DPR) [Karpukhin *et al.*, 2020]. The RAG-Sequence and RAG-Token variants further refine retrieval–generation interactions, with the latter enabling token-level grounding from different retrieved documents [Lewis *et al.*, 2020]. This framework has become a widely adopted strategy for mitigating hallucinations in neural generation.

After researching various approaches to integrating knowledge graphs with LLMs for domain-specific question answering, we identified RecipeRAG [Loesch *et al.*, 2025] as a particularly relevant prior work for our project. RecipeRAG addresses the challenge of answering complex recipe queries by combining structured knowledge graph embeddings with generative modelling. KGEs such as TransE [Bordes *et al.*, 2013], RotatE [Sun *et al.*, 2019], and QuatE [Zhang *et al.*, 2019] encode entities and relations into continuous vector spaces, enabling similarity-based retrieval that respects graph structure. RecipeRAG demonstrates that KGE-driven retrieval outperforms purely text-based similarity when handling multi-criteria recipe search, and further integrates retrieved KG evidence into a generative model to produce personalized recipes. This work highlights the potential of com-

bining symbolic knowledge, embedding-based retrieval, and generative modelling in domain-specific applications such as recipe recommendation.

Given this context, the project proposed in this report builds on these research directions by integrating (i) an RDFS-based KG, (ii) an NLP pipeline using BERT [Devlin *et al.*, 2019] and spaCy [Honnibal *et al.*, 2020] for intent and entity extraction, and (iii) a retrieval step inspired by RAG [Lewis *et al.*, 2020], enriched with structured recipe data. Similar to RecipeRAG [Loesch *et al.*, 2025]. Instead of relying solely on knowledge graph embeddings for retrieval, our approach combines precise SPARQL querying with an embedding similarity search over recipe descriptions. This hybrid architecture aims to leverage the strengths of structured knowledge representation, lightweight NLP techniques, and LLM generation to provide more accurate, context-aware answers to bilingual recipe queries based on both structured and unstructured data.

4 Data Selection

Firstly we searched for a dataset with portuguese recipes, but the ones we found were poorly structured. ([Antelo, 2021]) is a portuguese cuisine dataset that unfortunately did not meet our requirements for structure and completeness.
Thus, we opted to use a portion of the Food.com [Li, 2020] dataset that, in whole, contains over 200,000 recipes,
The dataset was already curated beacuse its based on the website data that contain only recipes with the following information:

Table 1: Fields present in each recipe entry.

Field	Description
`name`	Recipe name
`id`	Unique recipe identifier
`minutes`	Total preparation/cooking time
`contributor_id`	ID of the user who submitted the recipe
`submitted`	Submission date
`tags`	List of categorical tags
`nutrition`	Nutritional information vector
`n_steps`	Number of preparation steps
`steps`	Ordered list of preparation steps
`description`	Short user's recipe description
`ingredients`	List of ingredients
`n_ingredients`	Number of ingredients

However, as stated in the project proposal, we aim to have a bilingual dataset containing recipes in either portuguese or english languages. This is necessary to then be able to make acurate similarity retrievals and SPARQL queries based on user queries in either of these two languages.
This filtering was achieved using two methods

- Firstly, some portuguese recipes were automatically matched using a set of specific keywords defined by hand (for example: "bacalhau", "lisbon", "azores") using regular expressions. In this stage, a pre-trained langid model was also used [Lui and Baldwin, 2012] to classify portuguese recipes by their full scentences, with a probability score of above 0.8. In this stage
- A lot of false negatives were still present from the first step. A pre-trained fasttext LLM (lid.176.bin) [Joulin *et al.*, 2016] was used to predict the language of each token from the data of each recipe. To be labeled as portuguese a token would have to be considered as 'PT' with a confidence > 0.7 and the difference to the second detected language had to be > 0.15.

Both datasets were merged, however there was stil a relatively high rate of false positives, a total of around 1435. This was partially due to a lot of spanish recipes being classified as portuguese on account of the similarity of the two languages. This problem was solved with a pass of the merged dataset through an LLM (chatGPT) [OpenAI, 2025b] to classify the recipes as portuguese or not portuguese.

After removing the false positives 1015 portuguese recipes remained. After this around 9000 random recipes were added to the dataset. To keep portuguese and english recipes we passed the dataset through the fasttext model again and kept only recipes that were classified as english with a confidence > 0.8. This resulted, with the portuguese recipes, in a final dataset of around 10,000 recipes.

