UNIVERSITY OF TARTU

Institute of Computer Science

Computer Science

Carl-Christjan Bogoslovski

Interactive virtual assistant for Enhancing User Engagement with Open Government Data Portals

Bachelor's Thesis (9 ECTS)

Supervisors: Anastasija Nikiforova, PhD

Dimitris Symeonidis

Tartu 2025

Interactive virtual assistant for Enhancing User Engagement with Open Government Data Portals

Abstract:

Open Government Data (OGD) portals are designed to improve transparency and civic engagement by providing public access to datasets. However, these portals often suffer from low usability, technical complexity, and limited user guidance, reducing their impact and accessibility. This thesis presents a prototype of an interactive virtual assistant, *Oggy*, developed to enhance user engagement with OGD portals through natural language interaction. Built using Google Dialogflow CX and integrated with the European Data Portal API, the assistant enables users to search, summarize, download, and visualize datasets. The system leverages large language models (LLMs) for intent recognition and metadata explanation, along with supporting tools for chart generation and relevance scoring. The implementation was informed by a user survey (n=68), which identified common challenges and guided functional design decisions. Developed prototype evaluation results, in turn, suggest that while users find the assistant intuitive, improvements are needed in handling specific queries, maintaining conversational context, and delivering meaningful visualizations. The work contributes a modular, extensible framework for conversational access to open data and highlights the potential of virtual assistants in e-government services.

Keywords:

Open data, open government data, e-government, machine learning, virtual assistant

CERCS:

P170 Computer science, numerical analysis, systems, control; P175 Informatics, systems theory; P176 Artificial intelligence

Interaktiivne virtuaalne assistent kasutajate kaasamise tõhustamiseks avaandmete portaalides

Lühikokkuvõte:

Avaandmete portaalide (Open Government Data, OGD) eesmärk on edendada läbipaistvust ja kodanike kaasatust, pakkudes avalikkusele juurdepääsu riiklikele andmekogudele. Praktikas on aga paljudel neist portaalidest madal kasutatavus, tehniline keerukus ning vähene kasutajatuge pakkuv struktuur, mis piirab nende kasutusmugavust ja mõju. Käesolev lõputöö esitleb prototüüpi nimega *Oggy* – interaktiivset virtuaalset assistenti, mis toetab kasutajaid OGD portaalides loomuliku keele kaudu suhtlemisega. Assistendi arendus põhineb Google Dialogflow CX platvormil ning on integreeritud Euroopa andmeportaali API-ga. Selle kaudu saavad kasutajad otsida, kokku võtta, alla laadida ja visualiseerida avaandmeid. Süsteem kasutab intentsioonide tuvastamiseks ja metaandmete selgitamiseks suuri keelemudeleid (LLM) ning abivahendeid graafikute loomiseks ja andmete asjakohasuse hindamiseks. Prototüübi arendust suunas kasutajaküsitlus (n = 68), mille tulemustest selgusid peamised kitsaskohad ning funktsionaalsete vajaduste prioriteedid. Hindamistulemused näitavad, et kuigi kasutajad hindavad assistendi kasutusliidest intuitiivseks, vajab süsteem täiustamist täpsemate päringute käsitlemisel, vestluskonteksti säilitamisel ja tähenduslike visualiseeringute loomisel. Töö panustab modulaarse ja laiendatava lahenduse väljatöötamisse, mis võimaldab vestlusliidest avaandmete kasutamiseks ning toob esile virtuaalsete assistentide potentsiaali e-valitsemise teenustes.

Märksõnad:

Avaandmed, avatud valitsemise andmed, e-valitsemine, masinõpe, virtuaalne assistent

CERCS:

P170 Arvutiteadus, numbriline analüüs, süsteemid, juhtimine

P175 Informaatika, süsteemiteooria

P176 Tehisintellekt

Table of contents

Key Terms and Concepts

Open Government Data (OGD) – publicly available datasets published by governmental institutions. These datasets often include statistics, registers, geospatial data, and other resources intended to support transparency, innovation, and public services.

Virtual Assistant (VA) – conversational interface that interacts with users through natural language. It is capable of understanding user queries and responding with relevant information or actions in a dialogue-based format.

Webhook – a user-defined HTTP callback that allows a conversational agent to make real-time API requests and return dynamic responses based on the user's input.

Dialogflow CX – a platform for building advanced conversational agents. It provides tools for designing dialogue flows, managing intents and states of the conversation integrating back-end services via webhooks;

API (Application Programming Interface) – a standardized method that allows two software components or systems to communicate and exchange data with each other, typically over the internet. In the context of web services, APIs are often implemented using RESTful principles;

HTTP (Hypertext Transfer Protocol) – a protocol used for transmitting data over the web. It defines how messages are formatted and exchanged between clients (e.g., browsers, applications) and servers. HTTP supports **RESTful APIs**, enabling structured data requests and responses using standard methods such as GET, POST, PUT, and DELETE;

LLM (Large Language Model) – a machine learning model trained on large text corpora capable of understanding and generating human-like language. LLMs are used for tasks such as summarization, classification, and question answering;

Cloud Run – a Google Cloud service that allows developers to deploy and scale applications, such as webhooks in independent environments, without managing infrastructure;

JSON (JavaScript Object Notation) – a lightweight and widely-used data exchange format based on key-value pairs, often used for transmitting structured data between a client and server;

NLU (Natural Language Understanding) –a subfield of natural language processing (NLP) that focuses on interpreting user intent and extracting structured data (such as entities) from unstructured input;

CKAN (Comprehensive Knowledge Archive Network) – an open-source data management system widely used by governments and organizations to host and publish open data portals. It provides a RESTful API for querying and accessing metadata and datasets, enabling automated retrieval and integration with external systems such as VAs.

Introduction

Open Data has emerged as a movement that promotes the free access of data for anyone to use, reuse, and redistribute [1]. Governmental organizations make datasets publicly available to empower citizens, researchers, and businesses to create new services, conduct analysis, and hold governments accountable [2]. These datasets are distributed through open government data (OGD) portals - web-based platforms designed to share open government data [3]. In Estonia, the national open data portal provides access to a variety of datasets published by public authorities. As of 2025, the portal hosts over 5,600 datasets from different data publishers, offering data across domains such as environment, transport, and public administration [4]. The European Open Data portal aggregates metadata from public sector information portals across European countries, providing close to 2 000 000 datasets [5].

While the availability of open government data continues to grow, the usability of these platforms remains a challenge. OGD portals present several usability issues, partly due to the diverse needs of their users. For example, non-technical users may prefer keyword-based search and guided navigation, whereas data analysts typically require access to raw data, application programming interface (API) endpoints, and tools for visualizing and downloading data [6]. Regarding technical usability, inconsistent metadata and varying structure of datasets limit the effectiveness of these portals [7]. Addressing both user needs and structural limitations requires solutions aligned with the FAIR principles, i.e.,ensuring that data is findable, accessible, interoperable, and reusable [8].

The aim of this thesis is to develop a conversational virtual assistant that improves the usability of open government data portal. One promising solution for addressing common usability issues lies in leveraging interfaces which can interpret unstructured inputs from the user. Recent research has shown that natural language understanding (NLU) and large language models (LLMs) can enhance the usability of OGD portals [9]. Natural language simplifies the interaction with structured data, making dataset discovery and retrieval more intuitive for new or infrequent users [10].

Building on this premise, this study proposes a virtual assistant (VA) using the European Data Portal API, which aggregates multi-domain datasets across different countries. Compared to country-specific portals, such as statistics.gov.scot (used in prior research of LLMs in OGD portals [9]), the European portal offers centralized access, making it a suitable choice for a

virtual assistant that aims to simplify data retrieval and improve user interaction with open data repositories.

The virtual assistant is designed with two main functions: (1) **dataset retrieval** through extracting keywords from user queries and structuring them into API requests for dataset searches, as well as providing operations such as visualization and suggesting downloading dataset in one of available formats (e.g, JSON, XLS); (2) **website guidance for** answering common questions about datasets, documentation and portal functionality. The assistant is then evaluated with users defining future improvements.

The remainder of this thesis is structured as follows: Section 1 gives the background of core concepts for this thesis. Section 2 describes the methodology, including the survey, analysis approach, and evaluation strategy. Section 3 outlines the implementation framework of the prototype. Section 4 demonstrates the implementation through conversational flows. Section 5 presents the evaluation results. Section 6 discusses limitations and future developments, and the thesis ends with a conclusion in Section 7.

1. Background

This section introduces the concepts surrounding the thesis, i.e., open government data portals and virtual assistants,.

1.1 Open Government Data Portals

Open government data (OGD) refers to data made publicly available by government institutions for reuse by citizens, researchers, businesses and other governmental institutions. The concept is based on principles of transparency, accountability, and innovation promotion [2]. Platforms designed to support the discovery and access of such data are known as open data portals. As an example, the European Data Portal offers access to datasets from EU institutions, agencies, and national governments. The goal is to enhance public service delivery, support informed decision-making, and stimulate economic growth by making government-collected data freely accessible and reusable [2]. OGD portals typically host raw, structured datasets, rather than narrative or textual documents. To support the reuse of data, many portals provide application programming interfaces (APIs) that allow automated access to data, alongside advanced search functionalities that help users locate relevant datasets efficiently [11]. Metadata plays a pivotal role in the usability of these systems by describing the structure, content, and context of each dataset. It also allows for API access, allowing data to be systematically obtained and integrated into a variety of applications.

1.2 Virtual Assistants

Virtual assistants (VAs) are increasingly adopted in digital services due to their ability to provide real-time, interactive support through natural language interaction [14]. They are generally categorized into rule-based chatbots and AI-powered assistants. Rule-based chatbots typically follow decision trees or predefined scripts and are commonly used for handling routine tasks like answering frequently asked questions (FAQs) or directing users to resources. In contrast, AI-powered virtual assistants leverage natural language processing (NLP), contextual understanding, and in some cases voice and gesture capabilities to support more dynamic and complex user interactions [12].

In the context of OGD portals, virtual assistants offer a promising solution to overcome existing barriers to data access and user engagement. A frequent challenge for citizens is the

lack of intuitive support services on OGD portals. Traditional help sections, such as static FAQs or documentation, often fail to meet the needs of users, contributing to the underutilization of open data, even when they may be relevant to the needs of citizens [13].

1.3 Existing technologies and challenges

To date, the use of virtual assistants in public administrations has largely been limited to providing simple guidance and general information. A literature review [7] examining existing implementations of governmental virtual assistants identified only three studies [31, 32, 33] that systematically surveyed the state of the art in e-government chatbots. Most deployments were restricted to helping users locate government services, documents, or general information.

Advanced assistants are, however, increasingly being adopted across the public sector to automate complex workflows and improve service accessibility. According to Datafloq [14], conversational AI is already in use in areas such as tax services, healthcare, emergency response, and public administration, where it helps reduce response times, optimize operations, and ensure better access to services for diverse populations. Despite the potential benefits, such systems still face challenges related to data privacy, cybersecurity, and the need for regular updates and contextual accuracy [14].

These issues are further pointed out by Mamalis et al. [9], who describe a proof-of-concept application combining ChatGPT with the Scottish government's statistical data portal. Their implementation demonstrates the potential of LLMs in querying structured data but also highlights the importance of open-source, locally hosted alternatives to ensure transparency, privacy, and long-term trust in public sector systems.

In the context of the European Union, **Publio** is a virtual assistant tailored for the EU Publications Office to help navigate legal texts and documents using natural language [15]. However, its scope is limited to predefined task-specific functionalities and does not support dataset-level querying or structured data exploration, an essential capability for interacting with OGD portals.

Estonia has taken a more expansive approach with its Bürokratt virtual assistant network, which enables the training and deploying AI-based assistants across various public services. However, despite its broad vision, Bürokratt's current application remains limited with no

integration into the OGD portal. Based on an analysis of available documentation and existing services [16], Bürokratt mainly supports functions such as service registration or answering general queries, rather than facilitating structured data discovery or retrieval from open data repositories.

An example of an AI-based assistant used in the context of open government data is the Norwegian Government Data Portal, which integrates Google Vertex AI to interpret user input [17]. However, its functionality remains limited. More detailed or specific user intents often fail to produce relevant results (see Figure 1), indicating shortcomings in query interpretation and highlighting the need for further development.

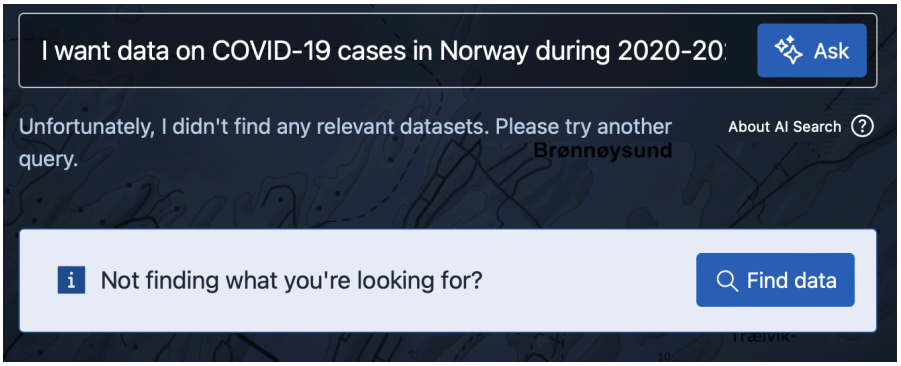


Figure 1. Example of the Norwegian Data Portal. User prompt requesting COVID-19 case data in Norway for 2020–2021 yields no relevant datasets.

The virtual assistant developed in this thesis addresses this gap by enabling communication with dataset repositories using modular design that allows for future developments and adjustments with open and privacy-compliant AI components. The following section outlines the methodology and research strategy used in the development and evaluation of this assistant.

2. Methodology

This section sets the methodological framework. It covers the overall research strategy, the design and execution of the user survey, the approach taken in developing the prototype, and the procedures used for evaluating and analysing the results.

2.1 Research Design

The thesis follows a practice-based and exploratory research design, focused on understanding user challenges in navigating OGD portals and testing whether a virtual assistant can help mitigate those challenges. The process consisted of three stages:

1. conducting a survey for identifying user needs regarding OGD portals;
2. designing and implementing a conversational assistant prototype informed by survey results;
3. evaluating the assistant's performance and user reception through user testing and feedback through survey.

This approach ensured that the assistant's features aligned with user expectations. The following subsection describes the motivations, methods and design of the survey.

2.2 Survey design

To identify user needs, expectations, and common challenges when interacting with OGD portals, a user survey was conducted. Its findings served as the fundamental basis for prioritising functional and visual features in the prototype's architecture. The objective of this survey was to: (1) evaluate the familiarity of users with virtual assistants and their experiences using similar technologies; (2) explore the specific challenges users face when accessing OGD portals; (3) identify the most valuable functionalities and design features that a virtual assistant should offer to enhance user engagement; (4) determine whether a virtual assistant is a necessary and effective solution for increasing accessibility and usability in OGD portals.

The survey was implemented using Microsoft Forms and followed key principles from Don Dillman's Tailored Design Method (2009), setting the framework for logical survey flow, clarity of question formulation, and thematic grouping [18].

The survey (provided in Appendix 3) was grouped into five sections, each aligned with a specific goal of the survey: (1) **respondent profile and prior virtual assistant experience** (Questions 1-5); **(2) use of open government data portals (**Questions 6–9); (3) **virtual assistant interactions** (Questions 10–14); (4) **virtual assistant attributes** (Questions 15–20); (5) **visual design** (Questions 21–27).

Questions 1–5 gathered respondents demographic and background information, such as country of origin, age group, professional environment and previous exposure to virtual assistants. Questions 6 through 9 focused on the frequency and purpose of OGD portal use.

Questions 10-14 asked users if they had encountered virtual assistants on OGD portals and whether they thought such a tool could improve their experience. Additional questions addressed characteristics such as natural language search, metadata summarization, and visualization capabilities.

