UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science

Kevin Kliimask

Automated Tagging of Datasets to Improve Data Findability on Open Government Data Portals

Bachelor’s Thesis (9 ECTS)

Supervisor: Anastasija Nikiforova, PhD

Tartu 2024

Automated Tagging of Datasets to Improve Data Findability on Open Government Data Portals

Abstract:

Efforts directed towards promoting Open Government Data (OGD) have gained significant traction across various governmental tiers since the mid-2000s. As more datasets are published on OGD portals, finding specific data becomes harder, leading to information overload. Complete and accurate documentation of datasets, including association of proper tags with datasets is key to improving data findability and accessibility. Analysis conducted on the Estonian Open Data Portal revealed that out of 1787 datasets published (as of April 23, 2024), 11% of datasets lacked any associated tags, while 26% had only one tag assigned to them, which underscores challenges in data findability and accessibility within the portal. The main goal of this thesis is to propose an automated solution to tagging datasets in order to improve data findability on OGD portals. This thesis presents a prototype application that employs Large Language Models (LLMs) such as GPT-3.5-turbo and GPT-4 to automate dataset tagging, providing tags in English and Estonian. The developed solution was evaluated by users and their feedback was collected to define an agenda for future prototype improvements.

Keywords:

Open Government Data, open data, data findability, automation, tag, large language model, LLM, GPT

CERCS:

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

Andmestike automaatne sildistamine andmete leitavuse parandamiseks riiklikes avaandmete portaalides

Lühikokkuvõte:

Alates 2000-ndate keskpaigast on erinevad valitsustasandid propageerinud riiklike avaandmete portaale. Kuna riiklikes avaandmete portaalides avaldatakse üha rohkem andmekogumeid, muutub konkreetsete andmete leidmine aina raskemaks. Andmekogumite leitavuse tagamise võtmeks on nende täielik ja täpne dokumenteerimine, sealhulgas andmestike seostamine asjakohaste siltidega. Eesti avaandmete teabeväravas on avalikustatud kokku 1787 andmestikku (23 aprill, 2024 seisuga) ning neid analüüsides selgus, et 11% andmestikest pole seotud ühegi sildiga. Lisaks selgus, et 26% andmestikest oli seotud ainult ühe sildiga. See viitab sellele, et Eesti avaandmete teabeväravas esineb probleeme andmekogumite leitavuse ja kättesaadavusega. Käesoleva töö peamine eesmärk on esitada automatiseeritud lahendus andmekogumite sildistamiseks, et parandada andmete leitavust riiklikes avaandmete portaalides. Selle töö käigus loodi rakenduse prototüüp, mis kasutab suuri keelemudeleid nagu GPT-3.5-turbo ja GPT-4 andmekogumite sildistamiseks inglise ja eesti keeles. Loodud prototüüpi hinnati kasutajate poolt ning nende tagasisidet kasutati rakenduse täiustamise planeerimiseks.

Võtmesõnad:
avalikud andmed, avaandmed, andmete leitavus, automatiseerimine, silt, märksõna, suur keelemudel

CERCS:
P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria); P175 Informaatika, süsteemiteooria; P176 Tehisintellekt

Table of Contents

Introduction

In recent times, governments around the world have developed Open Government Data (OGD) initiatives, merging the principles of open government and open data [1]. Governments around the globe publish governmental data on OGD portals that become available for a wide audience of users. Yet, users face difficulties in finding and discovering datasets related to their goals due to lack of sufficient descriptive metadata in open data catalogues [2]. As an increasing number of datasets become accessible, the challenge of information overload arises due to the difficulty in locating specific information [3]. A recommended approach to enhance the discoverability and shareability of published datasets involves employing expressive descriptors, such as tags, effectively [3]. Descriptors constitute a form of metadata, providing details about the content of a resource to assist in its discovery or comprehension [4]. Incomplete or inaccurate metadata inhibits consumers from discovering relevant data for their requirements, leading to the necessity of spending significant time manually searching through portals and the data itself to identify relevant datasets [3, 5].

Through an analysis of the Estonian Open Data Portal datasets, significant shortcomings were discovered in data tagging practices. Among the 1787 datasets published as of April 23, 2024, a trend emerged: 190 datasets (11%) lacked any associated tags, while an additional 457 (26%) possessed only one tag. These findings underscore challenges in data findability and accessibility within the portal.

The goal of this thesis is to address the challenges associated with dataset findability and metadata quality by developing a prototype that automates the tagging process by employing a Large Language Model (LLM). The development of such a tool holds significant promise for both data publishers and consumers. Automating the tagging process for data publishers mitigates the risk of datasets lacking tags, a common occurrence on portals where their inclusion isn't mandatory. Moreover, this automation minimises the association of datasets with incomplete or inaccurate tags. Consequently, it enhances the findability and accessibility of datasets, facilitating streamlined access for users. By streamlining the metadata enrichment process, publishers can allocate resources more efficiently and accelerate the dissemination of datasets associated with high-quality tags.

For the study, a LLM powered prototype application was developed to automate the tagging process for datasets formatted in CSV, one of the most popular open data formats [6, 7]. The solution was developed as a web service, designed to optimise interoperability and integration with different platforms and systems. Additionally, a front-end application was also developed to test the prototype's efficiency and usability.

To evaluate the prototype, a survey was conducted, which received 22 responses. Participants were asked to evaluate properties of the application such as relevancy of generated tags, user-friendliness and usefulness. Feedback from the respondents was used to outline areas of the prototype for future improvement.