5 Approach

5.1 Knowledge Graph Construction

After the final dataset of around 10,000 recipes was obtained, the knowledge graph was gemerated io RDFS format. For the graph generation only the following fields were used as RDFS classes: name, id, minutes, tags, nutrition, n_steps, steps, ingredients, n_ingredients. This was the decided structured considered most relevant to build the proposed chatbot.

5.2 Intent Training

The first step to analise the user's query is to detect its intent. A list of five possible intents was defined, and a dataset of around 1,100 training examples was generated using seed samples and an LLM. This dataset contained both portuguese and english examples. The following table shows some examples of user queries and their associated intents:

Table 2: Examples of user queries and their associated intents.

User Query	Intent
Quanto tempo para cozer peito de frango	`get_prep_time`
Find low-carb meals	`list_by_tag`
What do I need for paella	`retrieve_ingredients`
Como fazer francesinha em casa	`find_recipe`
Pratos para cozinhar em 45 minutos	`list_by_time`

In the dataset building, it was necessary to ensure a clear separation between similar cases such as "recipes that take 30 minutes" (`list_by_time`) and "how long does arroz de pato take to cook?" (`get_prep_time`).

5.3 Entity Extraction

The next task is to extract relevant entities from the user query in order to have a base to build the SPARQL queries.

This is done using a similarity search with fuzzy matching against the knowledge graph entities. The entites are extracted according to the predicted intent that has the most confidence. The entities that are considered and thus passed to the pipeline are ordered according to their similarity score, if above a certain threshold. The following table shows the mapping of intents to the entities extracted by the pipeline:

Table 3: Mapping of intents to the entities extracted by the pipeline.

Intent	Extracted Entity / Slot
`get_prep_time`	`recipe_name`
`list_by_tag`	`tag`
`retrieve_ingredients`	`recipe_name`
`find_recipe`	`recipe_name`
`list_by_ingredient`	`ingredient`
`list_by_time`	`minutes`

5.4 SPAQRQL Queries

The valid entities extracted from the user query are then used to build SPARQL queries to retrieve the desired information from the knowledge graph. Each intent has its own SPARQL query template that is filled with the extracted entities. The following graphs shows the mapping of intents to their corresponding SPARQL query functions:

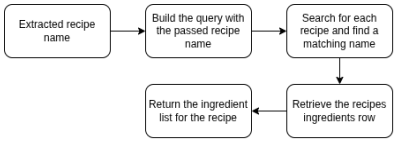

Figure 1: Retrieve Ingredients

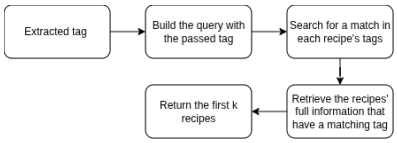

Figure 2: List by Tag

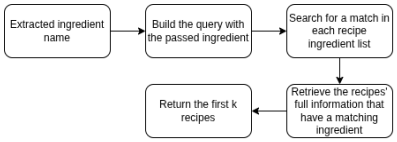

Figure 3: List by Ingredient

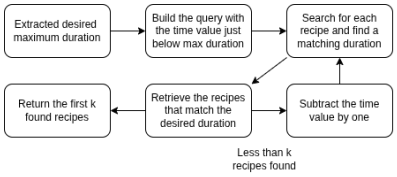

Figure 4: Get Maximum Time

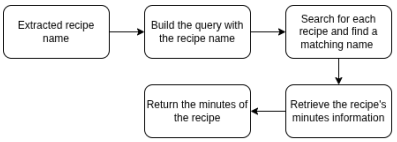

Figure 5: Get Preparation Time

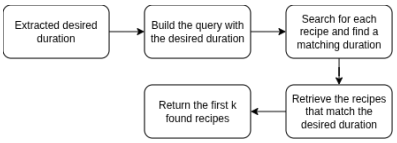

Figure 6: Get Exact Time

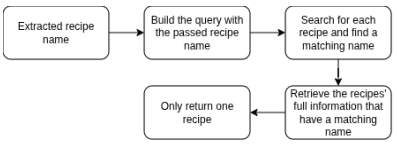

Figure 7: Find Recipe

5.5 Retrieval Augmented Generation

As the descriptions of the recipes in the knowledge graph are more textual and less structured,a RAG [Lewis *et al.*, 2020] (retrieval augmented generation) system was implemented to be able to retrieve the most relevant descriptions based on the user query.