For questions 15-20, respondents rated assistant capabilities and design features using a 5-point Likert scale (1 = Not important, 5 = Very important). The questions asked users to rank the importance of various assistant capabilities, such as assisting users with dataset search and recommending filters or data formats. Questions 21-27 requested users to rate the value of visual and usability features, such as clean interface design, interactive chips, responsive layout, and the use of icons or color, in driving UI design decisions for the deployed prototype.

The survey was distributed with the goal of reaching individuals who have previously interacted with or are regular users of OGD portals - whether as data users, analysts, civil workers or citizens. The survey was sent out to relevant communities, including the European Open Data Portal, the Open Data Charter community, and University of Tartu students. The survey was sent out via email, shared on LinkedIn and featured as an article on the European Data Portal [30] .

Regarding data analysis, the survey platform (Microsoft Forms) automatically gathered numerical responses for closed-form questions. These results were exported as a CSV file and used to identify trends in user priorities and assistant expectations. The CSV data was reviewed with the aid of GPT-4 to summarise response patterns, highlight prominent values, and generate charts [34]. All of the charts were generated by ChatGPT, unless stated otherwise.

2.3 Prototype Development Process

Based on the functional requirements identified through the user survey, a step-by-step implementation methodology was adopted to ensure that the assistant would meet user expectations in terms of functionality.

The development was divided into a series logical phases:

1. the initial creation of a virtual assistant in Dialogflow CX, including the structuring of conversational routes and the addition of static user intents;
2. the setup of a functional back-end capable of receiving webhook requests from Dialogflow CX and returning structured responses;
3. implementation of dataset search functionality, enabling users to query the European Data Portal directly through the assistant;
4. integration of website and documentation search using a programmable search engine;
5. the addition of language model–based keyword extracting from user intents for API searches;
6. refinement of user response structure based on conversational testing;
7. the implementation of user-triggered actions such as summarisation, download, and visualization.

Initial testing was conducted locally using Dialogflow's testing console and webhook logging, ensuring that each module functioned independently before full deployment.The overall strategy focused on functional stability, modularity of assistant design and alignment with user-informed priorities.

2.4 Prototype Feedback Design

To evaluate the usability and conversational success of the deployed prototype, a feedback survey was conducted. The aim of the survey was to assess how well the assistant performed in real user interactions and to identify strengths, shortcomings, and areas for improvement.

The objectives of the feedback survey were: (1) to measure the ease of use and success in accomplishing tasks (dataset or website search); (2) to evaluate user satisfaction with the assistant's functionality and interface; (3) to gather suggestions for further development; (4) to identify any issues experienced during use.

The survey was created using Microsoft Forms and distributed after the public deployment of the prototype. The link to the assistant and feedback form was shared with users who had tested the prototype through prior research outreach and platform networks.

The survey (provided in Appendix 4) consisted of eleven questions, grouped into four thematic sections: (1) **user background** (Questions 1–3) - age group, country of residence, and professional field; (2) **interaction experience** (Questions 4–6) seeking for respondents opinion about the ease of use, success in finding information, and available assistant features used; (3) **improvement feedback** (Questions 7–8) gathering suggestions for new functionality and reporting of errors or issues; (4) **overall evaluation** (Questions 9–11) assessing respondents satisfaction with the developed prototype, along with seeking for identifying willingness to participate in future testing and option to submit email address for future test notifications.

Questions seeking to assess the degree of agreement employed a 5-point Likert scale (e.g., "How easy was it to use the assistant?", "How satisfied are you overall?"). Binary questions (yes/no/partially) were used to assess conversation success rates and interest in participating in future testing. Multiple-choice and open-ended questions were used to collect feedback on the features used and suggestions for improvement.

The survey and prototype were published and made accessible via GitHub Pages. The results were interpreted qualitatively and quantitatively to develop the assistant. Findings from this feedback are presented in Section 5.2. The following section describes the implementation approach and technological choices for the virtual assistant prototype.

3. Implementation Framework

This section outlines the implementation approach and technological stack of the prototype application developed for this thesis. The assistant's functional scope and structure were directly shaped by the results of the user survey (see Section 5.1), emphasising the importance of natural language search, metadata interpretation, and datasets exploration tools. The core requirements were to: (1) provide a conversational interface for interacting with datasets in natural language; (2) support common user intents such as dataset search, metadata summarisation, dataset download, and visualisation; (3) maintain a clean look regarding the interface design of the assistant; (4) ensure access to an OGD portal, allowing real-time interaction with datasets; (5) be web-accessible and visible within a user interface.

To fulfil these requirements, the following technical components were selected: (1) **Dialogflow CX** as the conversational platform; (2) **Node.js** for implementing back-end logic; (3) **the European Data Portal API** for dataset search; (4) **GPT-3.5** for language model–based assistance (e.g, for extracting keywords from user intents); (5) GitHub Pages for hosting the front-end interface for the assistant.

Each component is described in the following sections, beginning with **Conversational Framework** in Section 3.1.

3.1 Conversational Framework

The choice of a conversational framework was influenced by both technical concerns and trends in public-sector virtual assistant deployments. To implement a virtual assistant capable of interacting with open government portals, the chosen framework needed capabilities such as understanding natural language queries, supporting multi-turn dialogues with state tracking and integrating with external APIs.

Although an initial review of related literature [7] identified Google Dialogflow as a popular framework in government-related services, the prototyping process in this study began with Rasa. This open-source framework was discovered through web searches and was selected for its modular architecture and customizability [19]. Rasa offers complete control over natural language understanding, dynamic dialogue management, and back-end integrations, therefore being a viable option for this thesis.

Setting up Rasa presented challenges due to hardware limitations. The development environment was based on an Apple computer running on ARM architecture, which caused compatibility issues with some of the dependencies for Rasa to work and the process became time-consuming given the project's timeline. As Rasa proved to be impractical due to compatibility issues and time constraints, an alternative framework was needed.

Google DialogFlow was selected as a suitable replacement. It is a NLU platform for designing conversational interfaces for virtual assistant applications, providing two main versions: **Dialogflow ES** (Essentials) and **Dialogflow CX** (Customer Experience) (as of May 2025 called **Conversational Agents**). Each version caters to different complexity levels in conversational design [20, 21]. Dialogflow ES is designed for simple conversational flows and offers quick deployment for straightforward use cases. It provides a basic intent-based structure, suited for simpler virtual assistants development [21]. Dialogflow CX is a more recent and advanced platform built for complex, multi-turn conversations. The Dialogflow CX console provides a visual, drag-and-drop interface for designing and managing conversational flows. Developers can define flows, pages, and intents in an intuitive manner. The platform also supports real-time agent testing, user journey tracking, and external API integration through configurable webhooks [20]. A comparison of the considered frameworks is presented in Table 1.

Table 1. Summary of conversational framework selections.

Framework	**Pros**	**Cons**
Rasa Open Source	● supports advanced NLU and dialogue control; ● local hosting; ● open-source.	● requires manual setup and technical expertise; ● not chosen due to compatibility issues.

Dialogflow ES	● quick to set up with minimal configuration; ● easy integration with Google services;	● limited support for complex dialogue flows; ● less suitable for multi-turn or dynamic interactions.
Dialogflow CX / Conversational Agents	● more recent than ES; ● designed for complex, multi-turn conversations; ● offers visual flow editor and built-in state management; ● supports LLM integration and webhooks.	● requires Google Cloud account and billing setup; ● less flexible than open-source alternatives in advanced customisation;

The assistant's front-end is hosted via GitHub Pages, providing users with a publicly accessible interface to interact with the Dialogflow CX agent and back-end services. The embedded interface runs a Conversational Messenger component, through which users communicate with the assistant. The assistant was given the name "Oggy" - a playful title inspired by the acronym OGD, while also serving as a neutral, non-technical reference point in conversations.

3.2 Back-end Framework

To support the assistant's conversational capabilities and external data access, a custom back-end service was developed. The back-end is responsible for handling incoming webhook requests from the Dialogflow CX agent, managing the logic for interacting with external APIs, and preparing structured responses to be delivered back to the user via the

front-end interface. The back-end has a modular structure and every functionality (dataset search, portal queries, user-initiated actions such as summarization) is handled by a separate module to maintain clarity of the source code.

Each back-end element was introduced to fulfil a specific need: (1) to fetch datasets from the European Data Portal (via API); (2) to perform Google Custom Search queries for searching general information, documentation and other resources on the European Data Portal; (3) to manage datasets tools logic (e.g., summarise/download/visualise); (4) to maintain a webhook entry point that routes requests to the appropriate module.

The following subsections describe each of these modules and their responsibilities in detail.

3.2.1 APIs

To manage user inputs and handle external API communication, the back-end is built using Node.js combined with Express.js, a lightweight web framework that simplifies HTTP request handling through modular routing [22, 23]. For external HTTP requests, the back-end leverages Axios, a JavaScript library that streamlines asynchronous API calls and facilitates handling of JSON responses [24].

Specifically, Axios is used to interact with the following APIs:

- **European Data Portal API** - the official programming interface provided by the European Union, offering access to diverse open datasets across Europe. The portal aggregates publicly available data from multiple sectors and countries, enabling developers to search, browse, and reuse European public sector information in applications and services, such as virtual assistants [25];

- **OpenAI API (GPT-3.5)** - a cloud-based API service that allows applications to leverage advanced language models developed by OpenAI, such as GPT-3.5. Trained on extensive textual data, the GPT-3.5 model is capable of interpreting and generating text in natural language, enabling applications to provide dynamic and contextually accurate conversations or text analysis [26];

- **Google Programmable Search Engine (Custom Search API)** - a customizable search interface provided by Google, allowing developers to limit search results to

specific websites or domains, offering users tailored and more precise search results compared to general web searches [27] .

The European Data Portal is built on top of **CKAN (Comprehensive Knowledge Archive Network)**, an open-source data management system widely adopted for publishing and managing open government data [28]. CKAN exposes a RESTful API, which allows external applications to query datasets and retrieve metadata. In this prototype, all dataset search operations are performed via CKAN's *package_search or package_show* endpoints[1].

3.2.2 Hosting and Deployment

To run the assistant in a publicly accessible environment, the back-end needed to be deployed to handle incoming HTTP requests from Dialogflow CX. For this reason, the assistant's back-end was deployed using Google Cloud Run - a platform for running containerized applications in response to HTTP requests. The platform was selected primarily for its native compatibility with Dialogflow CX in the Google Cloud ecosystem. Cloud Run dynamically adjusts resources based on the volume of incoming requests, without manually managing traffic [29].

The deployment used source upload via the Google Cloud Console, where the back-end code was directly uploaded and configured to run using the Node.js 20 (Ubuntu 22 Full) runtime. The entry point function (OGDHttp) was defined using the Cloud Run interface. Additional runtime settings were configured in Cloud Run: (1) **memory**: 512 MiB**;** (2) **CPU**: 1 vCPU; (3) **request timeout**: 60 seconds; (4) **concurrency**: 1 (single request per instance) (5) **startup CPU boost**: enabled; (6) **autoscaling**: enabled (up to 100 instances).

The configurations were based on Google Cloud Run's default runtime recommendations for lightweight web services. Except for the request timeout, other parameters followed the platform's defaults. The time was adjusted to 60 seconds to allow for unexpected delays in external API responses.

Environment variables were defined in the Cloud Run deployment environment to handle private credentials and values. The following variables were used: (1) **OPENAI_API_KEY** - the API key used for accessing OpenAI language models; (2)

[1] CKAN API documentation: https://docs.ckan.org/en/latest/api/

CUSTOM_SEARCH_API_KEY - the API key for the Google Custom Search service; (3) **CUSTOM_SEARCH_CX_ID -** the identifier for the custom search engine instance; (4) **LOG_EXECUTION_ID -** a custom variable used for request tracking and debugging.

These values enabled runtime access to external APIs (OpenAI, Google Custom Search) without containing secret keys in the prototype's source code. All communication between Dialogflow CX and Cloud Run was routed via HTTP webhooks, with responses returned in JSON format to support rich interactions.

To enable the web-based user interface, the prototype integrated Conversational Messenger, a customizable chat dialog element for virtual assistants provided by Google[2]. This element was embedded into a static HTML page and configured via attributes *project-id*, *agent-id*, and *intent*, which directly bind it to the Dialogflow CX agent, thereby enabling the assistant to receive and respond to user input through a chat dialog interface.

3.2.3 Large Language Model Integration

To support dynamic user interactions, the assistant integrates a large language model to handle the varying complexity of inputs (e.g, "*show energy datasets in Germany from 2020*" or "*datasets about Estonia*") and to generate explanations of metadata in natural language. Therefore, OpenAI's GPT-3.5 API was selected due to prior experience, strong generalisation capabilities and ease of integration. The model is used in two ways:

1. identifying and extracting structured parameters (topic, location, year) from user input;
2. generating explanations of dataset metadata (e.g., publisher, format, license) using natural language.

In cases where the user's intent is not recognized, the model responds with a fallback message or a clarification prompt (e.g., "*Could you specify which topic you are interested in?*"). This fallback logic is implemented within the webhook and helps prevent empty or irrelevant responses.

[2] Conversational Messenger documentation: https://cloud.google.com/dialogflow/cx/docs/concept/integration/dialogflow-messenger

Since the GPT-3.5 model requires a network call and external computation, the response times of LLMs sometimes take longer to reach the client. To avoid potential prolonged latency in the user interface, a request timeout of 60 seconds was defined in the Cloud Run configuration (see Section 3.2.5), preventing failed responses during dataset queries, metadata explanation tasks or generally handling complex user intents.

3.2.4 Supporting Tools and Techniques

In order to support dataset actions, such as metadata summarization, numerical data visualization and suggesting available file formats for downloading, additional tools and techniques were integrated. For visualization, the QuickChart.io[3] API was chosen due to simplicity of use. A lightweight web service, it allows the back-end to generate dynamic line charts based on CSV input. To rank the relevance of datasets returned by the European Data Portal API, a custom scoring algorithm was implemented, which ranks datasets by how well they match the user's query in terms of topic and location. For ensuring consistency across input and results, text normalization and tokenization are applied to both user queries and dataset metadata. Additionally, the system uses rich content formatting to construct structured responses in Dialogflow CX, including clickable dataset cards and suggestion chips. Cards are visual containers that display dataset details (e.g., title, description, download format) in a structured format to enhance readability. Chips are interactive buttons that offer users predefined reply options or next-step actions, helping streamline navigation and guide the interaction flow without requiring the user to type.

[3] The QuickChart.io API documentation: https://quickchart.io/documentation/

4. Implementation

The implementation of the assistant system is divided into three main components: the back-end logic responsible for API communication and parameter extraction, the front-end conversational agent configured in DialogFlow CX, and the deployment setup using Google Cloud Run, GitHub Pages and related services (see Figure 2).

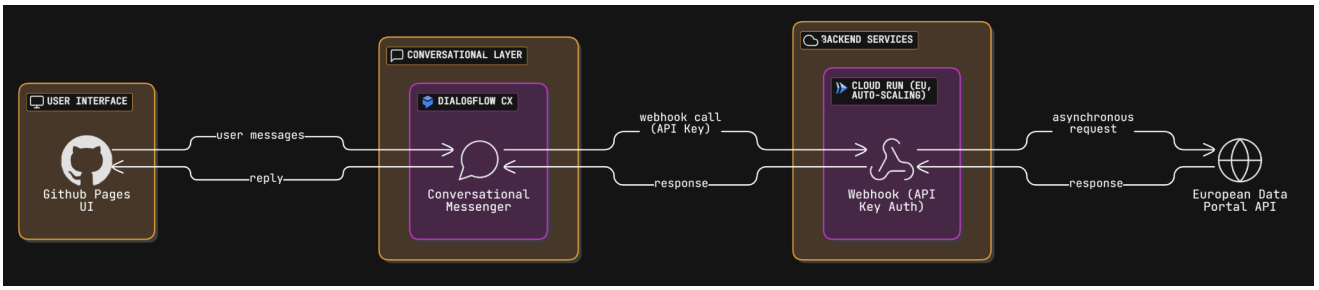


Figure 2. Architecture of the virtual assistant.