The thesis is organised in the following way: chapter 2 defines the core concepts related to the study and dives into data findability issues in data portals, as well as in the Estonian Open Data Portal. Chapter 3 introduces the implementation approach and technological framework of the prototype developed within this study. Chapter 4 dives into the implementation details of the developed prototype. Chapter 5 presents the methodology of the prototype evaluation and the results collected from participants. The paper concludes with chapter 6, which discusses the feedback received and outlines future improvements of the prototype.

ChatGPT 3.5 was used to enhance the readability and aesthetic appeal of the text in this thesis. For example, ChatGPT 3.5 was used to improve wording and sentence structure, and to fix grammar mistakes.

1. Terms and notions

API	Application Programming Interface
FAIR	Findability, Accessibility, Interoperability, Reusability
GUI	Graphical User Interface
LLM	Large Language Model
OGD	Open Government Data

2. Background

This chapter defines the core concepts used in the thesis, including open government data and a brief overview of the FAIR principles with further determination of the problem this thesis attempts to resolve. In addition, a short overview on the state of the art of Estonian Open Data Portal is provided and its current search capabilities are explored.

2.1 Open Government Data

Initiatives aimed at fostering Open Government Data (OGD), including the establishment of OGD portals, have seen widespread adoption since the mid-2000s across governmental levels [8]. The Organisation for Economic Co-operation and Development (OECD) defines OGD as both a philosophy and a set of policies aimed at fostering transparency, accountability, and value generation by making government data accessible to the public [9]. This data must be compliant with principles set by the Open Data Charter, according to which data to be recognized as OGD must be: open by default; timely and comprehensive; accessible and usable; comparable and interoperable; suitable for improved governance and citizen engagement, as well as for inclusive development and innovation [10]. According to the OECD, public entities generate substantial amounts of data, and by sharing this data, they enhance transparency and accountability to citizens [9]. Moreover, the OECD notes that encouraging the use, reuse, and free distribution of datasets by governments promotes the creation of businesses and innovative, citizen-centric services [9].

This movement has been joined by the vast majority of countries globally, including Estonia. In Estonia, Open Government Data is centralised within the Estonian Open Data Portal [11], which will be briefly examined in chapter 2.4.

2.2 FAIR Principles

Another concept closely related to OGD that aims to maximise the value and usability of data, albeit within another context, is FAIR. FAIR principles were introduced in 2016 by Wilkinson et al. [12]. The FAIR principles are guiding principles that enable both machines and humans to find, access, interoperate and re-use data and metadata [13, 14]. FAIR stands for Findability, Accessibility, Interoperability and Reusability, where:

- **Findability** is the principle according to which both humans and computers should encounter minimal difficulty in locating metadata and data resources. Machine-readable metadata plays a crucial role in facilitating automated discovery of datasets and services, thus constituting a fundamental aspect of the FAIRification process [14];
- **Accessibility** requires that after locating the desired data, the user must ascertain the methods for accessing them, which may involve considerations such as authentication and authorization processes [14];
- **Interoperability** sets prerequisites for data to be integrated with other datasets, making them capable of interoperating with various applications or workflows for purposes such as analysis, storage, and processing [14];
- **Reusability**, being the primary goal of FAIR, dictates the need to enhance the efficiency of data reuse. This entails ensuring that metadata and data are well-described so they can be reused in different settings [14].

As such, OGD initiatives and FAIR principles share common goals of maximising the value and usability of data by promoting principles of openness, accessibility, interoperability, and reusability. However, although both are related concepts, they serve slightly different purposes, where OGD initiatives focus specifically on making government data open and accessible to the public, while FAIR principles provide a broader framework for ensuring that data, regardless of its source, is findable, accessible, interoperable and reusable (FAIR). As such, data can be compliant with the open (government) data principles, but not necessarily compliant with FAIR principles and vice versa, FAIR data is not necessarily open (government) data principles-compliant, whereas the greatest result is achieved when both sets of principles are fulfilled [15].

2.3 Data Findability Issues

Data published on data portals is subject for search through several approaches, which is typically based on the metadata as indicated by the publisher [2]. Open data portals usually offer diverse browsing interfaces to aid users in locating relevant datasets, providing facets such as publisher, file format, spatial/geographical coverage, time period and other properties, whilst also keyword and tag based search being prevalent [16]. Tags associated with datasets are usually defined by data publishers when publishing datasets, where these tags can be thought of as "expressive descriptors" [3]. These expressive descriptors, or tags, play a crucial role in facilitating efficient navigation through data portals [2, 3]. By associating datasets with relevant tags, users can swiftly locate datasets relating to specific topics [2, 3]. Entering tags manually is slow and prone to human errors, where human entered tags are not always accurate or relevant to the actual dataset [3, 17]. Additionally, if the portal's design doesn't enforce mandatory tagging, publishers may overlook tagging entirely due to its time-consuming nature. Inadequate metadata, including descriptions or tags, renders both manual and automated searches ineffective in locating the dataset, thus making the dataset unuseful [18].

As advancements in artificial intelligence technologies continue, these advancements can be harnessed to enhance the findability of data through automated tagging of datasets. Moreover, automation elements are inherent to the FAIR vision [19].

