FRI Natural Language Processing Course 2025
Project report

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

Coccodrillo: A Retrieval-Augmented Conversational Agent for Smart Travel Guidance

Davide Belcastro, Constance Monluc, Ondra Pritel

Abstract

Coccodrillo is a multilingual, retrieval-augmented chatbot engineered to assist users in planning safe and informed travel experiences. Unlike traditional chatbots, it combines real-time data retrieval from APIs and web sources with curated datasets to answer questions about local safety, weather alerts, cuisine, events, transportation, and sightseeing. The system integrates a compact BERT-based intent classifier, a QA module for entity resolution, and a hybrid retrieval-summarization architecture. Its itinerary planning engine employs graph optimization under time constraints, delivering personalized travel suggestions. The system emphasizes robustness, linguistic adaptability, and user-centric interactions.

Keywords

Conversational AI, Retrieval-Augmented Generation, Travel Assistant, Intent Classification, Route Optimization

Advisors: Slavko Žitnik

Introduction

Modern travelers seek real-time, tailored information to plan their trips safely and enjoyably. While traditional chatbots operate on static knowledge, Coccodrillo introduces a dynamic, retrieval-augmented architecture capable of integrating current data from web sources and APIs. It assists users with travel decisions involving weather, safety, events, restaurants, attractions, and logistics. The system blends BERT-based intent detection, entity resolution, summarization, and optimization techniques to construct relevant, context-aware responses.

Structure

The system recognizes 8 main intents:

- **Safety updates for a location** – "Is it safe?"
  Sources: Bing News + Viaggiare Sicuri
- **Weather alerts and extreme weather conditions**
  Sources: Bing News
- **Recommended places to visit**
  Data: Dataset downloaded from Lonely Planet, with the help of Python scripts based on Selenium and BeautifulSoup.
  Integration: Uses Google Maps to check the availability of locations (for example, to know if a place is temporarily closed).
- **Information on concerts and events**
  Sources: Bandsintown
- **Best restaurants to eat**
  Sources: Yelp
- **Recommendations on typical dishes or foods**
  Data: Dataset downloaded from TasteAtlas, using Python scripts with Selenium and BeautifulSoup.
- **Future weather forecasts**
  Sources: Il Meteo
- **Information on trains, flights, and buses**
  Sources: TheTrainLine

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. It is important to mention that the model uses Google Maps to check whether each place is open or closed for business when the user plans to visit. In case of closure, the system will note this in the output.

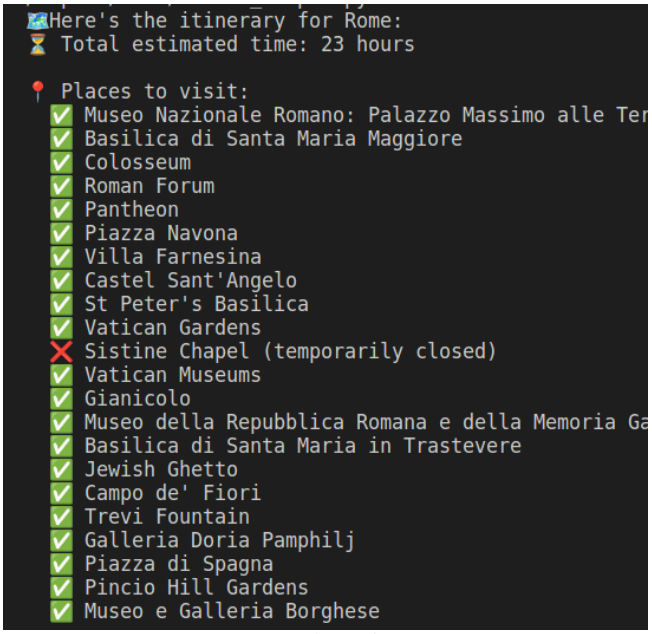

Figure 1. Places in Rome

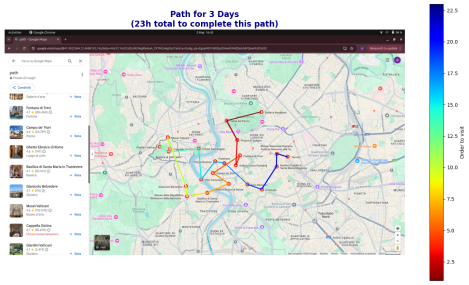

Figure 2. Example Path

For example, if the query is:
"I would like to go to Rome for 3 days. Can you recommend the best things to visit? Start from Termini Station."
With the following configuration:

- importance_time_visit = 0.0
- importance_beauty = 0.7
- importance_edge = 0.3

The output is on Fig 1.