Since the scope and core functionalities of the assistant have already been defined in the previous section (see Section 3), the following subsections focus on the implementation of both the back-end and front-end components, explaining how each part of the system is structured and interconnected. Section 4.3 illustrates all of the possible user conversational flows.

4.1 Back-end

The back-end webhook for accessing the European Data Portal is organized into: (1) **the API Endpoint** (index.js) which connects Dialogflow with the European Data Portal by building a Node.js HTTP server, listening for POST requests, and routing them to appropriate handler modules; (2) **handler modules** (dataHandler.js, portalSearchHandler.js, datasetAction.js) for maintaining the response logic for user intents, processing specific functionalities such as searching, visualizing, downloading or summarizing datasets; (3) **helper modules** (llmHelper.js, responses.js, csvParser.js, datasetMetadata.js) that **s**upport core functionality including LLM-based parameter extraction, metadata summarization, CSV parsing, and rich response formatting for the conversational interface. The overall architecture of the webhook is illustrated in Figure 3.

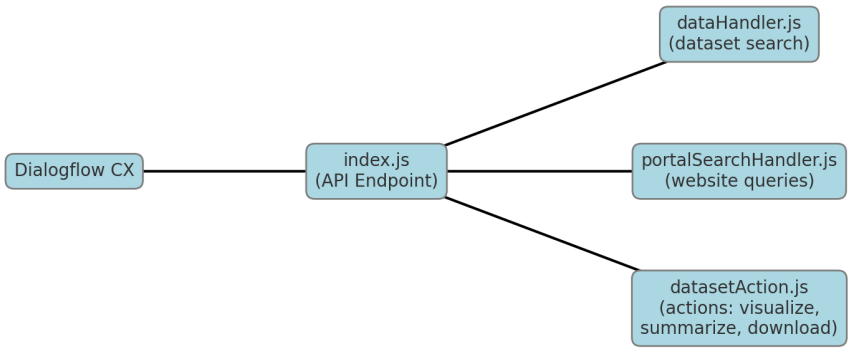


Figure 3. Webhook request routing flow.

4.1.1 API Endpoint

The main API endpoint is implemented in index.js using the Express.js framework (see Figure 4). Each incoming request includes the fields: (1) *sessionInfo* – stores session parameters (previously retrieved datasets), enabling further use; (2) *text* – contains the raw user input as a string, which can be parsed for command keywords or numeric dataset references; (3) *parameters* – includes extracted parameter values from user input (topic, location, year) based on Dialogflow CX entity recognition; (4) *fulfillmentInfo.tag* – identifies the **webhook tag** defined in Dialogflow CX, used to route the request to a specific action (e.g., "JsonPortalQuery"); (5) *intentInfo.displayName* – provides the name of the matched intent, which helps determine the type of user request (e.g., "DatasetSummarize").

```
// index.js
const express = require('express');
const { OGDHttp } = require('./indexWebhook');

const app = express();
app.use(express.json());

app.post('/', OGDHttp);
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
 console.log(`Server running on port ${PORT}`);
});
});
```

Figure 4. API endpoint file index.js.

Upon receiving a request, the indexWebhook.js file calls the function OGDHttp which extracts the intent name and webhook tag to determine which handler to access, both defined in Dialogflow (see Section 4.2). The routing logic handlers: (1) **dataHandler.js** - triggered for dataset search intents, triggered when the specified name of the intent is "JsonPortalQuery"; (2) **datasetAction.js** - handles dataset-specific actions such as "visualize", "summarize", or "download" based on the tag and parsed user input. This module is triggered with the following intent names: “DatasetsSummarize”, “DatasetsVisualize”, “DatasetsDownload”, “DatasetAction”, “DatasetFiletype”; (3) **portalSearchHandler.js** - handles general website search queries (set as default when intent is not recognized). The handlers are illustrated in Figure 5.

```
exports.OGDHttp = async (req, res) => {
 try {
   const intentName = req.body.intentInfo?.displayName || '';
   console.log("Intent name:", intentName);

   switch (intentName) {
     case 'JsonPortalQuery':
       return await queryDatasetsJson(req, res);


     case 'DatasetsSummarize':
     case 'DatasetsVisualize':
     case 'DatasetsDownload':
     case 'DatasetAction':
     case 'DatasetFiletype':
       return await handleDatasetAction(req, res);

     default:
       return await handlePortalSearch(req, res);
   }

 }
```

Figure 5. Intent routing in indexWebhook.js

The webhook includes validation to ensure required parameters, such as prompt text, are present and correctly formatted according to the expected schema defined by the assistant logic. If parsing fails or data is missing, a fallback message is returned to the user.

4.1.2 European Data Portal API Handler

The core dataset search logic is implemented in the **queryDatasetsJson** function, located in **dataHandler.js.** This handler receives a POST request from the Dialogflow CX webhook and is responsible for interpreting the user's prompt, constructing an API query, and formatting the results for display. To interpret natural user queries, the function uses **extractQueryParameters** from **llmHelper.js**. It uses GPT-3.5 to extract structured fields from the input prompt: topic, location, and year, as illustrated in Figure 6.

```
try {
   const parsedParams = await extractQueryParameters(userPrompt);
   keyword = parsedParams.topic || '';
   year = parsedParams.year || '';
   if (!locationStr) locationStr = parsedParams.location || '';
 } catch (err) {
   console.error("LLM extraction error:", err.message);
 }
```

Figure 6. Parameter extracting in dataHandler.js

These values are combined into a search string (see Figure 7). For example, on user input "*show me datasets about energy in Germany in 2020*", the combined term would be "*energy Germany 2020*".

```
const combinedTerm = [keyword, locationStr, year].filter(Boolean).join(" ");
```

Figure 7. Search term combined for an API request in dataHandler.js

The combined term is then passed to the function fetchDataEuropaJson(), which queries the European Data Portal using the CKAN API and returns the matching dataset results. This is illustrated in Figure 8.

```
try {
   result = await Promise.race([
     (async () => {
       const jsonResults = await fetchDataEuropaJson(combinedTerm, "en", 5);

       if (!jsonResults.length) {
         return {
           fulfillment_response: {
             messages: [{ text: { text: [`No datasets found for
"${combinedTerm}".`] } }]
           },
           sessionInfo: {
             parameters: {
               redirect_to_datasets: true,
```

```
            results: []
          }
        }
      };
    }
```

Figure 8: Fetching Data using the CKAN API in dataHandler.js. When no results are found, a relevant message is returned to the user, along with parameters for Dialogflow CX.

To keep the conversational interface simple, the system returns up to five datasets per query (see Figure 9). This restriction was intended to retain readability and allow users to easily analyze the findings within the chat flow, rather than overwhelming the user with large responses. Each dataset is processed to extract relevant metadata - title, description, publisher, available file formats, and distribution links for file downloads. These entries are deduplicated by title to prevent repetition. If no results are returned or the query exceeds 4.5 seconds, a fallback message is shown (e.g., *"The request timed out. Please try again later."*). The user is then offered the option to either restart the search from the beginning or return to the assistant's main menu.

```
exports.fetchDataEuropaJson = async (keyword, locale = "en", limit = 5) => {
 try {
   const baseUrl = 'https://data.europa.eu/api/hub/search/search';
   const url = `${baseUrl}?q=${encodeURIComponent(keyword)}&limit=${limit}`;
   console.log("data.europa.eu JSON API URL:", url);

   const response = await axios.get(url);
   const items = response.data?.result?.results || [];

   if (!items.length) {
     console.warn(`No results found for keyword: ${keyword}`);
   }
```

Figure 9: Data fetching using the CKAN API in dataEuropaJsonHandler.js

The API response is formatted using **formatDatasetsRichContent()** from the **responses.js** module, which generates a structured message containing rich cards and chips. The results, along with a *dataset_results_ready* flag, are stored in session parameters for further choice of activity to be taken on the dataset. (e.g., visualization or summarization actions in Dialogflow).

4.1.3 Dataset Actions

When a user selects a dataset action from in the chat by clicking one of the predefined chips (options being: "*Summarize a dataset from results*", "*Visualize a dataset from results*", "*Download a dataset from results*") they are directed to choose a dataset from the results of the API. The assistant parses the input using a regular expression that extracts the dataset index (/\d+/). The parsed index is then used to reference the requested dataset object stored in session memory under the results parameter, as illustrated in Figure 10.

```
const indexMatch = userInput.match(/\d+/);
const datasetIndex = indexMatch ? parseInt(indexMatch[0], 10) - 1 : null;
const dataset = results[datasetIndex];
```

Figure 10. The dataset action (tag) and the dataset index are identified by a regular pattern.

Each action corresponds to a switch-case structure in the webhook logic, where the Dialogflow CX webhook tag determines which dataset action to perform. As shown in Figure 11, when the tag is "summarize", the webhook constructs a metadata object by selecting relevant fields (title, description, formats, license) from the selected dataset. This object is then passed to the explainMetadata() function, which submits it as a structured prompt to the OpenAI API. The returned explanation is formatted into a text response and sent back via Dialogflow.

```
case 'summarize':
      const { explainMetadata } = require('./llmHelper');

      const metadata = {
        title: dataset.title,
        description: dataset.description || '',
        country: dataset.country,
        formats: dataset.formats || [],
        csvLinks: dataset.csvLinks || [],
        license: dataset.license || '',
        publisher: dataset.publisher || ''
      };


      const explanation = await explainMetadata(metadata);

      return res.json({
        fulfillment_response: {
          messages: [
            { text: { text: [`📘 Metadata summarization for
**${dataset.title}**:\n${explanation}`] } }
          ]
        }
      });
```

Figure 11. Metadata explanation in datasetsAction.js.

For visualization (tag “visualize”), the assistant attempts to retrieve a CSV download link either from the formatLinks["csv"] object (a field in the dataset object that maps available formats to their respective download links) or directly from the dataset’s csvLinks array, as illustrated in Figure 12.

```
const csvUrl = dataset.formatLinks?.["csv"]?.[0] || dataset.csvLinks?.[0];
const rows = await parseCsvFromUrl(csvUrl);
```

Figure 12. The visualization parses the data and chooses the CSV URL

If a CSV is available, it is parsed using the PapaParse library[4] - a JavaScript parser for CSV data that reads and transforms raw CSV text into a structured JavaScript array of objects. The assistant selects up to three numeric columns and one categorical column to construct a data visualization URL via the QuickChart.io API (see Figure 13), which is returned as a clickable link in markdown format.

```
const numericKeys = Object.keys(rows[0]).filter(key =>
  !isNaN(parseFloat(rows[0][key]))
);
const chartUrl =
`https://quickchart.io/chart?c=${encodeURIComponent(JSON.stringify({...}))}`;
```

Figure 13. A graph is created by selecting numerical columns.

In the case of “download”, the assistant checks how many formats are available. If more than one, it dynamically generates chips (interactive options) labeled “Download dataset 2 as CSV”. If only one format is available, it immediately returns a direct download link using the text message format, as shown in Figure 14.

```
return res.json({
        fulfillment_response: {
          messages: [{
            payload: {
              richContent: [
                [
                  {
                    type: "chips",
                    title: `Which format would you like to download **${dataset.title}**
in?`,
                    options: availableFormats.map(format => ({
                      text: `Download dataset ${datasetIndex + 1} as
${format.toUpperCase()}`
                    }))
                  }
```

[4] The PapaParse library https://www.papaparse.com/docs

```
              ]
            ]
          }
        }]
      },
      sessionInfo: {
              parameters: {
                dataset_results_ready: false,
              }
            }
    });
```

Figure 14. The webhook returns suggestion chips with available file formats as content.

4.1.4 LLM Helper

The LLM helper file manages the assistants' communication with OpenAI's API, encapsulating all logic related to interacting with available GPT models, such as GPT-3.5 and GPT-4. The central function for searching the portal for datasets, extractQueryParameters(userQuery), constructs a system prompt instructing the model to extract a topic, location, and year from a user's free-form input. The assistant expects the model to return only valid JSON in a predefined structure. Specifically, the model must respond with an object containing three fields: "topic", "location", and "year", each holding a string value. This is enforced via the system message and a try/catch block that falls back to an alternative model - GPT-4 instead of the GPT-3.5 if it fails.

```
const prompt = `
You are a helpful assistant...
Return JSON only:
{ "topic": "", "location": "", "year": "" }
`;

...

const completion = await openai.createChatCompletion({ ... });
const parsed = JSON.parse(completion.data.choices[0]?.message?.content?.trim());

...

try {
  return await tryModel(PRIMARY_MODEL);
} catch {
  return await tryModel(FALLBACK_MODEL);
}
...
```

Figure 15. The user's free text is used to prepare the query for LLM. The model's response must be in JSON format according to the prompt. The accuracy of the response is confirmed once a GPT request is made. A fallback model is used in the event that one model fails.

The helper also includes refineSearchQuery(), which is used when the user chooses to search from the European Data Portal website. In this case, the assistant reformulates question-based queries like "Where can I find API documentation?" into concise keyword-based search terms such as "data portal API guide", improving search accuracy and relevance of results returned by the Google Custom Search API.

To help users quickly understand unfamiliar datasets without reading the contents, the assistant uses a summarization function called explainMetadata(). This function converts the dataset's metadata into a JSON structure and sends it to the LLM along with an instruction to summarize its content concisely.

4.1.5 Response Formatter

The purpose of the response formatting module is to transform dataset objects returned from the API into a structured, interactive format that is compatible with Dialogflow CX, enabling a user-friendly conversational experience, as Dialogflow CX agents require responses to follow a specific rich content schema. The module implements the function formatDatasetsRichContent() which accepts three inputs: the original user keyword query, the country (if extracted), and the list of dataset results. The function then scores the datasets (see Figure 16) based on keyword relevance and returns them as an array of card objects to be displayed in the Dialogflow CX interface. The score is calculated based on whether the query keyword appears in the dataset's title or description and whether the selected location matches the dataset's country tag. Based on this score, datasets are classified as "relevant" or "other".

```
const relevanceScore = (item) => {
  let score = 0;
  if (title.includes(keyword)) score += 3;
  if (description.includes(keyword)) score += 2;
  ...
  return score;
```

```
};

const cards = results.map((item, i) => ([
  {
    type: "info",
    title: `${i + 1}. ${item.title}`,
    subtitle: `📍 Country: ${item.country}`,
    actionLink: item.link
  }
]));
const chipOptions = results.map((item) => ({
  text: `📊 ${truncateTitle(item.title)}`
}));
```

Figure 16. To extract the most relevant results, keywords are used to rate the results. Maps are created for each outcome. Selection boxes are created by generating suggestion chips

Each dataset is then formatted into a Dialogflow “info” card with a title, subtitle (usually the country), and a clickable actionLink. The top five datasets (based on score) are grouped into cards using Dialogflow’s “info” card structure. Each card includes the dataset’s title, a subtitle (typically the country of origin), and a clickable action link that leads to the original dataset page.

Once dataset results are available, the `buildActionChips()` function is invoked to generate a list of suggestion chips for user interaction. Each chip consists of: (1) a numeric prefix indicating the dataset index; (2) a truncated version of the dataset title (to improve display fit); (3) an action-related emoji (e.g. 📊 for "visualize", 📥 for "download", 📘 for "summarize").

The emoji is selected based on the type of user action passed into the function. The chips serve as quick-select options rendered in Dialogflow's UI under the selected action (e.g. "summarize a dataset"). Each chip allows the user to initiate the corresponding operation for the selected dataset index.

The whole response, including cards and chips, is returned to Dialogflow CX as a structured JSON payload using the fulfillment_response key, resulting in a response in the virtual assistant's front-end layer.