5.6 LLM Contextual Response

Finally, the retrieved information from the SPARQL queries and the RAG system is used to generate a summary response to the user query. This is done using an LLM that takes as input the user query, the retrieved recipes, and their descriptions. The LLM then generates a coherent and informative response that answers the user's query based on the retrieved information and past conversation history.

6 Implementation

This section will explain in detail how each of the components described in the approach section were implemented.

6.1 Knowledge Graph Construction

After pre-processing the dataset the next step is to generate a knowledge graph in RDFS format. The script built for this task uses the RDFlib library [Developers, 2025b] and converts the CSV into an RDF/Turtle knowledge graph by creating cleaned URIs and structured entities for recipes, ingredients, tags, steps, and nutrition, then serializes the result to a .ttl file.

Each recipe is represented as an instance of the `Recipe` class, with properties linking to the necessary components. Some of these components have their own classes, as shown below:

Table 4: Recipe class attributes & RDFS classes.

Recipe Attributes	RDFS Classes
recipe	Recipe
ingredient	Ingredient
tag	Tag
step	Step
nutrition	Nutrition
id	
minutes	
n_ingredients	
n_steps	

This structured representation allows for the reuse of entities like ingredients and tags across multiple recipes without duplication. The steps were transformed into a RDF sequence (rdf:Seq) as shown in the example above. To maintain their order, each step is linked to the recipe using the `ex:hasStep` property. Nutrition attributes (calories, protein, saturated fat etc) are encapsulated in a separate `Nutrition` instance linked to the recipe via the `ex:hasNutrition` property. Tags are linked using the `ex:hasTag` property, allowing for categorization and filtering of recipes based on various user tags. This allows for efficient querying and easier retrieval of the relevant information for the predicted intent. A small subset with two recipes and their full atribute was put into a ttl tag for observation using the pyvis [Developers, 2025a] library after parsing,

6.2 Intent Training

The first step to analyse the user's query is to detect its intent. In order to do so an intent classifier was trained using a multilingual BERT model [Devlin *et al.*, 2019] and the LLM generated dataset mentioned in the section above.

Using this dataset, we trained a multilingual BERT [Devlin *et al.*, 2019] intent classifier: it loads the labeled CSV, tokenizes the text with `bert-base-multilingual-cased`, builds label mappings, fine-tunes a sequence-classification model, evaluates it on an 80/20 split, and finally saves the trained model, tokenizer, and label map for later use in the pipeline.

6.3 Entity Extraction

Alongside the classifier, we developed an entity extractor that uses spaCy [Honnibal *et al.*, 2020] for NER (name entity recognition) [Munnangi, 2024]and RapidFuzz [**?**] for fuzzy matching against the Knowledge Graph entities. As mentioned before, this allows for the extraction of relevant ingredients, tags, recipe names, and time constraints from the user query based on the predicted intent.

The `list_by_time` intent required special attention because user descriptions of preparation time are highly variable and often ambiguous. Unlike other intents that rely on explicit lexical markers, time expressions can appear in many different forms: numerical values ("30 minutos"), written numbers ("trinta minutos"), comparative forms ("menos de 20 minutos", "under 10 minutes"), vague descriptions ("algo rápido", "something quick"), and mixed-unit expressions such as "1h20" or "1 hora e meia". In addition, user queries may omit units, combine formats, or express constraints indirectly, which makes it difficult to rely on a single pattern or keyword. The system therefore has to distinguish between exact-time queries (asking for recipes that take exactly a specific number of minutes) and maximum-time queries (asking for recipes that take at most a certain amount of time). It also must interpret comparative structures correctly and normalize written forms into numeric values. For these reasons, the `list_by_time` logic integrates layered numeric parsing, distinction between exact and maximum constraints, translation normalization, and fallback mechanisms so that the fuzzy tag search can be used when no explicit time is detected.