Infringement of findability is also apparent on the Estonian Open Data Portal regardless of its general competitiveness on the global scene (as per Open Data Maturity (ODM) report 2023)[1], which will be elaborated in chapter 2.4.3.

2.4 Overview on the Estonian Open Data Portal

The first Estonian open data portal was launched in 2015 [20]. It was built on the Comprehensive Knowledge Archive Network (CKAN) platform - one of the most widely used open-source data management systems used by Open Government Data (OGD) portals [20]. As the initial implementation encountered constraints, with several functionalities of the CKAN platform remaining unused and a limited number of datasets being published on it over time (which is also consistent with lower rank of Estonian OGD initiative in ODM reports), in 2021, the Estonian Ministry of Economics and Communication launched a new national open data portal - avaandmed.eesti.ee [20], which hosts 1787 datasets as of April 23, 2024.

[1] https://data.europa.eu/en/publications/open-data-maturity/2023

Similar to worldwide practices for OGD portals, Estonian national open data portal provides its users with several capabilities for dataset search, namely:

1. text search that allows dataset search by their title;
2. faceted search that allows datasets search by facets such as publisher, file format, keyword a.k.a tag, spatial/geographical coverage, year and category.

While some facets used for dataset search can be automatically retrieved from the data associated with the publisher or the dataset, e.g., dataset format, some facets, such as spatial coverage and tags, are expected to be provided by the data publisher. As such, the accuracy and completeness of tags, which are an integral part of metadata, depend directly on the data provided by publishers.

2.4.2 Challenges with Tags in Open Government Data Portals

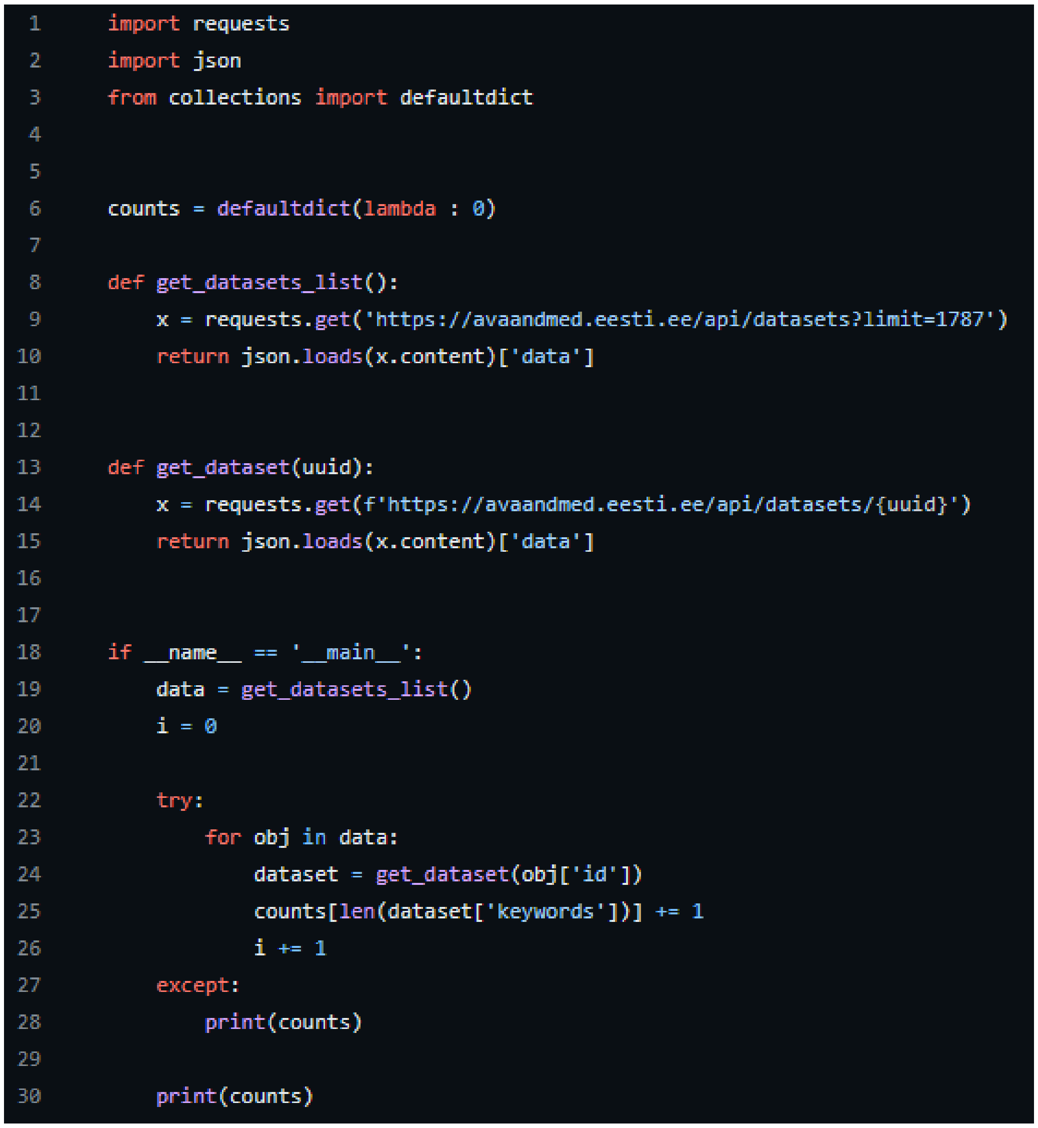
While tags may seem to be a trivial facet, the current practice shows that both their presence and relevance to the actual dataset tend to be a challenge for OGD portals, including the Estonian Open Data Portal. As found out in conversation with Estonian Open Data Portal representatives, tags are manually added to published datasets by their publishers. For example, the dataset "Clinical trials in the recruitment phase in Estonia"[2] has no "cancer" tag, although the dataset contains multiple clinical trials related to cancer research. To this end, to assess the relevance of the topic of this thesis, an analysis of datasets available on the Estonian Open Data Portal was conducted with the aim to examine the relevance of the issue in question, i.e., lack of or insufficient quality of tags associated with published datasets on Estonian Open Data Portal.