4.2 Front-end

The front-end implementation consists of designing the conversation flow based on user intent routes and integrating the agent with Conversational Messenger for the end user. The agent was designed with a modular flow-based architecture in DialogFlow CX (see Figure 17) , structured around three key sections:

1. the Start Page that automatically triggers a routing event (SHOW_SERVICE_PAGE) and guides the user to the Service page without requiring initial input;

2. the Service Page that enables users to choose between exploring datasets or accessing website-related support, such as finding documentation, portal usage guides, or answers to frequently asked questions;

3. two conversational flows:

    a. searching datasets from the portal by user input and using tools for visualizing, downloading or explaining datasets;

    b. searching the European Data Portal website for FAQ-like interactions and web search functionalities, powered by a custom Google search engine.

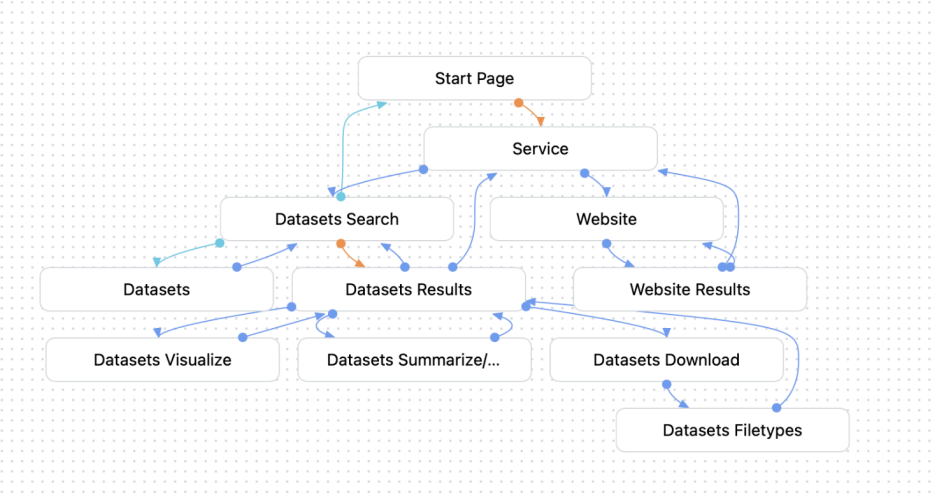


Figure 17. Agent Architecture view in Conversational Agents

The dialog design was built using pages and routes, enabling transitions based on user intent. The Datasets and Website pages route to corresponding results pages, where the user can then choose different paths:

1. *Datasets* - the user can access pages for visualizing, summarizing or downloading the found datasets;
2. for *Website*, the user can continue searching for datasets or go back to the Service page to choose a functionality.

All pages use intent-based routing, where route groups evaluate user intent and direct to the appropriate page. Each intent is assigned training phrases and parameters, e.g.:

- "JsonPortalQuery" handles user-defined natural language queries such as "traffic data in Estonia from 2020";
- "DatasetAction" captures user-selected inputs such as "download dataset 3";
- "DatasetFiletype" handles available file types for downloading (e.g., "as CSV");
- fallback intents exist on each page to handle unrecognized input and either re-ask or loop back.

Every functional page - Datasets, Visualize, Website includes webhook triggers. The webhooks send POST requests to the Cloud Run endpoint, which responds with: (1) plain

text (explanations for metadata); (2) rich content (cards or clickable chips for search results and dataset tools);

The front-end of the assistant is implemented as a static HTML page that embeds the Conversational Messenger web component (see Figure 16). This component (<df-messenger>) provides an interactive chat window that connects directly to the configured Dialogflow CX agent. The agent is rendered as a floating chat bubble in the bottom-right corner of the screen, set via the anchor="bottom-right" attribute. When opened, the chat expands into a full-height conversational window. The Messenger is configured to automatically launch with a predefined intent set to "SHOW_SERVICE_PAGE", which routes the user directly to the agent's Service Page flow, bypassing the need for a greeting or initial message.

```
<df-messenger
  project-id="dataeuropachatbot"
  agent-id="..."
  location="europe-west3"
  language-code="en"
  intent="SHOW_SERVICE_PAGE"
  wait-open="true"
  allow-fullscreen="always"
  anchor="bottom-right">
```

Figure 16. The core <df-messenger> component includes essential configuration parameters.

Project-id and agent-id bind the Messenger to a specific Dialogflow CX agent instance. The intent "SHOW_SERVICE_PAGE" triggers a custom event handler within the Dialogflow flow model, which routes the conversation to a service selection page immediately upon loading. The attribute *wait-open="true"* ensures the chat opens only after the web page has fully loaded. The *allow-fullscreen="always"* attribute enables users to expand the chat to occupy the full screen on smaller devices. *max-query-length="-1*" removes input length restrictions. Regarding the user interface of the chat window, Conversational Messenger has restrictions for formatting the chips, aligning them automatically with the size of the chat window. While waiting for a response, Oggy displays "Oggy is thinking" in the chat to let the user know that the request is in process, as illustrated in Figure 17.

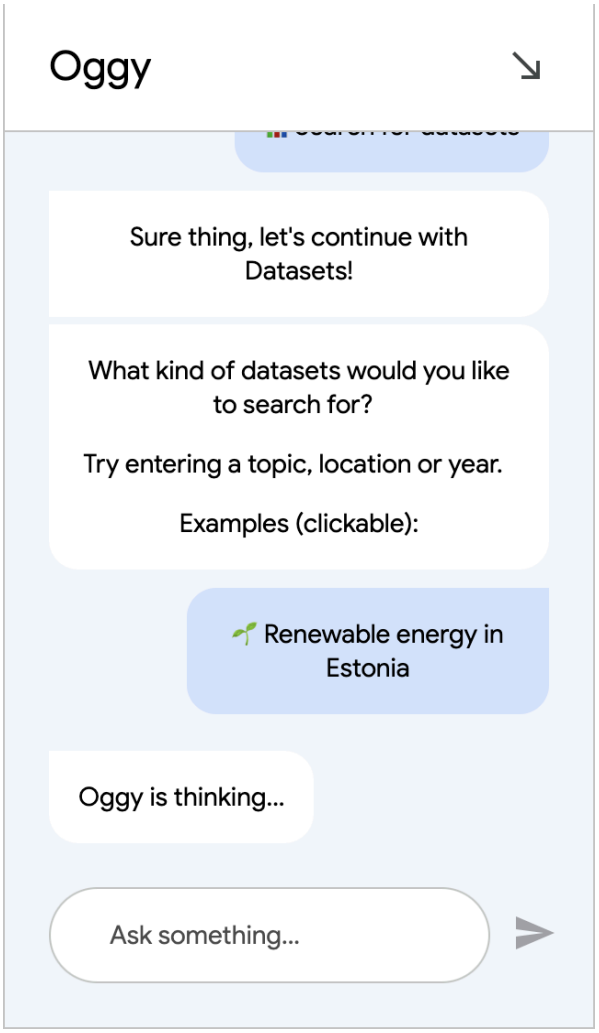


Figure 17. User interface of Oggy searching for datasets in Conversational Messenger. While waiting for a response, “Oggy is thinking” is displayed until the response comes.

4.3 Web Deployment

The deployment of the prototype is divided into three environments: a back-end deployed in Google Cloud Run, source code for the Conversational Messenger embedding versioned and managed using GitHub, and a static front-end hosted using GitHub Pages.

4.3.1 Google Cloud Run

The back-end component of the assistant, including the main webhook logic, deployed as a containerized Node.js application on Google Cloud Run. Cloud Run offers a fully managed, serverless environment where HTTP-based containers can scale automatically based on

incoming requests, making it suitable for hosting the Dialogflow webhook, which needs to respond to multiple tags and intent-based requests from the conversational agent [29].

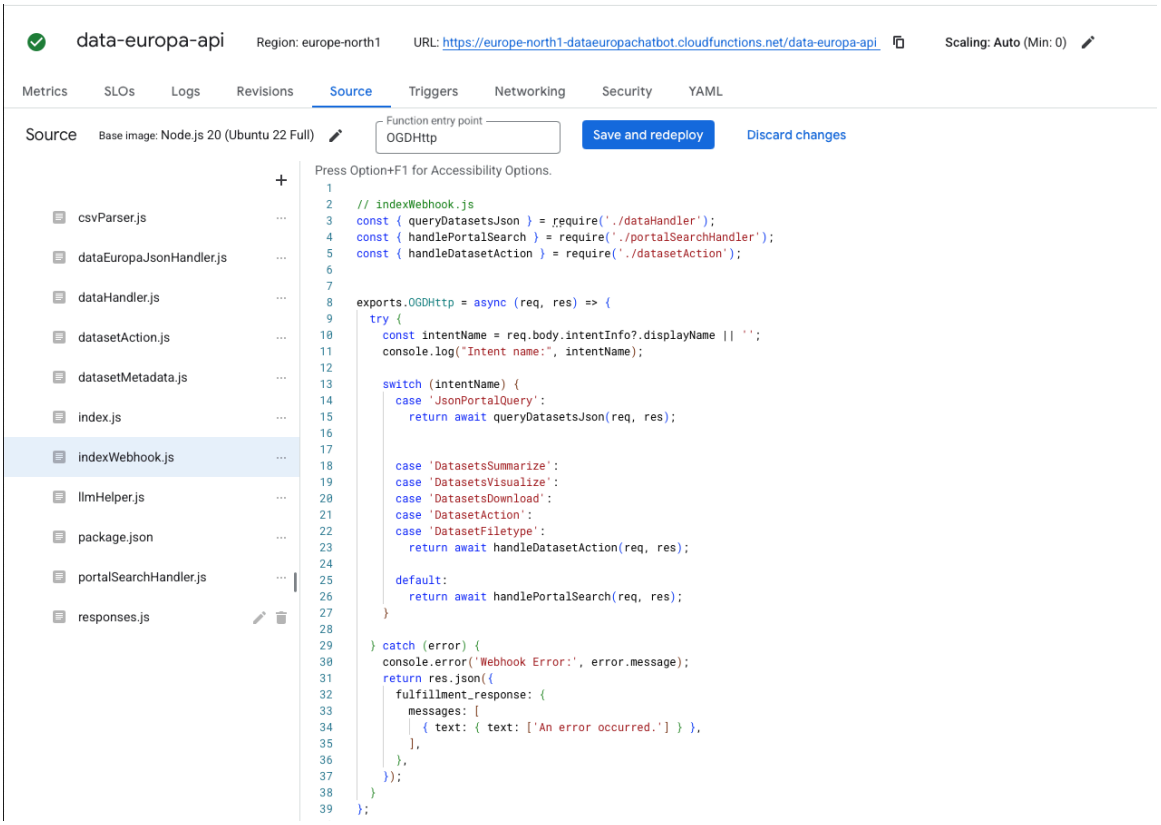


Figure 18. The user interface of the webhook in Google Run.

4.3.2 Github Pages

The front-end of the assistant (see Figure 19), consisting of a single static index.html file, was hosted on GitHub and deployed using GitHub Pages. The HTML file contains the embedded Conversational Messenger component, a welcome message and a tutorial for accessing the assistant. Once the index.html file was committed to the repository, GitHub Pages served it directly from the main branch as a URL - https://carlbogo.github.io/EuropeanDataPortalVirtualAssistant/. Since the Messenger component communicates with the back-end webhook hosted on Google Cloud Run, no server-side logic was required on the GitHub Pages side.

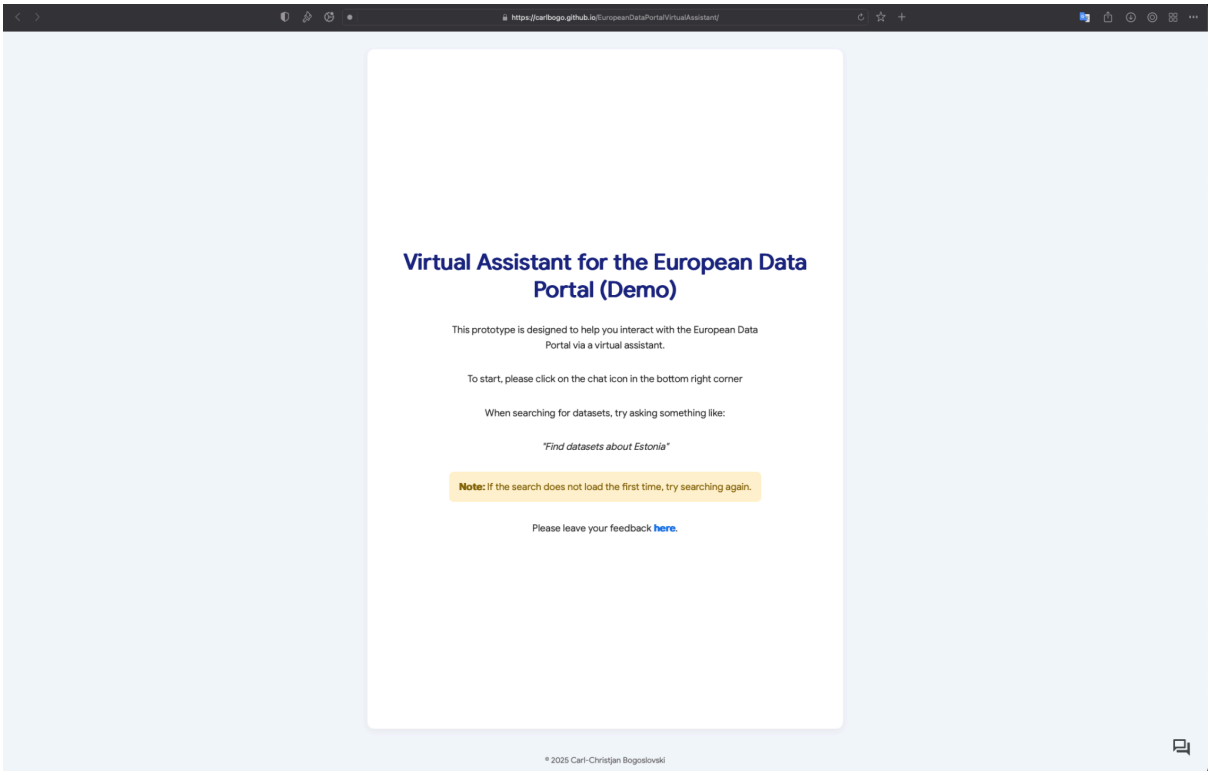


Figure 19. https://carlbogo.github.io/EuropeanDataPortalVirtualAssistant/ is the Github page for the prototype. The assistant pop-up appears in the lower right corner.

4.4 User Interaction Flows

Upon loading the webpage, the user can initiate the assistant by clicking the chat icon located in the bottom-right corner. The virtual assistant, named *Oggy*, is automatically launched and begins the interaction with a welcome message, asking: *"What would you like to do?"* The user is then presented with two primary options: **Search for datasets** or **Search the website**, as seen in Figure 20. These options direct the user into distinct conversational flows tailored to data exploration and website search, respectively

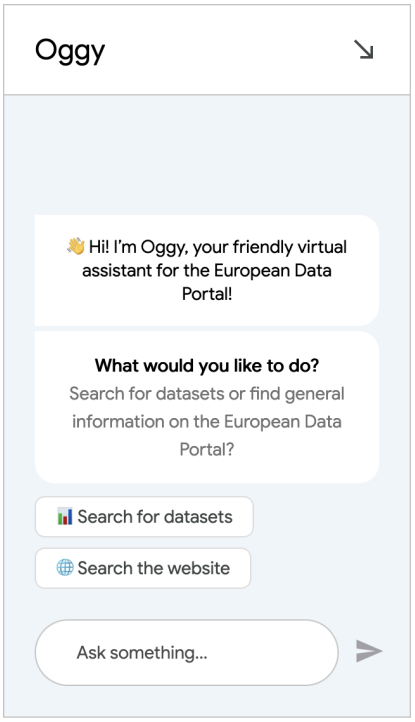


Figure 20. The Service landing page, presenting two options for dialog.