The Language detection and normalization detects query language (PT/EN) using the fasttext model [Joulin *et al.*, 2016] after detection using Portuguese heuristics (accents, "ç" and common PT words); strips accents for robust matching while preserving original forms for display. spaCy pipeline: Loads the (PT/EN) spaCy models; adds a flag for keywords related to time (min/minutos/hora/hours); extracts noun chunks, preserving order. Cooking-time parsing in the `list_by_time` intent (PT/EN, tolerant to typos): Normalizes number words (e.g., "quarenta e cinco", "forty-five") into digits; uses regexes to detect minutes, hours, or combined "H+M" formats for the minutes component (e.g., "1h 30m" to "90"); detects range prefixes (e.g., "under", "menos de", "até") to separate exact vs. max_minutes. Translation with protected culinary terms: Uses Argos Translate offline [Argos Open Tech, 2025] for PT↔EN; a dual pass is used for objective results no matter the input language, since the KG is bilingual but has more EN labels; preserves protected PT terms (e.g., "açorda", "brás") by not translating them. Candidate tokenization (`content_tokens`): Removes stopwords and domain noise; sorts tokens for canonical matching; aggregation of tokens from original chunks, full text, and its translation. Knowledge Graph indexing: Loads the KG labels into rdflib [Developers, 2025b]; precomputes accent-stripped normalized labels for fast fuzzy search. Intent-aware fuzzy search and linking: RapidFuzz [Bachmann and contributors, 2025] over normalized labels with thresholds: `ingredient`=85, `tag`=90, `recipe_name`=70; Portuguese labels are preferred with an increment of 3 percentual points when the query appears to be PT (this way Portuguese labels are favored).

After this the retrieved information is used to generate different SPARQL queries for each intent.These are explained in the following section:

6.4 SPARQL Queries

6.5 Pipeline

This file detects language, original and translated query, and detected slots before SPARQL execution. After, it runs the sequential pipeline and prints per-intent SPARQL outputs; returns a structured dict containing `intent`, `confidence`, `text`, `slots`, and `kg_results`.

The sequential query execution (`handle_query`) determines the user intent and extracts slots with

extract_and_link, then routes execution to the appropriate SPARQL query based on the intent. list_by_ingredient: Collects all ingredient labels and runs query_list_by_ingredient for each. list_by_tag: Uses all extracted tag labels, executes query_list_by_tag, and deduplicates recipes by URI. find_recipe: For each candidate recipe name, calls query_find_recipe and prints a full recipe card. retrieve_ingredients: Calls query_retrieve_ingredients for each recipe name and prints the ingredient list. get_prep_time: For each recipe name, runs query_get_prep_time and prints the preparation time in minutes. list_by_time: Reads cooking_time (exact) or max_minutes (range) from the slots. Range queries use query_by_max_minutes; exact queries use query_by_exact_minutes; sets result_basis. If no results or no numbers are detected, performs a fuzzy tag fallback: injects time-related tags (15/30/60-minutes-or-less), queries query_list_by_tag, deduplicates, and sets the result to fuzzy_tag_fallback.

7 RAG implementation

As mentioned before, this module builds and serves a vector-based semantic search index over the more textual recipe descriptions. Before indexing, the code applies a quality-filtering step as short, boilerplate, or low-information descriptions are detected using heuristics such as length, alphabetic-character ratio, presence of boilerplate patterns (e.g., prep time, yield), and token count. Any description that fails these checks is skipped. Valid descriptions are then turned into "chunks". In this version, each description is a single chunk which is stored in a list of records containing the fields {"id", "name", "chunk_id", "text"}. The build step encodes all chunks using a SentenceTransformer [Reimers and Gurevych, 2019] mode. The resulting embeddings are L2-normalized and saved to embeddings.npy. A FAISS inner-product index is then created and written to faiss.index. Alongside it, a metadata file is generated so it can be used later for querying. This whole prosses only needs to be run once, and the resulting files are stored for later queries. When querying, the index and metadata are lazily loaded and cached. A query string is encoded using the same model and the FAISS index retrieves the most similar chunks. Extra candidates are fetched and deduplication is employed by their chunk_id. Results below a similarity threshold are filtered out. The returned list includes the recipe ID, recipe name, chunk ID, the original description text, and the similarity score.