To analyse the number of tags associated with each dataset on the portal as defined by data publishers, a Python scraping script was developed (see Figure 1), the code of which is available in Appendix I. The Estonian Open Data Portal API manual [21] was used to define API endpoints for retrieving metadata, such as dataset tags. The script operates as follows:

- list of all datasets on the portal is retrieved using the *get_datasets_list()* function, which iterates over each dataset. *get_datasets_list()* fetches datasets from the API endpoint *https://avaandmed.eesti.ee/api/datasets*;

[2] https://avaandmed.eesti.ee/datasets/varbamisfaasis-kliinilised-uuringud-eestis

- for each dataset, detailed information is retrieved using the *get_dataset(uuid)* function, where the parameter *uuid* is the dataset's unique identifier. *get_dataset(uuid)* fetches the detailed information from the API endpoint *https://avaandmed.eesti.ee/api/datasets/{uuid}*;
- the length of the dataset's *keywords* field is determined. The count of tags for each dataset is then incremented in the counts dictionary;
- once all the retrieved datasets are processed, the dictionary containing counts for each number of tags is printed out.



```
import requests
import json
from collections import defaultdict


counts = defaultdict(lambda : 0)

def get_datasets_list():
    x = requests.get('https://avaandmed.eesti.ee/api/datasets?limit=1787')
    return json.loads(x.content)['data']


def get_dataset(uuid):
    x = requests.get(f'https://avaandmed.eesti.ee/api/datasets/{uuid}')
    return json.loads(x.content)['data']


if __name__ == '__main__':
    data = get_datasets_list()
    i = 0

    try:
        for obj in data:
            dataset = get_dataset(obj['id'])
            counts[len(dataset['keywords'])] += 1
            i += 1
    except:
        print(counts)

    print(counts)
```

Figure 1. Estonian open data portal scraping script.

The analysis performed in accordance with this procedure confirmed the relevance of the thesis objective, uncovering significant negative trends in datasets tagging practice. Specifically, out of the 1787 datasets published (as of April 23, 2024), 190 datasets (11%) lacked any associated tags, while 457 (26%) had only one tag assigned to them. This infringes the principles of FAIR, i.e., if a dataset is lacking relevant metadata such as tags, it will be more difficult for interested parties to find it (infringes findability) and to integrate it with other datasets (infringes interoperability) [14]. As an end result, lack of keywords/tags will make the dataset less likely to be reused, which infringes reusability [14]. This indicates potential areas for improvement in dataset tagging within the portal through augmentation of this process, which is a central objective of this thesis.

3. Implementation Framework

This chapter presents the implementation approach and technological framework of the prototype application developed in regards to the thesis. First, the process of automating dataset tagging is described. Subsequently, the technological choices made to reach the goal, including the large language model, web service framework, graphical user interface framework, translation service and cloud provider for hosting are explained.

3.1 Automation of Dataset Tagging

The objective of the thesis is achieved by automating dataset tagging, which, in turn, is achieved by employing a Large Language Model (LLM). A large language model is appropriate for this purpose, as it was found to be useful for predicting tags from partial content of a dataset [22]. As such, the following steps outline the process of automatically tagging a dataset:

1. the LLM gets a system prompt describing to it which data it will receive, which task it has to do and how its response should be formatted. A system prompt is a message that can be used to specify the persona used by the model in its replies [23]. Instructions to the LLM are provided in English;
2. then, the LLM is provided with the first rows of a dataset, including the dataset's header row. The number of rows provided to the LLM is 10. Experimentation has shown that this number of rows is one of the lowest that still allows the LLM to generate relevant tags. Moreover, every additional row provided for analysis would increase the computational resources required, thus making the process more expensive. Furthermore, this number of rows also fits inside the input token limit of the LLM, which determines the maximum length of the input string that the LLM can accept;
3. after processing the input, the LLM outputs a list of relevant tags. The tags are in English. The number of tags to output can be chosen by the user, of which the LLM is informed through the initial system prompt (step 1);
4. finally, in addition to the English tags generated by the LLM at step 3, translation of the generated tags in Estonian is also returned to the user. The translations do not originate from the LLM, instead, the English tags generated by the LLM are translated separately by using a machine translation service's API.

3.2 Interfacing with the LLM

Communication with the LLM is achieved through a RESTful web service, which handles interfacing with the LLM's API. A RESTful web service is a web application that adheres to REST[3] standards [24]. Web services enable various organisations or applications from diverse origins to interact without the necessity of exchanging sensitive data or IT infrastructure [25]. Developing the project as a web service has the benefit of not limiting the project to the Estonian Open Data Portal or OGD portals in general, thereby making it environment-agnostic, which will make it convenient to integrate the tagging service with other products.