4.4.1 Datasets Conversational Flow

The dataset interaction flow is initiated when the user selects the **"Search for datasets"** option, which opens the Datasets Search page (see Figure 21). This entry point allows users to begin querying datasets of interest via natural language input. To support this process, the assistant (Oggy) displays example prompts as clickable suggestion chips, helping users formulate effective queries.

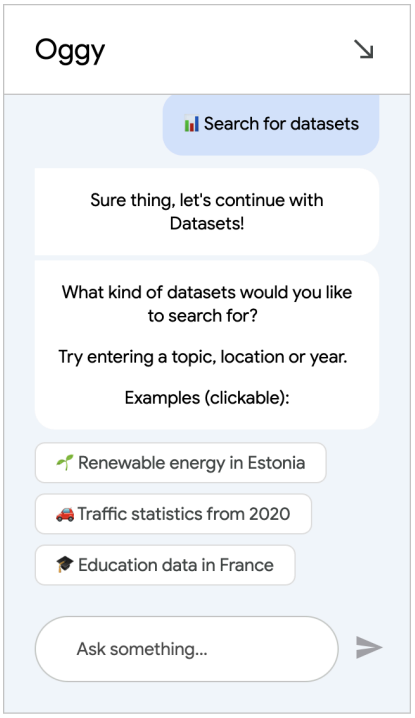


Figure 21. The Datasets Search page. Here the user can type a search question using natural language. The intent is then parsed and keywords are extracted using GPT-3.5.

After the user selects a search chip (e.g. *"Education data in France"*), the assistant extracts structured keywords (see Section 4.1.4). The extracted terms are formatted into a query string and sent via webhook to the European Data Portal API. As illustrated in Figure 22, the assistant responds with matching datasets displayed as cards, saving the datasets for further use and offering follow-up action buttons for *Summarize*, *Download*, and *Visualize* (see Figure 23).

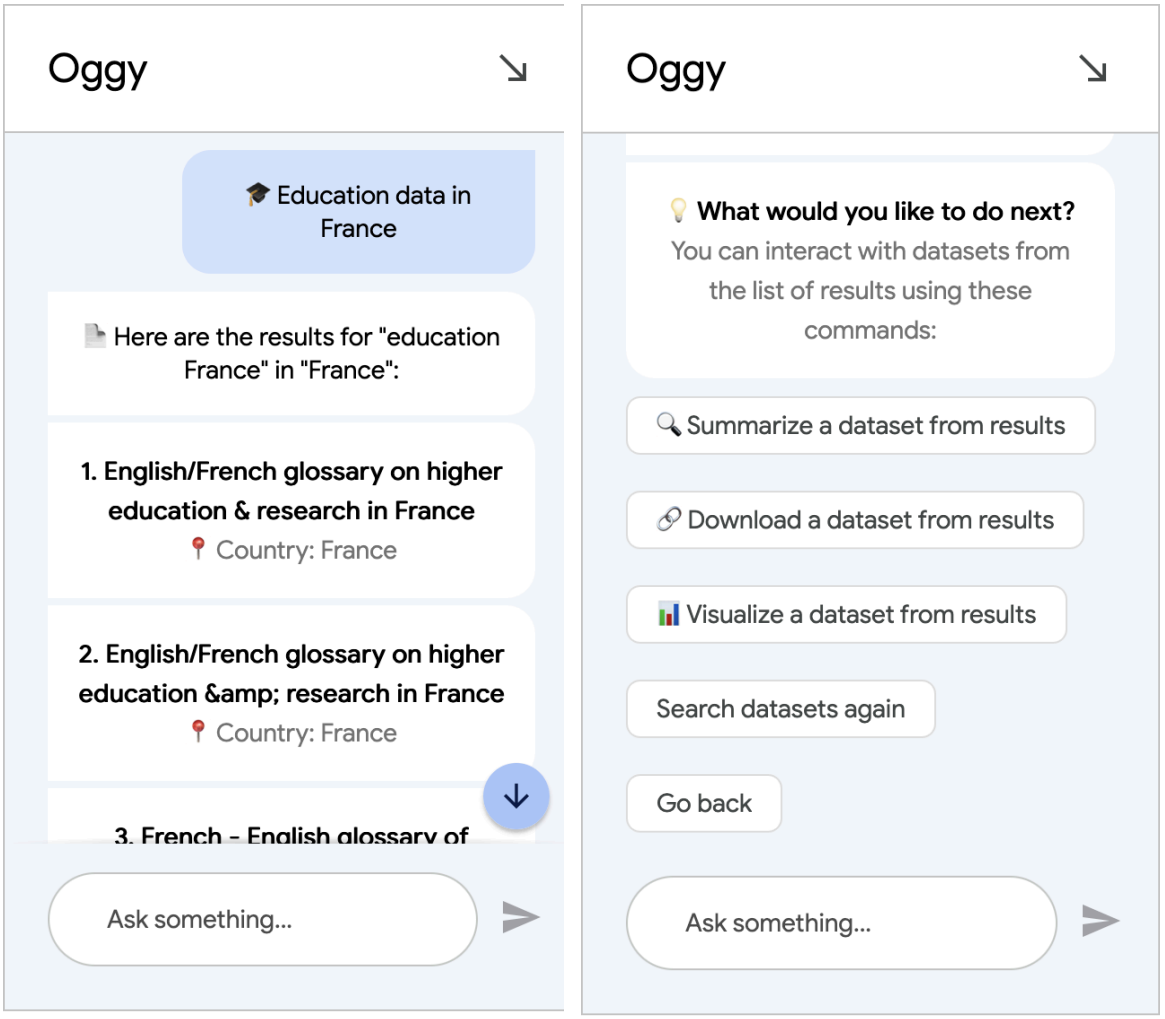


Figure 22 and Figure 23. Example of dataset search using natural language and presented follow-up actions available for interacting with the resulting list of datasets.

When the user chooses to summarize a dataset from the list (see Figure 24), the assistant uses the stored session data to identify the correct dataset object by index. As shown in Figure 25, the metadata fields are passed to a language model via the explainMetadata() function in llmHelper.js (see Section 4.1.4). The LLM returns a summary in natural language, which is then sent back to the user through Dialogflow.

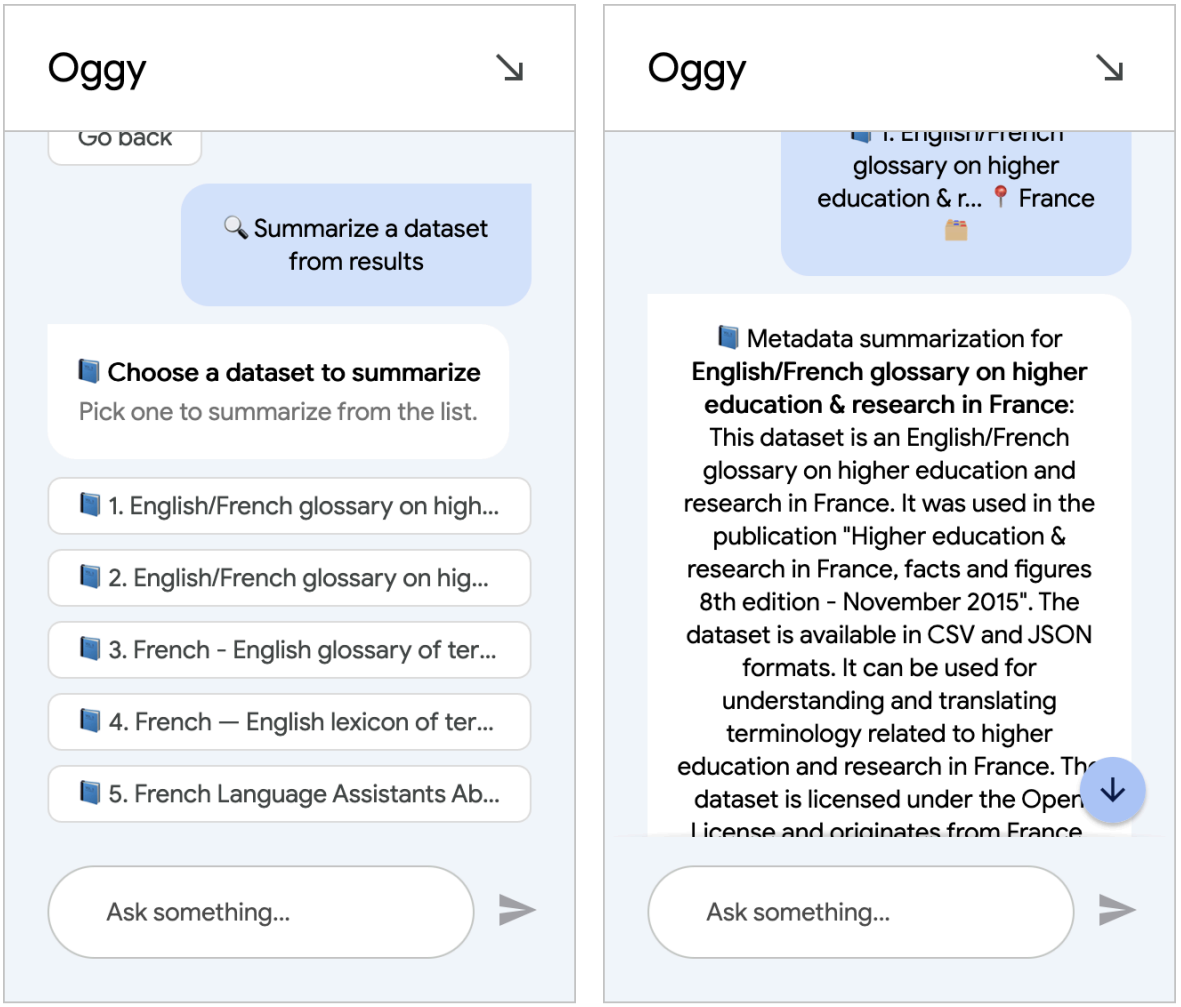
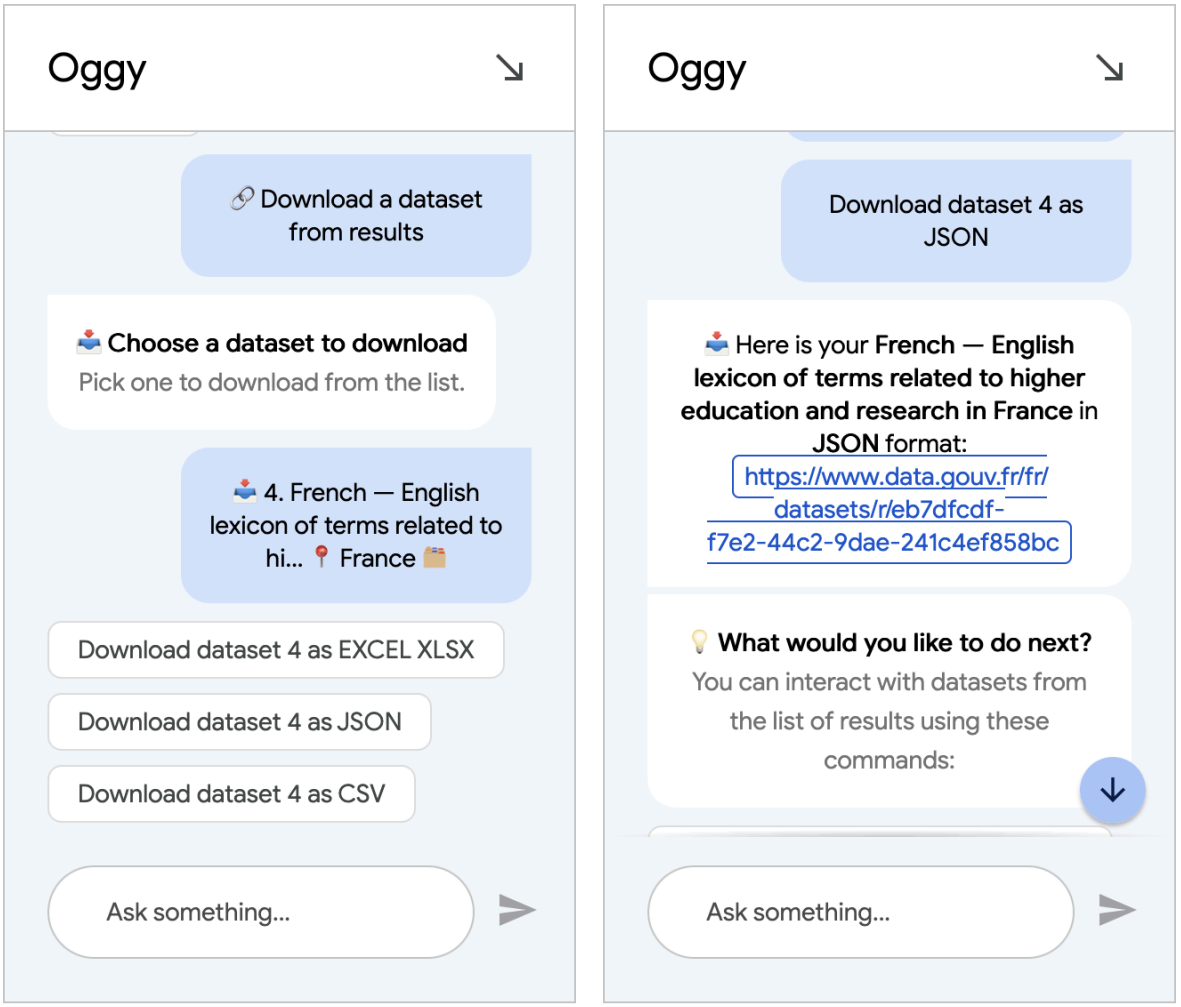


Figure 24 and Figure 25. Example of metadata summarization using LLM and resulting explanation.

In the download flow, the assistant checks which file formats are available for the selected dataset. As seen in Figure 26, it then displays format-specific options (CSV, JSON, XLSX) as chips. Once the user selects a format, the assistant returns a direct download link to the file, which is retrieved from the dataset's resources array in the API response (Figure 27).



Figures 26–27. Dataset download flow and download link returned by the assistant.

When a user requests to visualize a dataset, the back-end attempts to retrieve a CSV file and parse its contents. If no meaningful numeric columns are found, as shown in Figure 28, the assistant informs the user that visualization is not possible. In contrast, if visualization is possible (see Figure 29), the assistant generates a link to a live chart created using the QuickChart API, embedding it as a clickable chip that opens the graph in a new tab.