8 Decision of LLM summarization module and implementation

Initial tests with the local Ollama 8b [AI, 2024] model proved unsatisfactory. Despite the advantage of running locally without relying on an external service, the models demonstrated semantic inconsistencies with a more critical issue being the amount of time required for responses for each query. Consequently, the implementation was migrated to the Groq API [Groq, Inc., 2025], which provided, with the use of significantly bigger models, improved response accuracy, coherence across turns, and overall stability in the time it took to process and generate the expected output. The Groq API is initialized lazily, ensuring that the module is loaded only when required. The file that implements this module does the following steps:

- conditional loading of environment variables;
- lazy loading of the semantic pipeline (ensuring it's only initialized once and resused);
- integration with the RAG similarity engine;
- prompt construction with history and response policy;
- sending requests to Groq via client.chat.completions;
- logging conversation history in context.txt;
- maintaining a running session.

8.1 Frontend details

The frontend of the recipe chatbot was designed with several key user-focused features. Users can create a new chat, which clears the message history stored in the Groq prompt, ensuring that subsequent queries start with a fresh context. While the model is retrieving the context and generating the response, the interface allows users to view the previous message history for reference. The displayed text on the interface can also be dynamically translated. Additionally, all LLM responses are converted to Markdown for rendering, enabling proper formatting of tables and structured content. These frontend features are integrated with the backend described earlier, which is exposed via a FastAPI server [Ramírez and contributors, 2019]. The API has endpoints for health checks, submitting chat queries, and clearing the conversation context. The frontend itself is implemented in React, with a dedicated fonts and components folder to manage the styling and reusable UI elements.

9 Problems & Solutions

9.1 Deduplication of results

One challenge encountered during the development of the pipeline was the handling of duplicate recipe results arising from overlapping filtering criteria. The knowledge graph's structure and tagging system make it possible for a single recipe to satisfy multiple filtering criteria at once—for example, a dish might be labeled both as "quick" and "under 30 minutes," or appear in multiple result sets inferred through different SPARQL paths. Without explicit handling, such overlaps would cause duplicated recipe entries in the output, misleading users and complicating evaluation of recall and precision. To solve this issue, the pipeline applies a deterministic de-duplication pass (as mentioned above) after SPARQL results are collected. It iterates over the returned recipes, tracks seen URIs in a set, and retains only the first occurrence of each unique recipe. This lightweight but effective mechanism guarantees that results remain clean, non-redundant, and semantically meaningful.

9.2 Translation normalization

In the early versions, the pipeline relied on a plain large-language-model translation step to normalize user queries into english before intent classification and entity extraction. While this worked reasonably well, it produced non-deterministic translations as the same portuguese phrasing translated into english would not necessarily translate back into the same portuguese string, and small paraphrases in the input would sometimes yield semantically equivalent but structurally different english or portuguese renderings. This inconsistency became problematic because the the spaCy entity extractor depends on predictable lexical patterns. To address this normalization measures were introduced to make the translation stage behave more objectively.

9.3 Language and Number normalization

This began by removing stopwords that added noise and led the LLM to produce stylistic rather than literal translations. Secondly, instead of translating token by token, which caused the LLM to reinterpret fragments, the entire sentence was trasnlate as a single unit. Afterwards, we fixed a systematic discrepancy in number handling as PT→EN translation tends to convert written numbers into digits (e.g., "trinta minutos" → "30 minutes"), while EN→PT does not. To preserve semantic symmetry, a digit-to-word mapping that rewrites numbers consistently in the canonical representation was introduced. Finally, we stored the original query, the detected language, and the normalized English form so downstream components always receive a stable and deterministic string for tokenization. Together, these fixes substantially reduced variation and improved the intent classification's reliability.

9.4 Pre-loading of heavy components

Another crucial architectural decision in the pipeline is the pre-loading of the heavy components, namely the multilingual spaCy pipeline and the knowledge-graph, before any user query is processed. The spaCy model initialization involves loading the statistical components, taggers, dependency parsers, custom extension attributes and multilingual vocabulary vectors, as performing this at runtime for every query would be expensive. Similarly, the knowledge graph, sourced from a Turtle file exceeding one million lines, took too long to be reparsed from RDF on demand (33s for each query). To address this, the pipeline implements a PKL-based [Foundation, 2025] caching system: on the first run, the TTL file is parsed into an in-memory graph structure, which is then serialized to disk as a .pkl snapshot. Subsequent executions detect the presence of the pickle file and load the graph directly and only once at the start in a fraction of the time. This strategy not only improves performance dramatically but also stabilizes memory usage, since the expensive graph structure is built exactly once and then reused.