Additionally, a basic graphical user interface (GUI) is developed to interface with the web service. This is done to allow for a more streamlined and user-friendly usability testing (covered in chapter 5). In order to facilitate usability testing over distance, the application should be deployed to the cloud. By implementing this approach, users will be spared the need to set up the application locally, thus alleviating the associated inconvenience. Furthermore, it ensures that sensitive API keys remain protected and do not need to be shared with users during the testing phase (covered in chapter 4.4).

3.3 Technology Choices

This chapter presents the technological choices made to develop an automated tagging service, with the reference to both LLM, web service framework, GUI framework, translation service and cloud provider.

3.3.1 Large Language Model

Since the prototype under development is LLM-powered, the first technological choice concerned which LLM to use. The factors that determined the choice of LLM were performance, cost and ease of implementation. Several benchmarks have been developed to evaluate the performance of a LLM, such as HELM (Holistic Evaluation of Language Models), which is a research benchmark developed by the Stanford CRFM (Center for Research on Foundation Models) to assess performance across a variety of prediction and generation scenarios, Open LLM leaderboard by HuggingFace, which is a leaderboard for open source LLM evaluation

[3] https://www.integrate.io/blog/rest-api-standards/

across 4 benchmarks - MMLU, TruthfulQA, HellaSwag and AI2 reasoning, and Chatbot Arena by LMSys, which is a benchmark utilising an Elo-derived ranking system, aggregated over pairwise battles [26].

However, there is no widely used benchmark for evaluating performance of LLMs as data annotators [26]. For this thesis a technical report by Refuel [26] was used to find a LLM with the best tradeoff between label quality and cost. The report evaluated the performance of 6 LLMs, namely Text-davinci-003, GPT-3.5-turbo, GPT-4, Claude-v1, FLAN-T5-XXL, PaLM-2 for labelling datasets. The report identified that the top 3 LLMs with the best tradeoff between label quality and cost were FLAN-T5-XXL, PaLM-2 and GPT-3.5-turbo, which were further considered for the purpose of this study. An additional investigation of the three LLMs revealed that FLAN-T5-XXL requires self-hosting, which increases the complexity of developing the solution. PaLM-2 and GPT-3.5-turbo offer a paid API, which is easier to implement than a self-hosted LLM. As the performance of the 2 models is similar, where GPT-3.5-turbo generates better quality labels compared to PaLM-2 in 5 out of 10 datasets, while the cost per label of PaLM-2 is ~70% higher [26], the choice to use GPT-3.5-turbo was made.

In addition, during development, the decision to include GPT-4 as an additional option was made. This choice was made to better evaluate GPT-3.5-turbo, and, should GPT-4 noticeably outperform GPT-3.5-turbo in testing, GPT-4 would be kept as a primary selection, regardless of the fact that it has a higher cost per label.

3.3.2 Web Service

A RESTful web framework was used to develop a HTTP-based API for accessing the web service. The choice of framework was FastAPI - a "web framework for building APIs with Python 3.8+ based on standard Python type hints" [27], as according to independent benchmarks by TechEmpower, FastAPI is considered as one of the fastest Python frameworks available, only below Starlette and Uvicorn [28]. In addition to being one of the most performant Python frameworks, the author's previous experience with it contributed to this decision. FastAPI is built upon Starlette, which itself is built upon Uvicorn, which explains the differences in performance as this hierarchical architecture inherently introduces additional layers of abstraction, resulting in increased overhead [28]. But as an added benefit, FastAPI provides more features on top of Starlette, such as data validation and serialisation that are essential to building APIs [28]. By using a higher-level framework such as FastAPI, development time is saved and similar performance to a lower-level framework, such as Starlette, can be achieved as features missing in Starlette would have to be developed manually [28]. In addition, OpenAI (the company that offers the GPT-3.5-turbo and GPT-4 models) provides official Python bindings for using their models [29], which makes using a Python-based framework convenient.

3.3.3 Graphical User Interface

When making a decision about a graphical user interface, a choice in favour of one of two options should be made, namely a desktop application or a web application. A front-end web application as the graphical user interface was chosen to facilitate a more seamless user-testing experience. The decision was influenced by several factors. Notably, web applications offer the advantage of immediate accessibility without the need for installation, ensuring users can swiftly engage with the application across different devices and operating systems [30]. While it's acknowledged that web applications rely on an internet connection, which could be perceived as a limitation [30], usage of the LLM requires an internet connection regardless. Therefore, this potential drawback becomes irrelevant in the context of this study.

Node.js and React stand as two of the most used front-end web frameworks globally [31]. Node.js is an open-source JavaScript runtime environment that facilitates the development of servers and web applications [32]. Conversely, React is described as a “library for web and native user interfaces” [33]. Given that Node.js is predominantly tailored towards API creation, while React is renowned for its prowess in creating user interfaces [34], the decision to use React was made due to its better alignment with the project’s requirements. Additionally, the author’s prior experience with React further bolstered its selection.

3.3.4 Machine Translation Service

For this project, the criteria for choosing a machine translation service was that it must be accurate and have an accessible API. According to research conducted by Intento [35], DeepL[4] emerged as the top-performing neural machine translation service. DeepL offers a free, although

[4] https://www.deepl.com/translator

limited, access plan to access their API. Additionally, the existence of an official Python library maintained by DeepL[5] facilitates its convenient integration into the application. Considering these factors, the decision to use DeepL as the project's machine translation service was made.