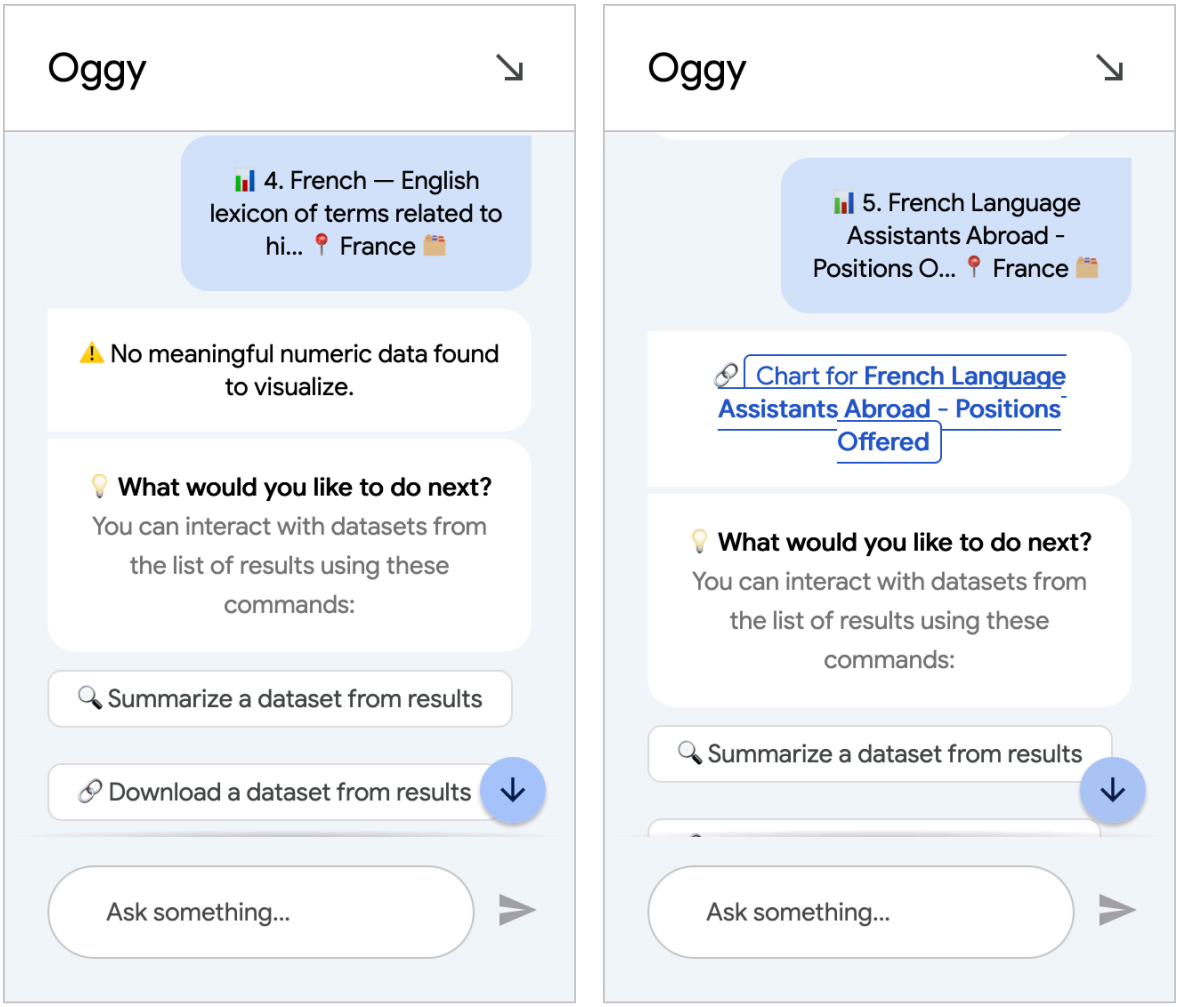


Figure 28 and Figure 29. Visualization tool error handling and output link to the chart.

Figure 30 shows the actual output chart, rendered using QuickChart based on the user's selection of a visualizable dataset. Internally, the back-end parses the CSV file, extracts suitable categorical and numerical columns, and creates a chart configuration JSON. This JSON is then encoded into a URL sent to QuickChart, which generates the final visual representation.

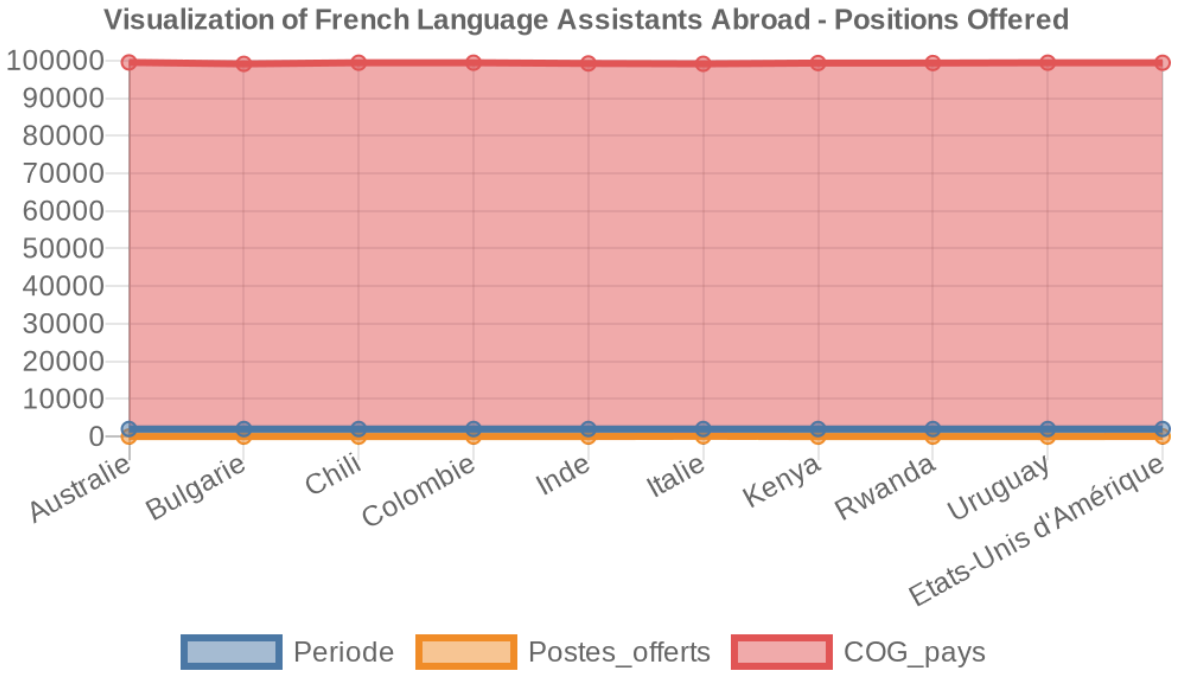


Figure 30. Visualization of dataset using QuickChart. The chart is generated dynamically from parsed CSV data and rendered using an external URL (https://quickchart.io/chart).

Overall, the datasets flow shows how a virtual assistant can be used to simplify user interaction in an OGD portal by combining intent recognition, external API querying, and contextual result management. The following section covers the assistant's other functionality for FAQ-style portal support and website navigation.

4.4.2 Website Conversational Flow

The assistant also supports functionality for helping users find general information about the data portal and its contents. This flow mimics a FAQ-like experience, where the user can ask open-ended questions about the portal's features, licensing, API access, and documentation. Rather than performing dataset-centered actions, this flow uses a custom search engine to return pre-indexed answers from selected sources.

When the user selects "Search the website" on the Service Page, they are routed to the Website flow (see Figure 21). There, the assistant prompts them with guidance and clickable example chips, designed to represent expected queries about the platform.

In the example in Figure 31, the user selects “What is open data?” as a question. This is a predefined chip, designed as an expected website-search intent. The assistant responds by returning a list of short snippets and titles.

The results (see Figure 32) include the title, source, and a short extract, followed by a “View more” link. These responses are generated dynamically via webhook, using the portalSearchHandler.js module and a lightweight formatting system in responses.js.

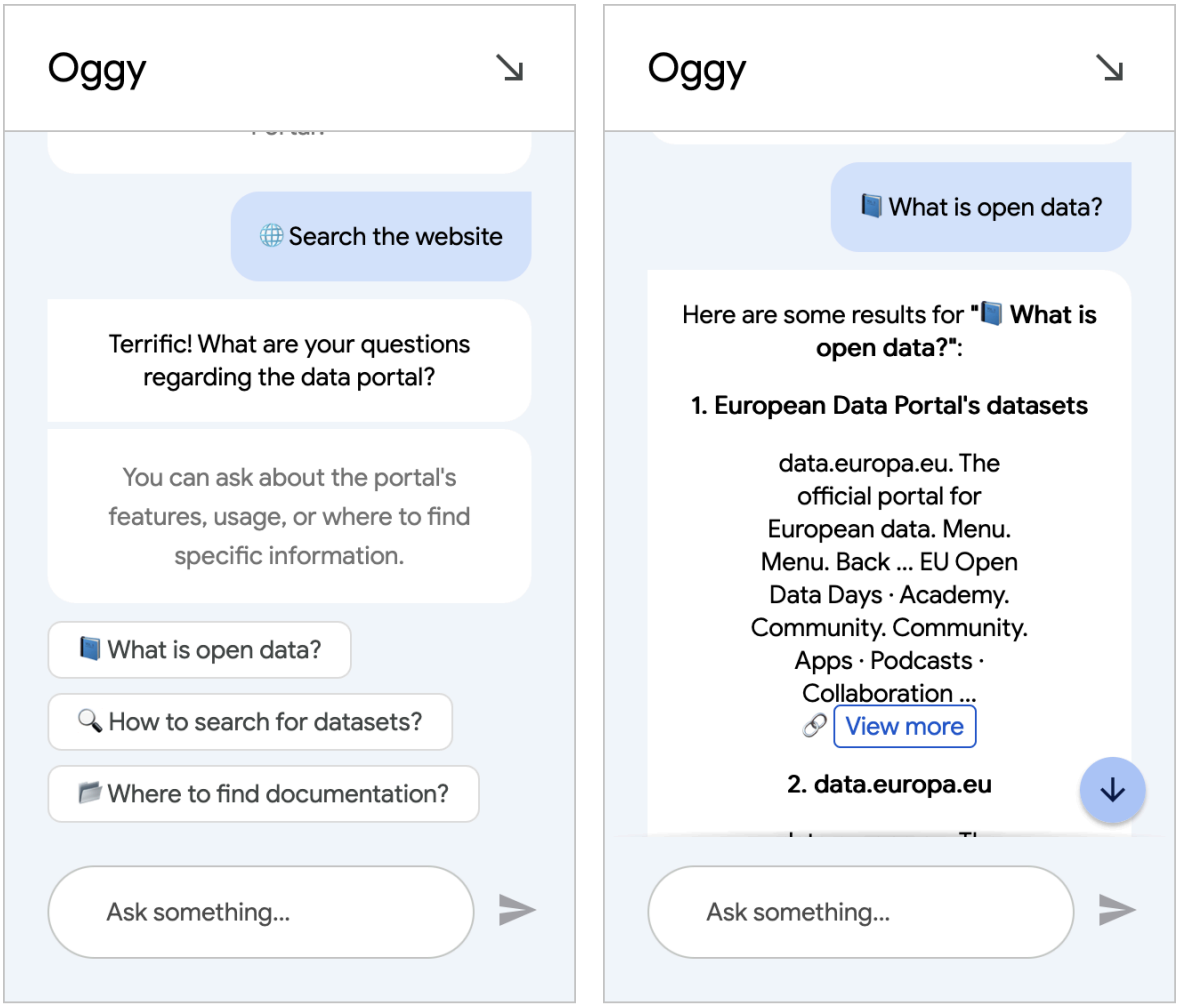


Figure 31 and Figure 32. Transition from the Service/Welcome page to Website Search and asking general questions about the portal

Two action chips are introduced after the search results list - for returning to the starting Service page (“Go back”) and continuing to search the website (“Search the website”) (see Figure 33). The user then selects a second example: “Where can I find documentation about APIs” This is a free-form query that triggers the same search intent. The back-end reformulates the question into search engine–friendly keywords using refineSearchQuery() in

llmHelper.js, then sends the query to the Google API to search the European Data Portal website.

As shown in Figure 34, the assistant returns documentation-related links and snippets, such as licensing information, source code details, and developer tools like CKAN endpoints. Each result is displayed as a simple card with a clickable link to the found resource.

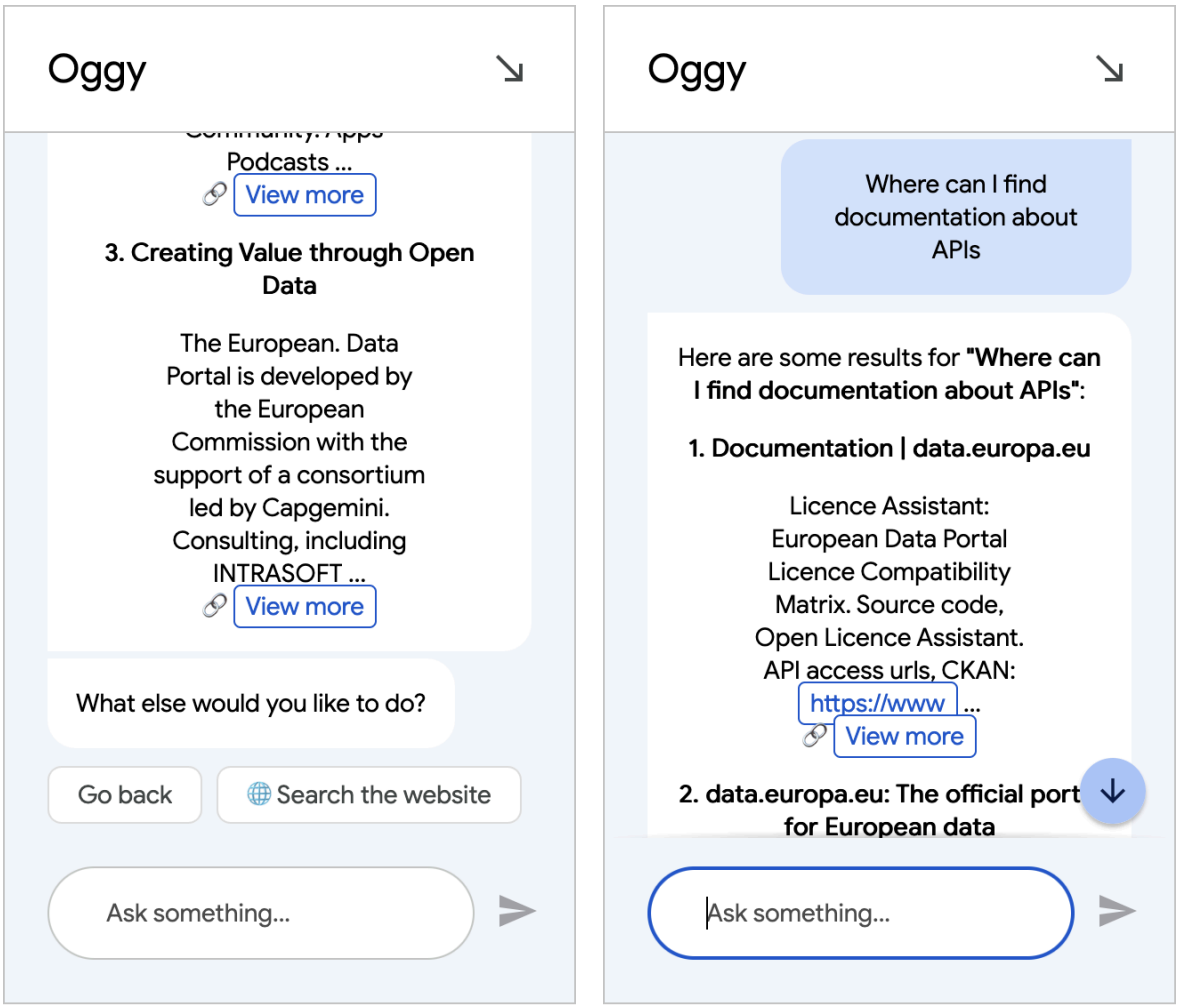


Figure 33 and Figure 34. Finding documentation and developer information

When testing the virtual assistant, the described flows performed as expected. To analyze the assistant's usability, the next section provides a detailed description of the evaluation and user comments

5. Evaluation

This section presents the evaluation of the implemented assistant prototype based on survey results and user feedback collected during live testing. The section reflects on the assistant's usability, relevance, and effectiveness in supporting interaction with the European Data Portal.

5.1. Survey Results

A total of 68 responses were collected during the active survey period from 16 February 2025 to 30 March 2025 (a duration of six weeks), with an average completion time of 18 minutes. The survey received responses from a diverse international audience, with the most represented countries being Estonia (9), Canada (7), and Greece (3). Additional responses came from over 30 countries across Europe, the Americas, Asia, and Africa.

The age group distribution of the respondents:

- 45–54: 22 responses (32.35%)
- 25–34: 15 responses (22.06%)
- 35–44: 13 responses (19.12%)
- 18–24: 9 responses (13.24%)
- 55+: 9 responses (13.24%)

Occupationally, the largest respondent groups included public sector employees (20), researchers or academics (11), and civil society members (11). The remaining responses were from students, citizens, IT professionals, journalists, and other digital or policy-related professionals.