The pipeline also introduces a dedicated mechanism that controls when entity extraction is executed, preventing unnecessary or harmful parsing for intents that do not require semantic grounding. For queries such as greetings, confirmations, or generic help requests, attempting entity extraction would lead to inconsistent results, noise in the entity linker, and contradictions between the detected intent and the returned slot set.

Conversation History of Recent Messages

To avoid exceeding the per-minute request limit and reduce computational costs, a compact history of the last three messages was implemented. While the full context of all the results in the entity extraction is given to groq for the current message, for the messages in the history only two fields are stored:

- `query` – the original user question,
- `response` – the final response returned by the LLM.

This history is used to generate a reduced context block appended at the end of the *prompt*. The messages appear in order of recency, with the most recent first. The number of messages included is configurable (default: 3). Placing the history at the end ensures that the trimming system always cuts from the tail, preserving the most relevant part of the main prompt.

As this implementation made the prompts longer, it was also necessary to add a trimming mechanism to the current query. This is done by beginning with the items found in the fuzzy matching with the lowest confidences. Then, for each of these items, the SPARQL results are eliminated one by one until the threshold of 7000 tokens is reached. This was the value decide because the model used for the experiments (chatgpt 120b) [OpenAI, 2025a] has a per minute prompt limit of 8000.

9.5 Groq base prompt

Before each query is sent to Groq, a structured base prompt is provided that defines the model's role, constraints, and formatting rules. This prompt positions the model as a bilingual PT/EN recipe assistant and explicitly instructs it to always answer in the same language as the user's query—including translating tags, ingredients, steps, and recipe names when necessary. It also clarifies what contextual information the model is strictly allowed to use: the current query, up to three previous conversational messages, and any structured data extracted from the knowledge graph. Equally important, the prompt imposes strict behavioral guidelines: the model must rely solely on the provided structured data, avoid hallucinating missing information, and follow a modular answer format when multiple entities are detected (ingredients, tags, name). It also specifies how the model must handle meta-queries such as "what did I ask before?" by checking the last few conversational messages and replying strictly based on that context, if relevant. For off-topic queries, the prompt instructs the model not to fabricate recipe results and instead provide a short clarification request or a single safe example query that can guide the user back to a valid intent. The final section of the prompt firstly enforces a conservative use of semantic description snippets (RAG), allowing them only

when they are clearly relevant and never as a source for inventing missing structured data. Finally it also defines a discovery footer containing any secondary results that were not elaborated on in the main response. This prompt was built through iterative testing and refinement to ensure consistency and alignment with user expectations. After this base prompt, a structured block is appended containing:

- The pipeline result (intent, confidence, slots, SPARQL results);
- The top similarity snippets from the RAG index;
- The compact block of recent message history.

10 Results

In this section, the results of the experiments evaluating the performance of the recipe chatbot system are presented.

10.1 SPARQL Query Effectiveness vs Groq Responses

This evaluation focuses on the performance of two aspects of the system: the SPARQL query effectiveness and Groq's [Groq, Inc., 2025] ability to utilize the retrieved information to generate accurate and coherent responses. For that a series of tests were conducted using a diverse set of user queries. These were designed to cover various scenarios, with different levels of complexity and specificity. The system's ability to accurately interpret the queries, retrieve relevant recipes from the knowledge graph, and generate satisfactory responses was measured.

Table 5: Prompt assigned difficulty levels and example queries.

Metric	Result
Easy	Recipe's with tomato
Medium	Recipes with melon in under 10 minutes
Hard	Invent a recipe

Easy Queries

These queries were considered "easy" as they are straightforward queries with clear intent and well-defined entities. The system demonstrated high accuracy in both SPARQL retrieval and Groq response generation. The SPARQL queries correctly retrieved relevant recipes from the knowledge graph, and Groq produced coherent and contextually appropriate responses. The system consistently met user expectations for these straightforward requests.

Table 6: Example easy query and response.

Query	Receitas com tomate
Recipes in the response	Crock-Pot Salsa Chicken, Sopa Fácil de Repolho, Francesinha, Assado de Macarrão com Mussarela e Peru, Gazpacho de Camarão e Caranguejo Moído .

