FRI Natural language processing course 2025
Project report

University *of Ljubljana*
Faculty *of Computer and Information Science*

Coccodrillo: Conversational Agent with Retrieval-Augmented Generation

Davide Belcastro, Constance Monluc, Ondra Pritel

Abstract

The abstract goes here. The abstract goes here.

Keywords
Keyword1, Keyword2, Keyword3 ...

Advisors: Slavko Žitnik

Introduction

Crocodile is a retrieval-augmented conversational agent that assists users with travel, weather, safety, and event planning. It combines a lightweight BERT-based intent classifier and QA module to extract entities (e.g., locations, dates) and correct input errors. Depending on intent, Crocodile queries real-time APIs (e.g., Bing News, Il Meteo, Yelp, TheTrainLine) or curated datasets (Lonely Planet, TasteAtlas), filters and summarizes results, and generates structured responses. For itinerary planning, it models points of interest as a weighted graph and computes optimal routes under a time budget. Crocodile thus delivers accurate, up-to-date guidance for informed travel decisions.

Structure

The our chatbot can will recognize three different pattern of questions;

- **What to do in (city) for (n) days? or What is the best path to visit (city) for (n) days to start in my hotel that is (here)?**
  If is possibile (maybe ask at the user) also include lunch and dinner breaks at typical local restaurants, snacks with traditional desserts, and typical dishes.
  If the user write a precision point of departure (like hotel or station) it will create a best path from the departure. Otherwise, it will write a path that start from the best place or in random way (choosen between k best places)

- **I'm leaving from (city start) to (city arrive) Could you give me the weather forecast, events, and traditions I might encounter during this period?** In this case it will search the argument that the user ask, and maybe can prose at the user to search other.

- **I'm thinking of leaving from (city start) to (city arrive) Could you find the best hotels, the best flight/-train routes, etc., with a budget of (euro)?**

- If the question not is similar at one of this three, it check if the question is similar at "travel" iusses. In this case it will ask on the web.
  Otherwise it will write "Sorry, i can answer only for travel iusses".

Warning: If it see that there's rain, it will give suggestions on indoor places to visit.

Methods

Our end-to-end pipeline integrates intent classification, entity extraction, and retrieval of real-time data from heterogeneous sources. The system comprises the following components:

Intent Classification

We employ the `all-MiniLM-L6-v2` BERT encoder fine-tuned for multi-class intent detection across eight categories (e.g., safety updates, weather alerts, travel logistics). This lightweight model encodes user queries into embeddings, which are subsequently passed through a softmax classifier to infer the most probable intent.

Entity Extraction and Clarification

To identify locations, dates, and other key entities, we use a BERT-based question-answering module (`bert-large-uncased-whole-word-masking-finetuned-squad`). Absent or ambiguous information (e.g., missing travel dates) triggers a clarification subroutine: the system formulates follow-up questions to the user, ensuring complete query parameters. Spelling and formatting errors are corrected via semantic similarity checks against gazetteer and date lexicons.

Data Retrieval and Integration

Depending on the classified intent, the system consults:

- **News APIs (Bing News, Viaggiare Sicuri)** for safety and weather alerts.
- **Domain-specific APIs and datasets:**
  - Lonely Planet and TasteAtlas (via Selenium & BeautifulSoup) for attractions and local dishes.
  - Bandsintown for concerts and events.
  - Yelp for restaurant recommendations.
  - Il Meteo for extended weather forecasts.
  - TheTrainLine for train, flight, and bus schedules.
- **Google Maps API** to verify real-time availability and status of points of interest.

News Summarization and Relevance Filtering

For safety-related queries in non-English locales, the system:

1. Retrieves articles in the target language.
2. Summarizes content using a `facebook/bart-large-cnn` pipeline.
3. Translates summaries back to English.
4. Ranks articles by computing TF-IDF vectors over the corpus (via `sklearn`), retaining only those above a similarity threshold relative to the user query.

Custom Itinerary Optimization

For sightseeing requests, we model points of interest as a weighted graph:

- *Nodes*: attractions with weights for aesthetic score and estimated visit duration.
- *Edges*: travel distances between nodes.

We apply a minimum-cost path algorithm under a time budget constraint (e.g., 8 hours/day number of days) to generate an optimal route. The start location is either user-defined or selected from top-ranked nodes.

System Execution

Users invoke the system via `run.py`, which orchestrates the above modules, compiles retrieved data, and formats the final response into a coherent, structured summary.

Figures

You can insert figures that span over the whole page, or over just a single column. The first one, Figure 1, is an example of a figure that spans only across one of the two columns in the report.