Respondents reported varying levels of engagement with open government data portals. The most common usage frequency was weekly (27.94%, 19 responses), followed by daily and seasonal use, each accounting for 22.06% (15 responses). A slightly smaller group accessed portals on a monthly basis (20.59%, 14 responses). Only 7.35% of respondents (5 individuals) indicated that they had never used such portals (see Figure 35).

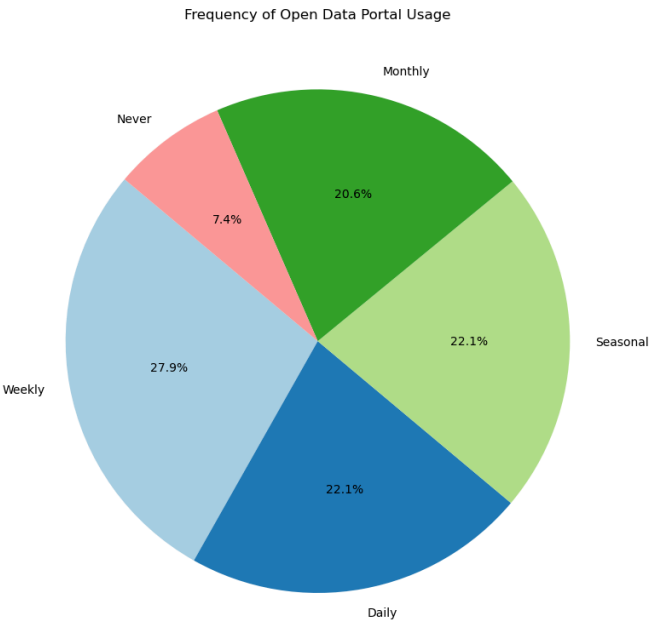


Figure 35. Frequency of OGD portal usage

The majority of survey respondents (59 out of 68) reported prior experience using virtual assistants, primarily in customer support or search contexts. However, only 4 respondents had encountered a virtual assistant within an OGD portal. 54 had never encountered a virtual assistant in OGD context, and 10 were unsure. These findings revealed that although virtual assistants are familiar to most users, they are largely absent in the context of OGD portals. When asked whether a virtual assistant would improve their OGD portal experience, 57 participants agreed, 4 disagreed, and 7 were uncertain. The aspects are illustrated in Figure 36.

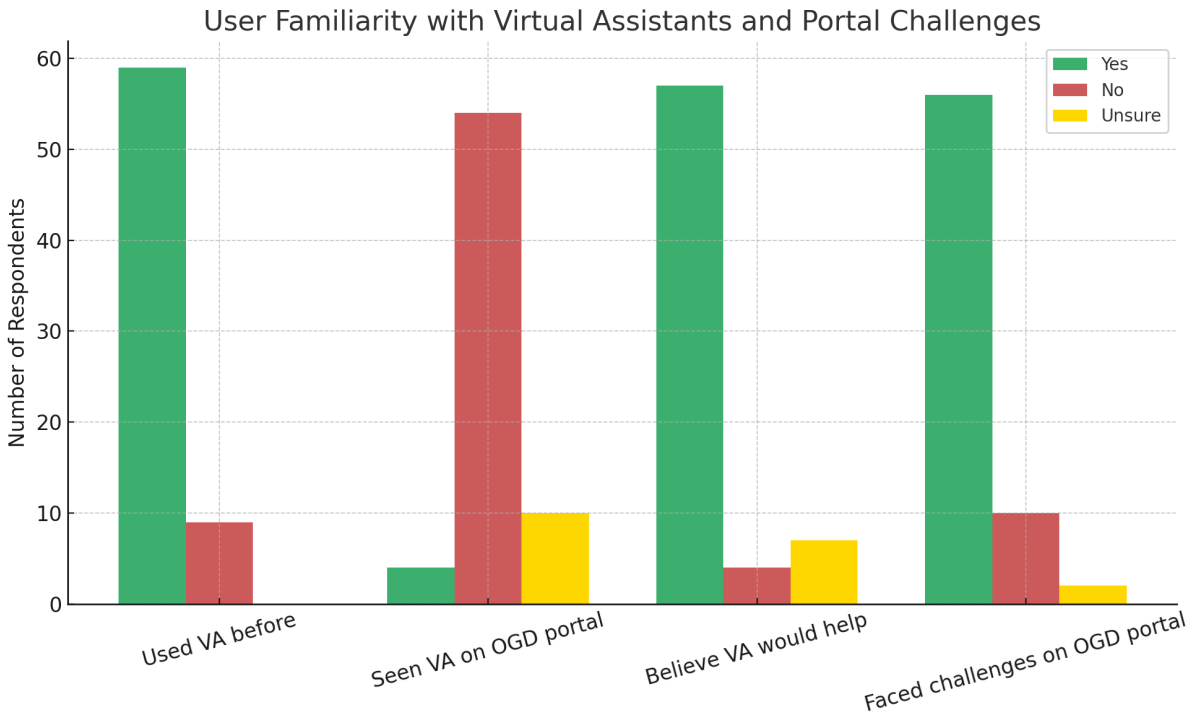


Figure 36. Chart of responses regarding user familiarity with virtual assistants, and challenges in OGD portals

56 respondents acknowledged having encountered challenges while using OGD portals, while 10 reported no such issues and 2 had no experience with such platforms. The most common difficulties related to finding relevant datasets, interpreting metadata, and navigating complex or incomplete data, as shown in Figure X.

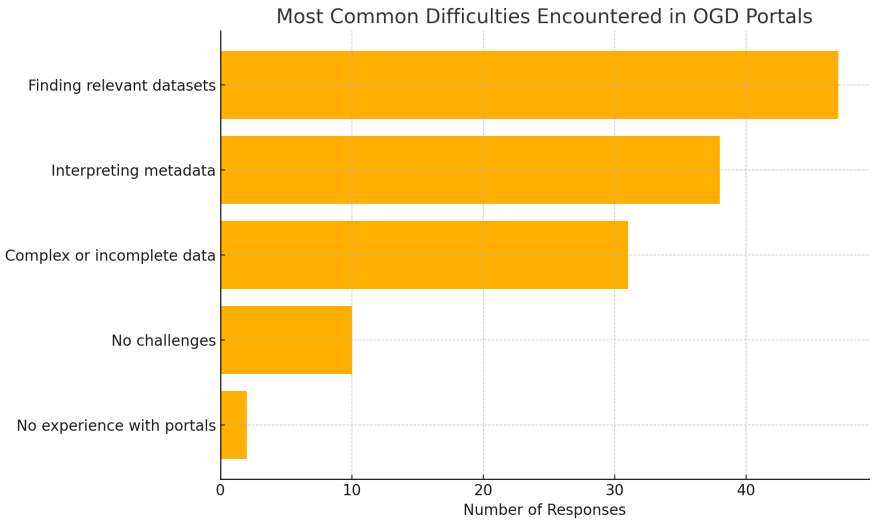


Figure 37. Most common difficulties encountered in OGD portals

The highest-rated functionality was the ability to find datasets based on a topic of interest, with an average score of **4.34**, followed by handling specific queries (e.g., "*show datasets comparing energy use*") with **4.28**. Other highly rated features included generating summaries or visualisations (**4.19**), easy-to-read fonts and text sizes (4.18), and a minimalistic interface (4.15). From a design perspective, participants positively rated "*clean and minimalistic interface"* (**4.15**), "*easy-to-read text"* (**4.18**) and "*responsive layout for both desktop and mobile"* (**4.01**). Slightly lower, but still favourable, scores were observed for visual support elements such as *interactive buttons and menus* (**3.97**) and *icons or highlights to clarify responses* (**3.60**).

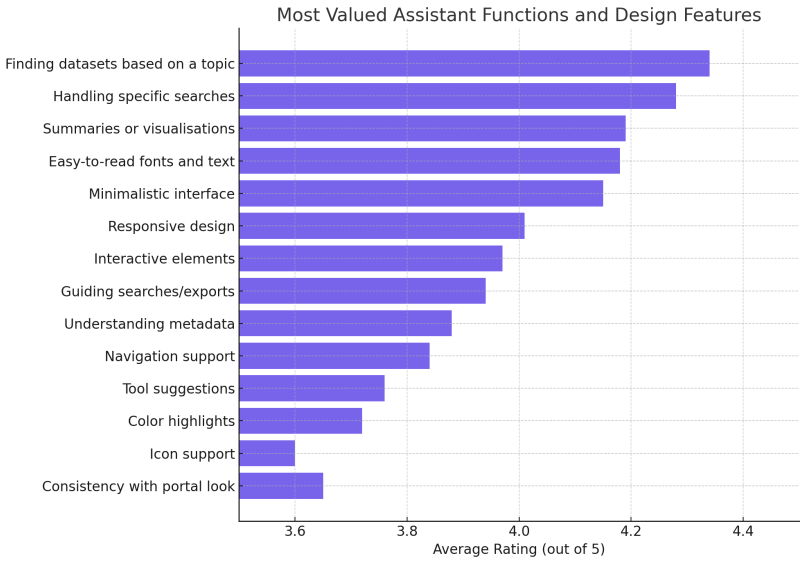


Figure 38. Most valued assistant functions and design features

These results suggest that users value both functional capability and interface clarity, particularly when interacting with complex datasets. The assistant's final structure and priorities were shaped by these findings, with an emphasis on natural language search, metadata support, and a clean user interface. To evaluate how well these expectations were met in practice, the next section presents an assessment of the prototype through user testing and interaction feedback.

5.2 Evaluation Survey Results.

To evaluate the deployed prototype, a follow-up feedback survey was distributed and completed by 15 participants. The aim of this evaluation was to assess the assistant's

usability, functional effectiveness, and perceived value in supporting users on an open data portal.

Usability ratings indicate that the assistant was generally perceived as user-friendly. 80% of respondents rated the assistant's ease of use as either 4 or 5 on a 5-point Likert scale, suggesting that the interface and interaction flow were intuitive for most users.

Perceived task success, however, was notably lower. Only 1 out of 15 users answered "Yes" when asked if they found the desired information. The majority (9 out of 15) selected "Partially," and the remaining 5 answered "No" (see Figure 39). This discrepancy suggests that while the interface was approachable, the assistant's ability to retrieve complete and relevant results was inconsistent.

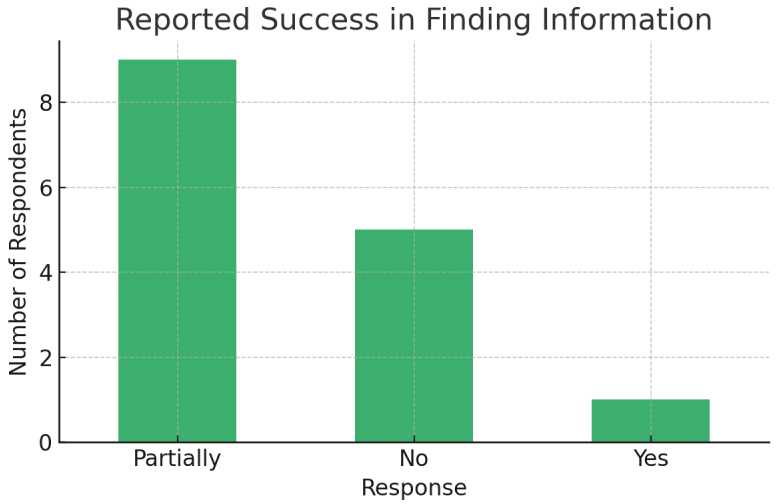


Figure 39. Reported success in finding information

The most commonly used functionality during testing was dataset search, which was used alone by 9 participants and in combination with website search by 5 more. Only one user used website search exclusively (see Figure 40). This result aligns with the initial findings from the user survey, where dataset discovery was identified as a more valued feature.

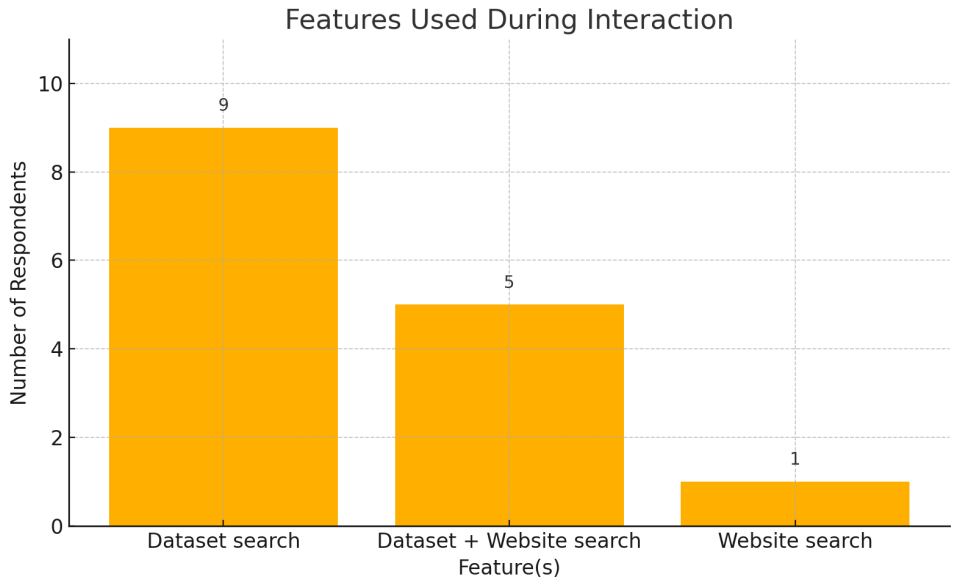


Figure 40. Chart of interacted assistant features

The respondents made several suggestions for improvement and reported various issues. Suggested areas for improvements were: (1) providing clearer and more informative error messages; (2) improving natural language intent recognition, especially for paraphrased or loosely structured queries; (3) enhancing the conversational flow to allow backtracking and query reformulation; (4) incorporating system responses that acknowledge incomplete queries or prompt users for clarification; (5) adding feedback before visualisation to validate whether data is suitable for plotting.

In terms of technical and functional issues, participants reported the following problems: (1) the assistant sometimes failed to respond after a single query; (2) duplicate datasets appeared in response lists; (3) back-end timeouts occurred under some flows; (4) unexpected chatbot resets interrupted interaction.

These findings suggest that, while the assistant's interface meets expectations for clarity and navigation, intent matching, failed replies, and timeouts require improvement before enabling more complicated or less structured user inputs. The following chapter discusses these findings in relation to the original research goals, highlighting the feedback results, the limitations of the current implementation, and opportunities for future development.

6. Discussion

This section reflects on the overall outcomes of the development process, discussing the results of the feedback survey, the current limitations of the prototype and future improvements.

6.1 Feedback and Current Limitations

User feedback collected through the evaluation survey (see Section 5.2) confirmed that the assistant interface was intuitive and generally easy to use. 80% of participants rated usability either 4 or 5 out of 5 points, and dataset search was the most commonly used functionality. However, users also indicated that while the interface was clear, it did not always lead to successful task completion. Only one user reported fully finding what they were looking for, while nine out of fifteen reported partial success.

Despite a high usability rating of the system, several functional and technical limitations were observed: (1) **incomplete search results** - users frequently reported receiving partial or irrelevant results, particularly for more detailed queries, which mention multiple keywords; (2) **system instability** - the assistant occasionally failed to respond or reset unexpectedly during conversations; (3) **lack of error guidance** - when no results were found, users were not provided with clear feedback or suggestions on how to adjust their queries; (4) **visualisation issues** - some datasets lacked numerical data necessary for generating meaningful visualizations, which led to failed attempts or user confusion; (5) **no conversation memory** - the assistant did not retain context across multi-turn interactions, limiting ability to handle follow-up queries.