Although Google Translate was initially considered during the project's early stages, a comparative analysis revealed that DeepL consistently delivered more accurate translations. This performance disparity ultimately solidified DeepL as the preferred choice for the project.

3.3.5 Cloud Provider

When selecting a cloud provider, the primary criteria were cost-effectiveness and ease of application deployment. For this project Vercel was chosen. Vercel is a cloud-based platform specifically tailored for hosting static sites and serverless functions, offering developers a streamlined process in developing and launching web projects [36]. Vercel offers the ability to run back-end code as serverless functions [37]. A serverless function embodies business logic that operates without retaining data (stateless) and has a temporary lifespan, being created and then terminated [38]. These functions persist for short durations, mere seconds, and are intended to be triggered by a specific condition, such as an user making a request. Given that the web service does not need to retain data and only needs to run upon a request, the utilisation of serverless functions was deemed aligned with the project. In addition, Vercel offers a free tier and its straightforward deployment process further solidified its suitability for the project.

[5] https://github.com/DeepLcom/deepl-python

4. Implementation

In this chapter, implementation of the dataset tagger prototype is presented. First, implementation of the prototype back-end is presented, with subsequent presentation of the front-end. Finally, the process of hosting the application is presented.

4.1 Back-end

The back-end of the developed prototype consists of 4 main modules: (1) API endpoint, (2) OpenAI service, (3) translator service and config module (4), each playing its own crucial role:

- **API endpoint** accepts requests and validates received data from the user. The data consists of the first 10 rows of a to-be tagged dataset, including its header row;
- **OpenAI service** handles interfacing with OpenAI API. It creates a system prompt, appends data received from the API endpoint to an user prompt and sends both messages to the LLM. It, in turn, receives a response from the LLM with tags (generated by the LLM in English);
- **Translator service** handles interfacing with DeepL API. It takes tags received from OpenAI service and translates them to another language, which within the context of the thesis is Estonian;
- **Config module** handles loading environment variables into the application. The necessary environment variables for the back-end application to function are front-end url, OpenAI API key and DeepL API key.

In addition, the back-end project contains a requirements file for required Python packages and a Vercel configuration file for the prototype application deployment purposes. The required Python packages for the project are *fastapi*, *pydantic*, *pydantic-settings*, *python-multipart*, *uvicorn*, *openai*, *deepl* and all dependencies of the preceding packages. All parts of the developed prototype are available in a Github repository[6] (also in Appendix I).

In subsequent subchapters each module is presented in more detail.

[6] https://github.com/kevinkliimask/gpt-tagger

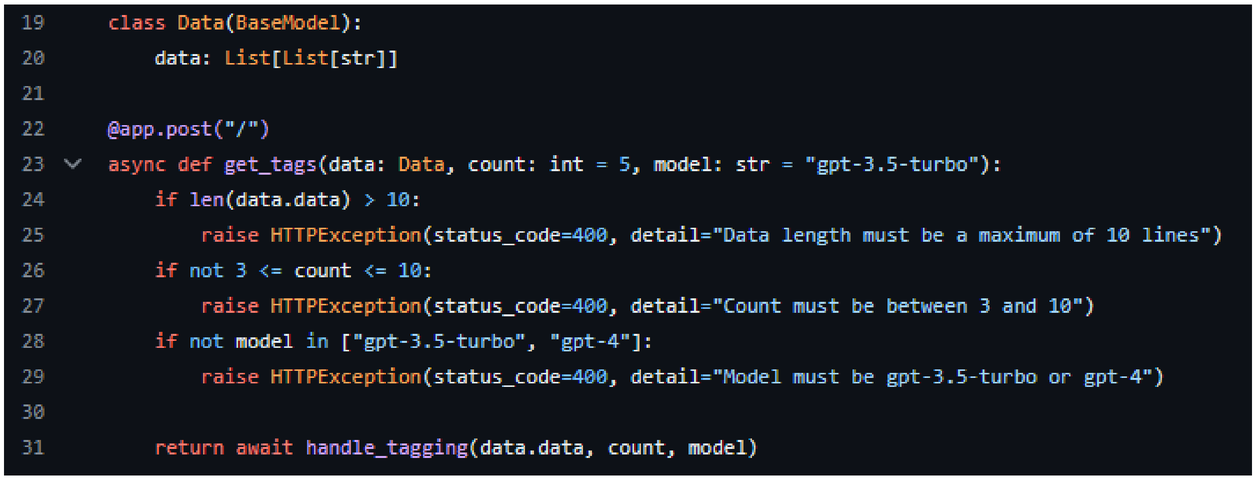
4.1.1 API Endpoint

The API endpoint accepts data sent via HTTP POST method. Furthermore, the endpoint is mapped to the "/" route, also known as the root route. As there are no other endpoints in the application, it is sufficient to accept requests only on the root route.

Received data is validated to prevent unexpected behaviours in the application. The API endpoint accepts a body consisting of a matrix, where the matrix represents data from a dataset. In addition, the endpoint accepts *count* and *model* as query parameters from the user. These determine how many tags the LLM should generate and which LLM model should be used, respectively. The default values for these parameters are 5 tags and gpt-3.5-turbo model, correspondingly. The validation logic sets the following rules for the received data:

- length of data in the request body, which represents the number of rows of a dataset, must be a maximum of 10 lines;
- *count* should be in the range of 3 to 10;
- *model* should be either gpt-3.5-turbo or gpt-4.