The recipes in the response were directly retrieved from the knowledge graph, showcasing the system's ability to accurately interpret and respond to simple ingredient-based queries.

Medium Queries

Medium-difficulty queries require more complex reasoning, such as having multiple intents, and as such presented more challenges. Since only one intent is extracted from a query, when more complexity was added and two intents are present, only one query was executed thus leading to incomplete information being provided to the LLM. However, Groq was still able to generate reasonable responses based on the available data, since its own language understanding capabilities helped filter the results, using the provided recipes.

Table 7: Example medium query and response.

Query	Receitas com tomate fáceis
LLM Response	Macarrão à carne com sopa condensada Campbell's, Frango à salsa na panela lenta, Sopa fácil de repolho, Gazpacho de camarão e caranguejo, Baguete rústico de legumes com abacate amassado

The response included recipes that were relevant to the ingredient "tomato" and addressed the "easy" aspect because it only included the recipes with the "easy" tag. The rest of the recipes retrieved were only mentioned in the footer as suggestions for further inquiry. This selection was due to Groq's interpretation capabilities used on top of the SPARQL query results. The system only identified the ingredient intent, leading to do the same search as it did in the easy query example.

Hard Queries

Hard queries are defined as open-ended or creative queries that challenge the system's ability to generate novel content, such as those requiring the invention of new recipes. These clearly showed the limitations of the current system. While groq showcased generative capabilities, the lack of relevant data from the SPARQL queries limited the quality of the responses. The system also struggled to produce meaningful content due to the defined constraints to invent past the retrieved data.

The system detected the `find_recipe` intent, and did search for a similarity match, giving fully unrelated recipes. However, groq used these unrelated recipes to ask back a question to the user on how he would like to invent the said recipe and giving suggestions that better fit the system's intent.

10.2 Language & Chat History

Portuguese vs English Queries

The system was evaluated on its ability to handle queries in both Portuguese and English. After the many improvements made to the translation and normalization process, the system showed robust performance across both languages. Although before there was discrepencies in the results when comparing the two languages, now the results are much more consistent, as stated in the table below:

Although these are very legitimate translations, it caused the query to retrieve different recipes from the KG due to the variations in phrasing. With the improved translation process, both languages now yield consistent and more deterministic results as its clear to see on the first two table entries, that are the most probable to be made my by a user. Some recipes

Table 8: Translastion before and after improvements.

Original prompt	Previous Translation	Improved Translation
Receitas com manteiga	Butter recipes	Recipes with butter
Recipes with butter	Receitas que contenham manteiga	Receitas com manteiga
Butter recipes	Receitas com manteiga	Receitas com manteiga

were already stable, such as the last entry, but now all variations are consistent.

Chat History

The system's handling of chat history was also evaluated. The inclusion of previous interactions in the prompt provided context for groq, enhancing the coherence and relevance of responses in subsequent and low context queries. However, excessive history led to longer prompts, which showed to sometimes overwhelm the model. This, together with the limit of tokens, led to the decision to limit the chat history to the last 3 interactions. The chatbot now is able to have more natural conversations, as it can refer back to previous queries and responses when necessary, improving the overall user experience.

10.3 RAG Impact

To assess the impact of the Retrieval-Augmented Generation (RAG) component, experiments were conducted comparing system performance with and without RAG integration. The results indicate that, in this system, the RAG component had little to no measurable impact on the quality of the generated responses. This outcome is likely due to the structured nature of the knowledge graph, which already provides comprehensive and relevant information for the intents being handled. This, coupled with the fact that the descriptions present in the dataset were often brief and lacked depth, meant that the additional context provided by RAG did not significantly enhance the model's ability to generate accurate or informative responses. Future work may explore scenarios where RAG could play a more pivotal role, particularly in cases where the knowledge graph is less comprehensive or when dealing with more complex queries that require deeper contextual understanding.

11 Conclusions and Future Work

This paper presented a comprehensive approach to building a bilingual recipe chatbot that leverages a knowledge graph, SPARQL queries, retrieval-augmented generation, and large language models. The system effectively interprets user queries, retrieves relevant recipes, and generates coherent responses in both Portuguese and English. The integration of various components, including intent classification, entity extraction, and translation normalization, contributed to the robustness and versatility of the chatbot.

