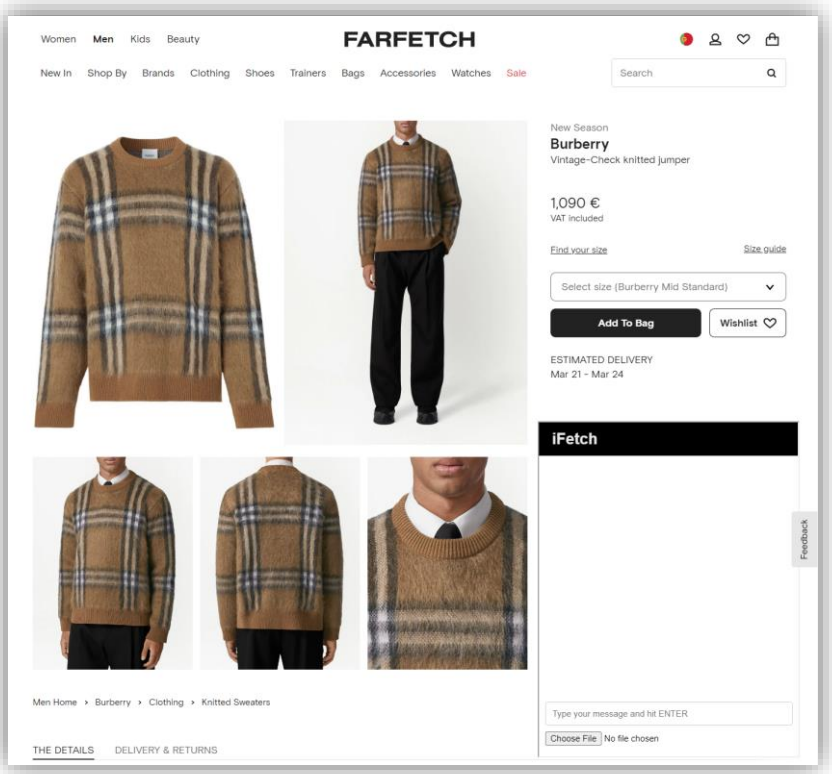
Conversational Shopping Agent

Web Search and Data Mining

Project Guide

João Magalhães
(jmag@fct.unl.pt)



Universidade Nova de Lisboa
2022/2023

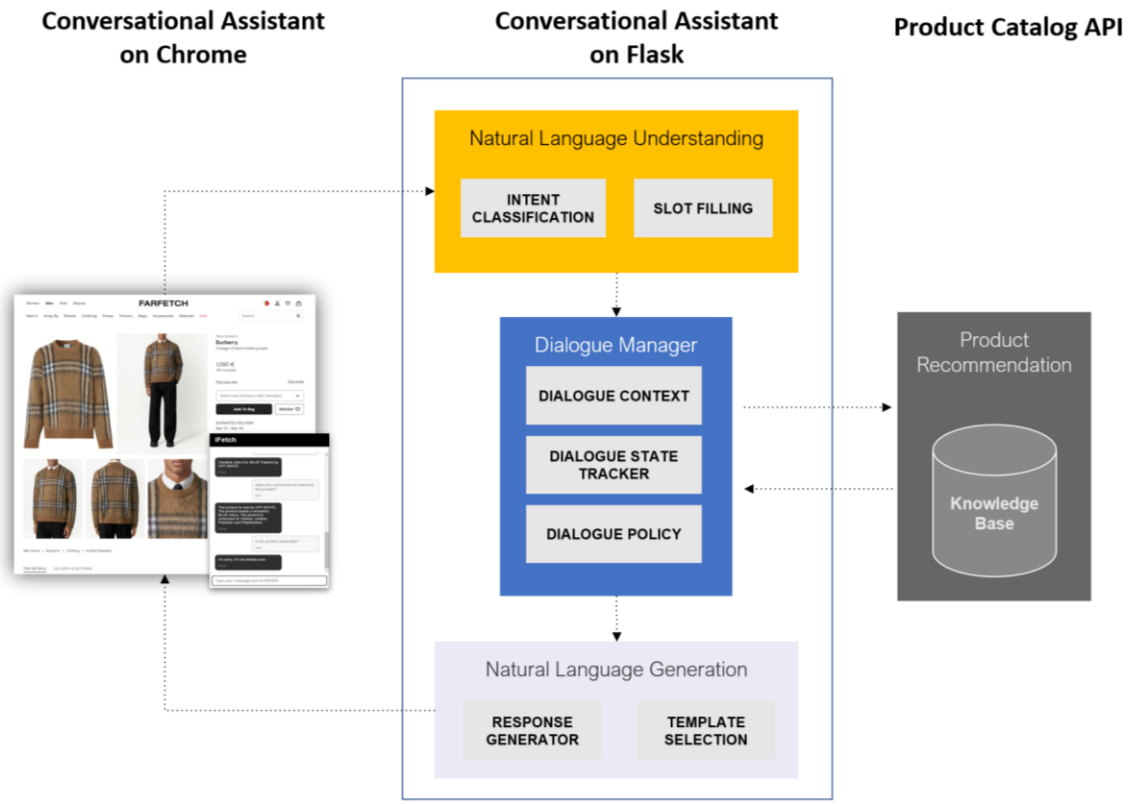
Introduction

In modern societies, there are many daily tasks that can be supported by conversational agents – we are currently witnessing the beginning of a larger shift from the traditional search-and-browse paradigm to the conversational paradigm. The WSDM project is positioned within this context.