These limitations suggest that while the virtual assistant is a promising tool for guiding users in open data environments, refinement is necessary in order for it to be successfully adopted. Based on the obtained feedback, an improvement agenda has been defined for enhancing the assistant's functionality presented in the next section.

6.2 Future Improvements

Based on user feedback and technical observations, several improvements can be implemented to increase assistant usability. One key area is the need to improve request

stability and reduce timeouts. The virtual assistant prototype was designed to use LLM computing to process and provide search results. Due to being a compute heavy operation, the use of LLMs may have caused instability regarding responses. Users occasionally encountered situations where the assistant failed to return results due to delays in processing or external API failures. Implementing back-end optimisations, such as asynchronous handling, retry logic, and better error management, could significantly increase reliability.

Another priority is introducing validation before visualisation. In the current implementation, users can request visualisation of datasets without knowing whether the data is actually suitable for plotting. Although including relevant error handling, a pre-check mechanism could alert the user if the dataset lacks numerical or structured fields and suggest alternatives.

Enhancing intent recognition is also essential. While the assistant performs well with straightforward queries, more complex or incomplete questions often lead to fallback responses. Integrating fine-tuned language models to handle inputs unfamiliar to the assistant or manually adding user intents would allow the system to interpret a wider variety of user inputs. In addition, enabling multi-turn memory would allow the assistant to retain context from previous exchanges. This would make it possible to support follow-up questions, clarify ambiguous queries, and maintain a more coherent conversational flow. Users also noted a desire for more detailed and informative responses. This includes presenting richer metadata, offering contextual advice (e.g. on dataset formats), and incorporating suggestions for next steps.

The assistant currently operates only in English, even though the European Data Portal contains multilingual data. Future versions could incorporate language detection and translation services to allow interaction in multiple languages to broaden accessibility. Implementing these enhancements could potentially improve the assistant's reliability and expand its practical application in multilingual and multi-domain contexts.

7. Conclusion

This thesis explored whether a conversational virtual assistant could support users in interacting with OGD portals more effectively. Through a combination of user research, iterative prototyping, and real-world evaluation, the project demonstrated both the promise and current limitations of such a solution.

The study began with a structured survey aimed at identifying the issues of OGD portals, establishing whether a virtual assistant could be a viable solution for these issues and obtaining user expectations for functionality. The results showed strong interest in natural language search, dataset summarisation, and simplified access to downloadable content. Based on these findings, a functional prototype was implemented using Dialogflow CX, Node.js, and the European Data Portal API. The assistant, Oggy, was designed to support tasks such as dataset discovery, visualisation, and metadata explanation through a conversational interface.

User feedback on the deployed prototype revealed that the assistant was generally intuitive and easy to use, with 80% of participants rating its usability 4 or 5 out of 5. However, only one-third of users fully achieved their search goals, highlighting issues related to natural language understanding and service stability. Although suggesting the value of conversational interfaces, it also underscores the technical and linguistic complexities of delivering consistent assistant performance.

The contributions of this thesis include a modular virtual assistant prototype integrated with a live open data API following a user-informed design.

The limitations of this work include a small respondents sample size in both phases of the survey, limited capabilities for dynamic user inputs, and the prototype's exclusive support for English. Future work should focus on improving stability, enriching conversational memory, expanding multilingual capabilities, and integrating more advanced reasoning and validation mechanisms.

By lowering the entry barrier to complex data portals, conversational assistants such as the virtual assistant developed in the course of this thesis offer a promising opportunity for increasing open data accessibility and engagement, especially for non-technical users and new audiences.

References

[1] Link Digital, “Artificial Intelligence and Open Data – a Mutually Beneficial Interaction,” [Online]. Available: https://linkdigital.com.au/news/2025/01/artificial-intelligence-and-open-data-a-mutually-beneficial-interaction. [Accessed: Oct. 28, 2024].

[2] European Union, “What is open data?” [Online]. Available: https://data.europa.eu/en/dataeuropa-academy/what-open-data. [Accessed: Oct. 28, 2024].

[3] S. Sheoran, S. Mohanasundaram, R. Kasilingam, and S. Vij, “Usability and Accessibility of Open Government Data Portals of Countries Worldwide: An Application of TOPSIS and Entropy Weight Method,” *Int. J. Electron. Gov. Res.*, vol. 19, pp. 1–25, 2023. [Online]. Available: https://doi.org/10.4018/IJEGR.322307

[4] Riigi Infosüsteemi Amet, “Eesti avaandmete portaal,” [Online]. Available: https://avaandmed.eesti.ee. [Accessed: Oct. 28, 2024].

[5] European Union, “EU open data portal,” [Online]. Available: https://data.europa.eu. [Accessed: Oct. 28, 2024].

[6] F. Xiao, D. He, Y. Chi, W. Jeng, and C. Tomer, “Challenges and supports for accessing open government datasets: Data guide for better open data access and uses,” in *Proc. 2019 Conf. Human Inf. Interact. Retrieval*, pp. 313–317. ACM, 2019. [Online]. Available: https://doi.org/10.1145/3295750.3298958

[7] M. E. Cortés-Cediel, A. Segura-Tinoco, I. Cantador, and M. P. Rodríguez Bolívar, “Trends and challenges of e-government chatbots: Advances in exploring open government data and citizen participation content,” *Gov. Inf. Q.*, vol. 40, no. 4, p. 101877, 2023. [Online]. Available: https://doi.org/10.1016/j.giq.2023.101877

[8] M. D. Wilkinson et al., “The FAIR guiding principles for scientific data management and stewardship,” *Sci. Data*, vol. 3, no. 1, pp. 1–9, 2016. [Online]. Available: https://doi.org/10.1038/sdata.2016.18

[9] M. E. Mamalis, E. Kalampokis, A. Karamanou, P. Brimos, and K. Tarabanis, “Can Large Language Models Revolutionalize Open Government Data Portals? A Case of Using

ChatGPT in statistics.gov.scot," in *Proc. 27th Pan-Hellenic Conf. Progress in Comput. Informatics (PCI 2023)*, Lamia, Greece, Nov. 2023. ACM. [Online]. Available: https://doi.org/10.1145/3635059.3635068

[10] R. Bommasani et al., "On the opportunities and risks of foundation models," *arXiv preprint*, arXiv:2007.10503, 2020. [Online]. Available: https://arxiv.org/abs/2007.10503

[11] European Commission, "Open data portals," [Online]. Available: https://digital-strategy.ec.europa.eu/en/policies/open-data-portals. [Accessed: Oct. 28, 2024].

[12] IBM, "Types of chatbots and how they work," *IBM Think Blog*, 2023. [Online]. Available: https://www.ibm.com/think/topics/chatbot-types

[13] D. Wang, D. Richards, A. A. Bilgin, and C. Chen, "Implementation of a conversational virtual assistant for open government data portal: Effects on citizens," *J. Inf. Sci.*, pp. 1–21, 2023. [Online]. Available: https://doi.org/10.1177/01655515221151140

[14] Datafloq, "Six Use Cases of Conversational AI in Public Sector Governance," [Online]. Available: https://datafloq.com/read/six-use-cases-of-conversational-ai-in-public-sector-governance/. [Accessed: May 10, 2025].

[15] Publications Office of the European Union, "Publio – the Publications Office virtual assistant," [Online]. Available: https://op.europa.eu/en/web/webtools/publio-the-publications-office-virtual-assistant

[16] Kratid, "Bürokratiga liitumine," [Online]. Available: https://www.kratid.ee/burokratiga-liitumine

[17] Brønnøysundregistrene, "Norwegian National Data Catalog," [Online]. Available: https://data.norge.no

[18] D. A. Dillman, J. D. Smyth, and L. M. Christian, *Internet, Mail, and Mixed-Mode Surveys: The Tailored Design Method*, 3rd ed. Wiley, 2009.

[19] Rasa, "Rasa Open Source Platform Overview," [Online]. Available: https://rasa.com/docs/rasa/

[20] Google Cloud, “Conversational Agents (Dialogflow CX) documentation,” [Online]. Available: https://cloud.google.com/dialogflow/cx/docs. [Accessed: Oct. 28, 2024].

[21] Google Cloud, “Dialogflow editions,” [Online]. Available: https://cloud.google.com/dialogflow/docs/editions. [Accessed: Oct. 28, 2024].

[22] Node.js, “Node.js Overview,” [Online]. Available: https://nodejs.org/en/docs

[23] Express.js, “Express Web Framework,” [Online]. Available: https://expressjs.com

[24] Axios, “Axios HTTP Client Library Overview,” [Online]. Available: https://axios-http.com/docs/intro. [Accessed: May 6, 2025].

[25] European Data Portal, “API Documentation,” [Online]. Available: https://data.europa.eu/en/which-apis-are-available-and-where-can-i-find-information-about-them. [Accessed: May 6, 2025].

[26] OpenAI, “GPT-3.5 Model Overview,” [Online]. Available: https://platform.openai.com/docs/models/gpt-3-5

[27] Google, “Custom Search API Overview,” [Online]. Available: https://developers.google.com/custom-search/v1/overview. [Accessed: May 6, 2025].

[28] Open Knowledge Foundation, “CKAN – The open source data management system,” [Online]. Available: https://ckan.org

[29] Google Cloud, “Cloud Run documentation,” [Online]. Available: https://cloud.google.com/run/docs/

[30] European Union, “Research on the potential of virtual assistants in governmental open data portals,” [Online]. Available: https://data.europa.eu/en/news-events/news/research-potential-virtual-assistants-governmental-open-data-portals

[31] T. Makasi, A. Nili, K. C. Desouza, and M. Tate, “A typology of chatbots in public service delivery,” *IEEE Software*, vol. 39, no. 1, pp. 58–66, 2021. [Online]. Available: https://doi.org/10.1109/MS.2021.3116101

[32] K. K. Nirala, N. K. Singh, and V. S. Purani, “A survey on providing customer and public administration based services using AI: chatbot,” *Multimedia Tools and Applications*, vol. 81, pp. 22215–22246, 2022. [Online]. Available: https://doi.org/10.1007/s11042-021-11458-y

[33] P. Ramires Hernández, D. Valle-Cruz, and R. V. Méndez Mendoza, “Review on the application of artificial intelligence-based chatbots in public administration,” in *Handbook of Research on Applied Artificial Intelligence and Robotics for Government Processes*, M. R. Pérez and R. A. García, Eds. Hershey, PA: IGI Global, 2023, pp. 133–155. [Online]. Available: https://doi.org/10.4018/978-1-6684-5624-8.ch007

[34] OpenAI, “ChatGPT,” OpenAI, San Francisco, CA, USA, 2024. [Online]. Available: https://chat.openai.com

Appendix

I. Prototype Source Code

The prototype source code is available on GitHub:
https://github.com/carlbogo/EuropeanDataPortalVirtualAssistant.git

II. Prototype Application

The prototype application is available via https://carlbogo.github.io/EuropeanDataPortalVirtualAssistant/

III. Preliminary Survey

Virtual Assistants in Open Government Data Portals

Introduction

The aim of this survey is to gather insights about user interactions with virtual assistants for open government data (OGD) portals, in order to inform the development of a virtual assistant prototype for the European Open Data Portal. The survey takes approximately 15–20 minutes to complete. All responses are anonymous.

Section 1 – Background Information

1. Country of Origin *(Short answer)*

2. What is your age group? *(Multiple choice)*

- Under 18
- 18–24
- 25–34
- 35–44

- 45–54
- 55+

3. Have you ever used a chatbot or virtual assistant (not necessarily on an OGD portal)?

- Yes
- No

4. If yes, for what purpose(s)? *(Select up to 3)*

- Website navigation help
- Booking or scheduling services
- Policy/legal guidance
- Dataset or information search
- Other: _____

5. What was your overall experience with such assistants? *(Rating scale)*

Section 2 – Using Open Government Data Portals

6. What best describes you in your use of OGD portals? *(Select one)*

- Student
- Researcher/Academic

- Data Analyst
- Developer/IT Professional
- Public Sector Employee
- Journalist
- Citizen
- Member of Civil Society Organization
- I have not used an OGD portal before
- Other: _____

7. In general, how frequently do you use OGD portals?

- Daily
- Weekly
- Monthly
- Seasonally
- Never

8. Which types of OGD portals have you accessed? *(Select all that apply)*

- Local/municipal

- National (e.g., Data.gov, Gov.UK)
- International/supranational (e.g., EU Open Data Portal)
- Statistical bureaus (e.g., Eurostat)
- I have never used an OGD portal
- Other: _____

9. What is your primary purpose for using OGD portals? *(Select all that apply)*

- Civic engagement
- Data analysis & research
- Policy development
- NGO/social impact
- Business/innovation
- Urban planning/smart cities
- Other: _____

10. Have you encountered challenges using OGD portals?

- Yes
- No

- I have not used an OGD portal

11. What were the main challenges? *(Select all that apply)*

- Difficulty finding datasets
- Too much data
- Incomplete datasets
- Complex datasets
- Difficulty interpreting metadata
- Other: _____

Section 3 – Virtual Assistant Interactions

12. Have you come across a VA while using an OGD portal?

- Yes
- No
- Maybe
- Unsure

13. Would a VA improve your experience on such portals?

- Yes

- No
- Not sure

14. What features would you expect from such a VA? *(Select all that apply)*

- Portal navigation help
- Topic-based dataset search
- Suggestions for dataset use
- Related dataset suggestions
- Dataset summarisation/visualisation
- Filter/query assistance
- Metadata understanding
- I have not used an OGD portal
- Other: _____

Section 4 – Virtual Assistant Attributes

How important are the following features? (1 = Not important, 5 = Very important)

15. Helping navigate the portal

16. Finding datasets based on a topic of interest

17. Guiding user through searches/exports

18. Handling specific searches (e.g., "Compare energy usage datasets")

19. Suggesting tools or visualisation types

20. Helping interpret metadata

Section 5 – Visual Design

How important are the following aspects of the VA's interface design? (1 = Not important, 5 = Very important)

21. A clean and minimalistic interface

22. Interactive buttons or menus

23. Design consistency with the portal

24. Easy-to-read fonts and text sizes

25. Use of color to highlight key information

26. Responsive design (desktop/mobile)

27. Use of icons or visuals for clarity

IV. Feedback Survey

Introduction

The following feedback survey was used to evaluate the user experience and functionality of the virtual assistant prototype developed for the European Open Data Portal. The survey was created and distributed via Microsoft Forms.

Title: *Interactive Virtual Assistant "Oggy"*
Purpose: Feedback for a virtual assistant prototype intended for the European Open Data Portal

Survey questions:

1. **What is your age group?**
    - 18–24

- 25–34
- 35–44
- 45–54
- 55–64

2. **Country of residence** *(open-ended)*

3. **Profession or area of activity** *(open-ended)*

4. **How easy was it to use the virtual assistant?**
   (1 = Very difficult, 5 = Very easy)
   - 1 / 2 / 3 / 4 / 5

5. **Did you find the information you were looking for?**
   - Yes
   - Partially
   - No

6. **Which features did you use?**
   - Dataset search
   - Website search

7. **What features or improvements would you like to see?** *(open-ended)*

8. **Did you experience any issues or errors? If so, what kind?** *(open-ended)*

9. **Overall, how satisfied are you with the virtual assistant?**
   (1 = Very dissatisfied, 5 = Very satisfied)

    - 1 / 2 / 3 / 4 / 5

10. **Would you be interested in participating in future testing?**

- Yes

- No

11. **If yes, leave your email** *(optional, open-ended)*

License

I, Carl-Christjan Bogoslovski,

1. grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the digital archives of the University of Tartu until the expiry of the term of copyright, my thesis **Interactive virtual assistant for Enhancing User Engagement with Open Government Data Portals**, supervised by Anastasija Nikiforova ;
2. grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright;

3. am aware of the fact that the author retains the rights specified in points 1 and 2;
4. confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Carl-Christjan Bogoslovski

15/05/2025