If any of the validations fail, a HTTP exception is returned as a response and is shown to the user, specifying the nature of the error (see Figure 2).

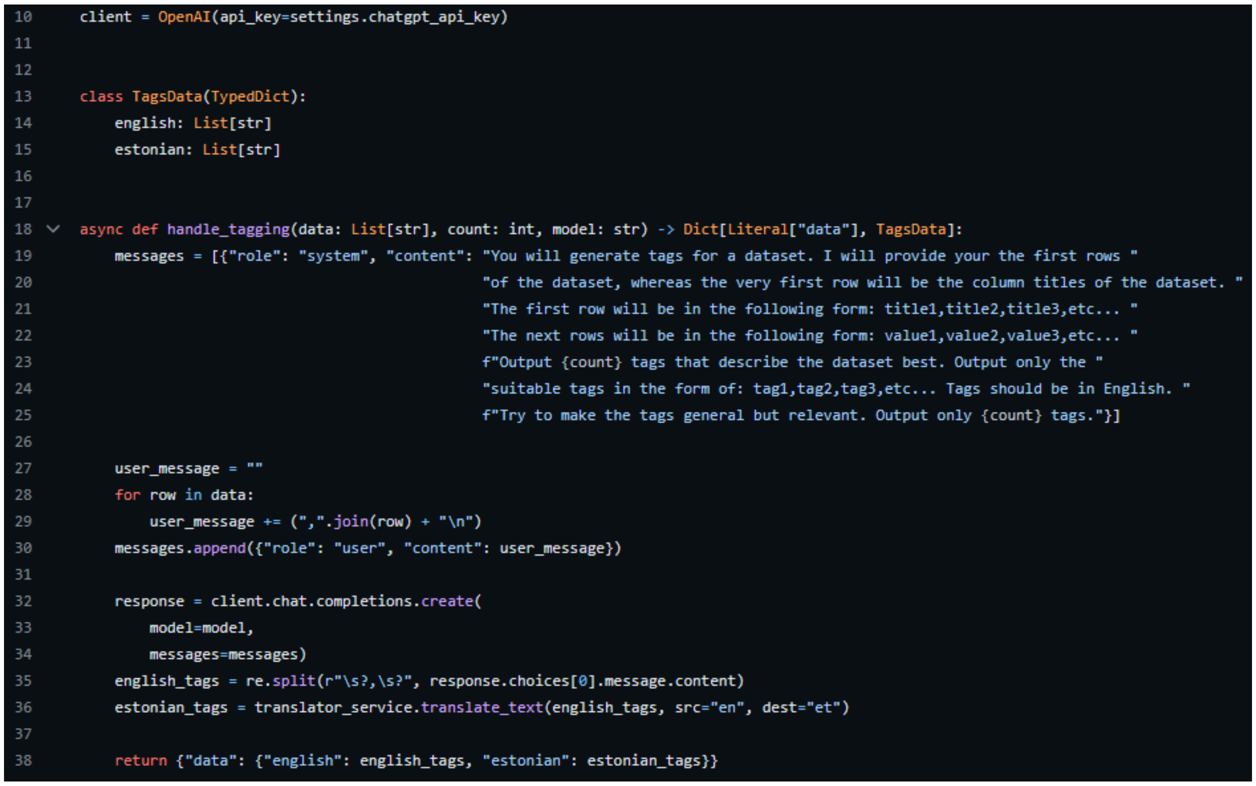


```python
class Data(BaseModel):
    data: List[List[str]]

@app.post("/")
async def get_tags(data: Data, count: int = 5, model: str = "gpt-3.5-turbo"):
    if len(data.data) > 10:
        raise HTTPException(status_code=400, detail="Data length must be a maximum of 10 lines")
    if not 3 <= count <= 10:
        raise HTTPException(status_code=400, detail="Count must be between 3 and 10")
    if not model in ["gpt-3.5-turbo", "gpt-4"]:
        raise HTTPException(status_code=400, detail="Model must be gpt-3.5-turbo or gpt-4")

    return await handle_tagging(data.data, count, model)
```

Figure 2. API endpoint and data validation logic.

4.1.2 OpenAI Service

The OpenAI service defines a function *handle_tagging* that uses OpenAI API to generate tags for a dataset. Communication with OpenAI API is handled by OpenAI Python library. The function takes a list of records from a dataset, the number of tags to generate, and the model to use as input parameters, all provided by the user. The function builds messages to send to the OpenAI API, formats the data into a user message, sends the messages to the API, retrieves the generated tags, splits them into English tags, and then translates them into Estonian using translator service (whose operation is described in the next subchapter). Finally, it returns a dictionary containing both English and Estonian tags (see Figure 3).