You will implement a conversational assistant that will help users in finding the products they wish. Throughout the process, the assistant should be able to *talk about the product features*, *answer questions* about it and *find similar* or *matching products*.

The project will be divided into three parts, see the figure below:

- **Phase 1:** Framework and text-based product search.
- **Phase 2:** Multimodal product search.
- **Phase 3:** Natural language understanding and dialog manager.



Phase 1: Framework and text-based product search

In the first phase of the project, you will implement the basic skeleton of the conversational agent and interconnect the three components of the agent: the **Chrome Extension** (UI), the **Product Catalog** (KB) and the **basic dialog logic**. Following a natural conversation, the **basic dialog logic** should implement the basic procedure of a dialog: receive user utterance, reply with the agent utterance and consult the knowledge based (product catalog) whenever necessary. Ideally, the first prototype should be able to engage in a simple conversation.

Chrome extension

The iFetch Chrome Extension allows the integration of a conversational agent into the Farfetch store, allowing users to interact with a chatbot while browsing the store's items. The extension is available here for download and installation: https://github.com/pmvalente171/iFetch-Chrome-Extension

Flask

Your conversational agent will be implemented in the Flask framework. Flask is a web application framework providing a simple way to integrate the agent.

To install Flask, active the correct environment and install the following packages:

```
conda activate myenv
conda install flask
conda install flask_cors
```

The code below is an example of how your code should interact with the Chrome extension:

```
from flask import Flask, request
from flask_cors import CORS
import json

app = Flask(__name__)  # create the Flask app
app.config['CORS_HEADERS'] = 'Content-Type'
cors = CORS(app)

@app.route('/', methods=['POST'])
def dialog_turn():
    if request.is_json:
        data = request.json
        print(data)
        print(data.get('utterance'))
        print(data.get('session_id'))
        print(data.get('user_action'))
        print(data.get('interface_selected_product_id'))
        print(data.get('image'))

        responseDict = { "has_response": True, "recommendations":"",
                         "response":"Hello world!", "system_action":""}
        jsonString = json.dumps(responseDict)

    return jsonString

app.run(port=4000)  # run app in debug mode on port 5000
```

Product Catalog API

The **Product Catalog API** is an OpenSearch endpoint of a index with a wide range of products – the example notebook available in CLIP details how to access the API. Note that not all products are available.

The **OpenSearch tutorial** available at https://wiki.novasearch.org/wiki/courses/WSDM_2023 details its different capabilities. For a complete reference, please refer to the documentation.

Index mappings

The OpenSearch index contains a subset of Farfetch products and they are mapped into specific fields. Explore the contents of each field and experiment with different search queries.

```
'product_id': {'type': 'keyword'},
'outfits_ids': {'type': 'integer'},
'outfits_products': {'type': 'integer'},
'product_attributes': {'type': 'text'},
'product_brand': {'type': 'text'},
'product_category': {'type': 'text'},
'product_family': {'type': 'text'},
'product_gender': {'type': 'text'},
'product_highlights': {'type': 'text'},
'product_image_path': {'type': 'text'},
'product_main_colour': {'type': 'text'},
'product_materials': {'type': 'text'},
'product_second_color': {'type': 'text'},
'product_short_description': {'type': 'text'},
'product_sub_category': {'type': 'text'},
'combined_embedding': {'dimension': 512,
                       'method': {'engine': 'faiss',
                                  'name': 'hnsw',
                                  'parameters': {'ef_construction': 256,
                                                 'm': 48},
                                  'space_type': 'innerproduct'},
                       'type': 'knn_vector'},
'image_embedding': {'dimension': 512,
                       'method': {'engine': 'faiss',
                                  'name': 'hnsw',
                                  'parameters': {'ef_construction': 256,
                                                 'm': 48},
                                  'space_type': 'innerproduct'},
                       'type': 'knn_vector'},
'text_embedding': {'dimension': 512,
                   'method': {'engine': 'faiss',
                                  'name': 'hnsw',
                                  'parameters': {'ef_construction': 256,
                                                 'm': 48},
                                  'space_type': 'innerproduct'},
                   'type': 'knn_vector'}
```

Text-based search

- Understand how to use the OpenSearch framework. Read the provided OpenSearch tutorial and the OpenSearch documentation for a complete reference.
- Understand the search examples and apply them to the Farfetch product search.

Supported searches

In this phase you should implement different search methods to support searching with natural language questions and search with filters. This is intended search for recipes with specific traits, e.g. ingredients, servings, time, ratings, etc. Consider the following possibilities:

- text based search
- embeddings based search
- boolean filters
- search with boolean filters

Embeddings-based search

- Understand how to use the dual-encoders. Read the provided code and the documentation.
- Revise the index mappings to support k-nn vectors.
- Understand the different search examples.

Hints and advices

- Some tasks take a long time to process, e.g. computing the embeddings, hence, you may wish to put the embeddings in some persistent storage.
- Python pickles allow you to (de)serialize objects: https://docs.python.org/3/library/pickle.html
- HDF5 or Pandas DataFrames provide you a more structured solution you can use
- If you wish to store JSON objects, you can do so in the index, however, remember that whenever you delete the index you will lose this data.
- Consult Flask documentation: https://palletsprojects.com/p/flask/
- Consult OpenSearch documentation: https://opensearch.org/