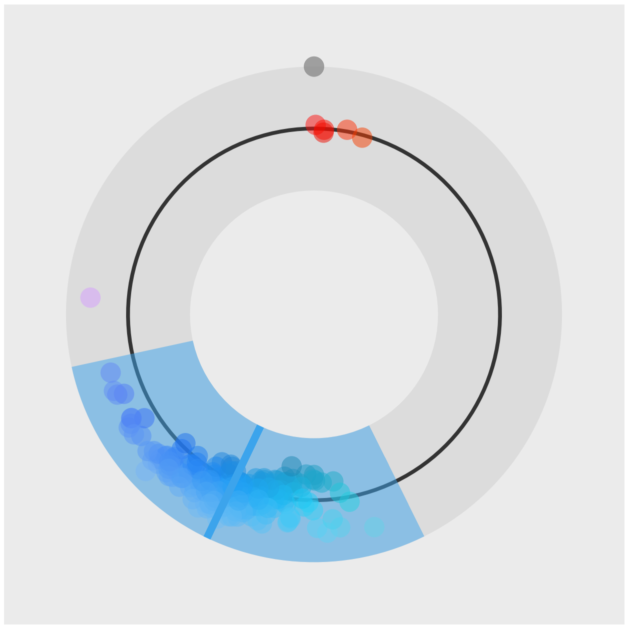

Figure 1. A random visualization. This is an example of a figure that spans only across one of the two columns.

On the other hand, Figure **??** is an example of a figure that spans across the whole page (across both columns) of the report.

Tables

Use the table environment to insert tables.

Table 1. Table of grades.

Name		
First name	Last Name	Grade
John	Doe	7.5
Jane	Doe	10
Mike	Smith	8

Code examples

You can also insert short code examples. You can specify them manually, or insert a whole file with code. Please avoid inserting long code snippets, advisors will have access to your repositories and can take a look at your code there. If necessary, you can use this technique to insert code (or pseudo code) of short algorithms that are crucial for the understanding of the manuscript.

Listing 1. Insert code directly from a file.

```
import os
import time
import random

fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Listing 2. Write the code you want to insert.

```
import(dplyr)
import(ggplot)

ggplot(diamonds,
           aes(x=carat, y=price, color=cut)) +
  geom_point() +
  geom_smooth()
```

Results

We evaluated system performance using three complementary test suites:

- **Final-output testing**, to verify that correctly posed queries yield appropriate, informative responses.
- **Intent classification testing**, to measure how often the model assigns the correct intent label.
- **Query-error testing**, to ensure the system gracefully handles under-specified or malformed queries.

In the final-output tests, we submitted well-formed questions and confirmed that the agent both recognized the intent and returned coherent, contextually relevant information.

For intent classification, our test set comprised 118 examples spanning all eight intent categories. The model achieved 91.53 % accuracy. Misclassifications are visualized in Figure 2. The most frequent confusion occurred between "Safety updates" and "Severe weather alerts," likely due to overlapping terminology. Contrary to expectations, "Best restaurants" and "Typical food dishes" exhibited minimal interchange errors.

Finally, query-error testing validated the system's ability to detect missing parameters (e.g., unspecified location or date) and prompt the user for clarification rather than producing nonsensical results.

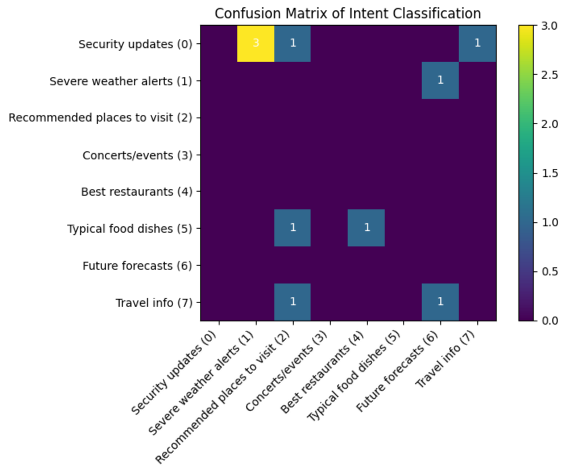

Figure 2. Intent Classification Confusion Matrix. Rows denote the true intent; columns denote the predicted intent.

Limitations

Although our system performs robustly in most scenarios, we identified the following areas for improvement:

- *Rate limiting and blocking*: Automated scraping of transportation schedules (e.g., TheTrainLine) can trigger anti-bot defenses after multiple rapid requests.
- *Data accuracy*: Occasionally, third-party APIs return stale or incorrect connections.
- *Translation constraints*: To maximize relevance, safety alerts are fetched in the local language and translated back to English; however, our current pipeline only handles short snippets, necessitating sentence-by-sentence translation.
- *Geographic coverage*: The itinerary planner is currently calibrated for Rome only; expanding to additional cities requires enriching our POI dataset and recalibrating route-optimization weights.

Future Work

Prior to final submission, we will:

- Implement request throttling and caching to mitigate blocking by external sites.
- Integrate fallback data sources to improve robustness when primary APIs fail.
- Extend the translation module to batch-process longer texts efficiently.
- Generalize the route-optimization framework to support multiple cities by importing and standardizing additional POI datasets.

Discussion

Use the Discussion section to objectively evaluate your work, do not just put praise on everything you did, be critical and exposes flaws and weaknesses of your solution. You can also explain what you would do differently if you would be able to start again and what upgrades could be done on the project in the future.

Acknowledgments

Here you can thank other persons (advisors, colleagues ...) that contributed to the successful completion of your project.

References