> The total time for the complete itinerary is approximately **23 hours**, which fits within the 3-day plan (8 hours per day).
> We can notice that the **Sistine Chapel** is *closed* — as shown in the image 2 and 1, it is marked as *"temporarily closed"*.

Supported Cities

Currently, the route optimization feature supports the following cities:

- Rome
- Ljubljana
- Prague
- Vienna
- Florence
- Naples
- Maribor
- Paris
- Valencia
- Barcelona
- Madrid

Future Expansion

We plan to extend the list of supported cities in the future. The dataset for additional cities has already been downloaded but requires manual formatting to meet the model's specifications.

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.

System Output

Output structure:

- **First Response:**
  An introductory sentence generated with a bi-grams model based on a simple dataset, explaining that the system is searching for information.
  Hello, this is the information about the typical food you asked me for.
- **Second Response:**
  A structured output of all the information found.

Examples

Query 1

User: *Some concerts in Ljubljana for tomorrow*

Output:

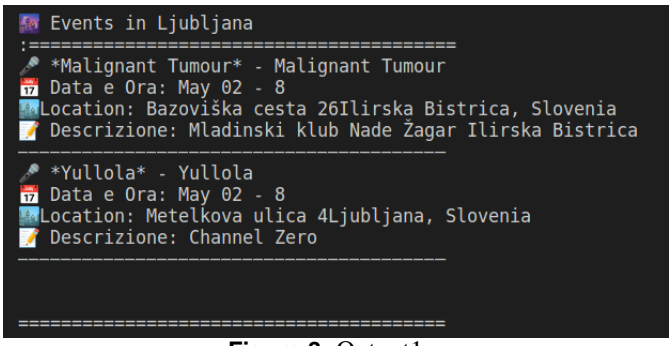

Figure 3. Output1

Query 2

User: *I am going in Berlin, tell me the temperature for friday.*

Output:

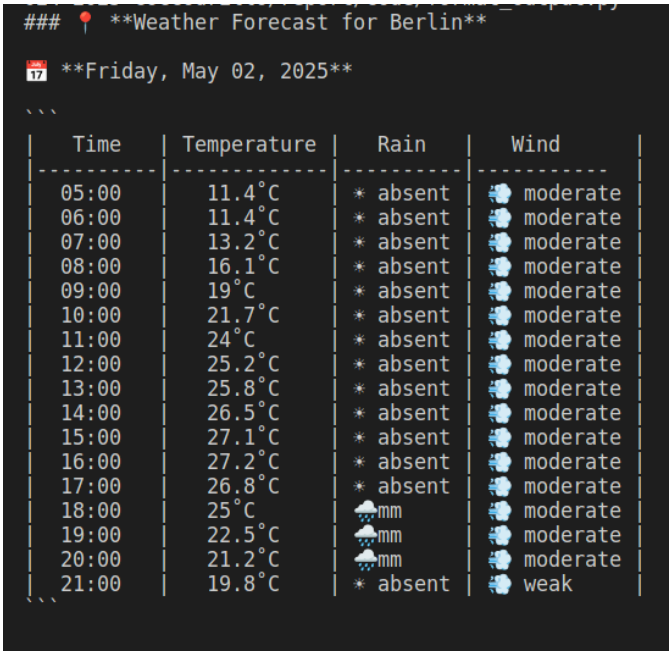

Figure 4. Output2

Testing and Performance Evalutation

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 Fig.5. 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.

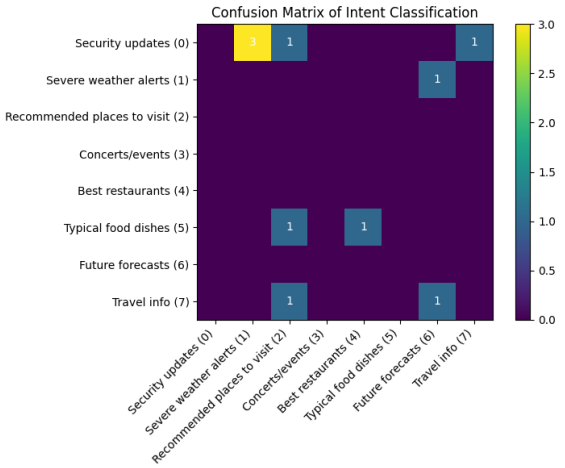

Figure 5. Intent Classification Confusion Matrix. Rows denote the true intent; columns denote the predicted intent.

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.

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

For a more detailed explanation, please refer to the README file available on GitHub at the following link.