Future work includes:

- expanding KG coverage;
- adding automatic evaluation and reinforcement learning from human feedback (RLHF);
- adding more relevant intents;
- improving RAG with better chunking and filtering;
- exploring more advanced LLMs and fine-tuning strategies;

References

[Agrawal *et al.*, 2024] Garima Agrawal, Tharindu Kumarage, Zeyad Alghamdi, and Huan Liu. Can knowledge graphs reduce hallucinations in llms?: A survey. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2024. Accepted paper in NAACL 2024; also available as arXiv:2311.07914 [cs.CL].

[AI, 2024] Meta AI. Llama 3 8b. https://ollama.com/library/llama3:8b, 2024. Model card hosted on Ollama.

[Antelo, 2021] Catarina Antelo. Portuguese meals dataset. https://www.kaggle.com/datasets/catarinaantelo/portuguese-meals, 2021.

[Argos Open Tech, 2025] Argos Open Tech. Argos open tech, 2025.

[Bachmann and contributors, 2025] Max Bachmann and RapidFuzz contributors. Rapidfuzz. https://github.com/rapidfuzz/RapidFuzz, 2025.

[Bordes *et al.*, 2013] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems*, pages 2787–2795, 2013.

[Developers, 2025a] PyVis Developers. Pyvis documentation. https://pyvis.readthedocs.io/en/latest/documentation.html, 2025. Python library for interactive network visualization.

[Developers, 2025b] RDFLib Developers. Rdflib documentation. https://rdflib.readthedocs.io/en/stable/, 2025. Python library for working with RDF.

[Devlin *et al.*, 2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2019. Submitted 11 Oct 2018 (v1); last revised 24 May 2019 (v2).

[Foundation, 2025] Python Software Foundation. pickle — python object serialization. https://docs.python.org/3/library/pickle.html, 2025. Python standard library documentation.

[Groq, Inc., 2025] Groq, Inc. Groq. https://groq.com/, 2025. Company website.

[Honnibal *et al.*, 2020] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spacy: Industrial-strength natural language processing in python. https://zenodo.org/record/1215370, 2020. Software release.

[Joulin *et al.*, 2016] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

[Karpukhin *et al.*, 2020] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020. EMNLP 2020; arXiv preprint.

[KL2, 2025] Course slides, 2025. Course materials, PowerPoint slides.

[Lewis *et al.*, 2019] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[Lewis *et al.*, 2020] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. *arXiv preprint arXiv:2005.11401*, 2020. Accepted at NeurIPS 2020; v4.

[Li, 2020] Shuyang Li. Food.com recipes and interactions dataset. https://www.kaggle.com/datasets/shuyangli94/food-com-recipes-and-user-interactions, 2020.

[Loesch *et al.*, 2025] T. Loesch, E. Durmuş, and R. Celebi. Reciperag: A knowledge graph-driven approach to personalized recipe retrieval and generation. In *Proceedings of RAGE-KG 2025*, 2025.

[Lui and Baldwin, 2012] Marco Lui and Timothy Baldwin. langid.py: An off-the-shelf language identification tool. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Demonstration Session*, pages 25–30, Jeju Island, Korea, 2012. Association for Computational Linguistics.

[Munnangi, 2024] Monica Munnangi. arxiv:2411.05057. https://arxiv.org/abs/2411.05057, 2024. University of Massachusetts, Amherst; mmunnangi@cs.umass.edu.

[OpenAI, 2025a] OpenAI. Gpt-oss-120b. https://huggingface.co/openai/gpt-oss-120b, 2025. Model card on Hugging Face.

[OpenAI, 2025b] OpenAI. Intent extraction dataset generated by chatgpt. Generated by ChatGPT (Version 5.1) during an interactive session, 2025. Large language model.

[Ramírez and contributors, 2019] Sebastián Ramírez and FastAPI contributors. Fastapi. https://fastapi.tiangolo.com/, 2019. Documentation.

[Reimers and Gurevych, 2019] Nils Reimers and Iryna Gurevych. Sentence-transformers. https://sbert.net/, 2019.

[Sun *et al.*, 2019] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations (ICLR)*, 2019. Accepted to ICLR 2019.

[Zhang *et al.*, 2019] Shuai Zhang, Yi Tay, Lina Yao, and Qi Liu. Quaternion knowledge graph embeddings. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. Accepted by NeurIPS 2019.