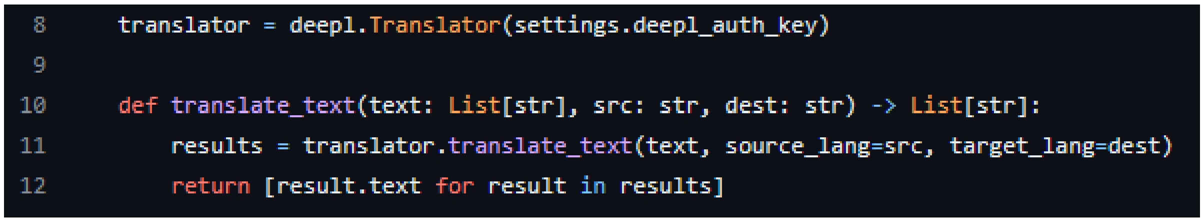


```python
client = OpenAI(api_key=settings.chatgpt_api_key)


class TagsData(TypedDict):
    english: List[str]
    estonian: List[str]


async def handle_tagging(data: List[str], count: int, model: str) -> Dict[Literal["data"], TagsData]:
    messages = [{"role": "system", "content": "You will generate tags for a dataset. I will provide your the first rows "
                                              "of the dataset, whereas the very first row will be the column titles of the dataset. "
                                              "The first row will be in the following form: title1,title2,title3,etc... "
                                              "The next rows will be in the following form: value1,value2,value3,etc... "
                                              f"Output {count} tags that describe the dataset best. Output only the "
                                              "suitable tags in the form of: tag1,tag2,tag3,etc... Tags should be in English. "
                                              f"Try to make the tags general but relevant. Output only {count} tags."}]

    user_message = ""
    for row in data:
        user_message += (",".join(row) + "\n")
    messages.append({"role": "user", "content": user_message})

    response = client.chat.completions.create(
        model=model,
        messages=messages)
    english_tags = re.split(r"\s?,\s?", response.choices[0].message.content)
    estonian_tags = translator_service.translate_text(english_tags, src="en", dest="et")

    return {"data": {"english": english_tags, "estonian": estonian_tags}}
```

Figure 3. OpenAI service logic.

4.1.3 Translator Service

To ensure tags are generated in a language other than English, such as Estonian, as is the case for this study, translator service is used. The service defines a function *translate_text* that uses DeepL API to translate tags originally generated by the LLM. Interfacing with DeepL API is handled by the DeepL Python package. The function accepts a list of tags, source language and destination language as input parameters, which in this case are English and Estonian, respectively. The function translates every string in the input list and returns the translated strings as a list (see Figure 4).

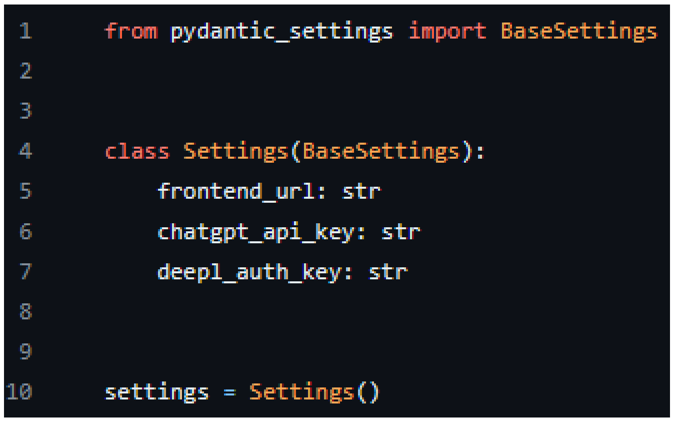


```python
translator = deepl.Translator(settings.deepl_auth_key)

def translate_text(text: List[str], src: str, dest: str) -> List[str]:
    results = translator.translate_text(text, source_lang=src, target_lang=dest)
    return [result.text for result in results]
```

Figure 4. Translator service logic.

4.1.4 Config module

The config module defines a *Settings* class that inherits from *BaseSettings* provided by Pydantic[7] - a library for data validation and settings management. It specifies the environment variables required for the application, namely *frontend_url*, *chatgpt_api_key*, and *deepl_auth_key*. Then, it creates an instance of the *Settings* class to load the values of these environment variables (see Figure 5). This approach ensures that the application's settings are correctly loaded and validated from the environment. Additionally, this setup enables anybody to run the application and use their own environment variables seamlessly.



```python
from pydantic_settings import BaseSettings


class Settings(BaseSettings):
    frontend_url: str
    chatgpt_api_key: str
    deepl_auth_key: str


settings = Settings()
```

Figure 5. Config module.

[7] https://docs.pydantic.dev/latest/

4.2 Front-end

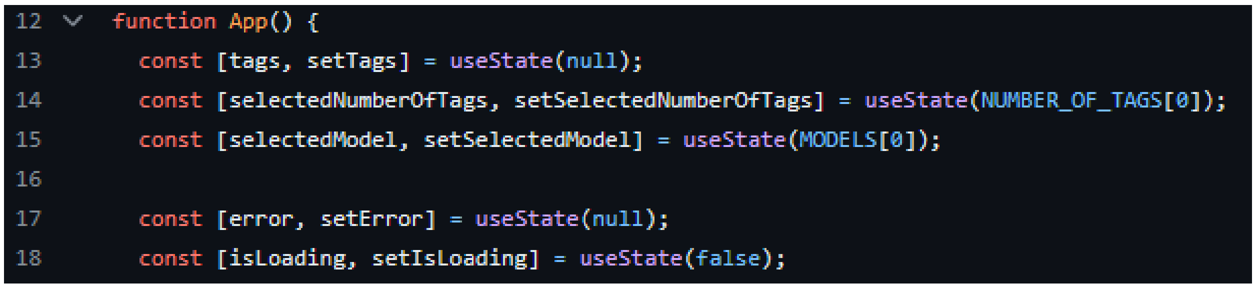
The front-end architecture is centred around a single React component named App, functioning as the primary entry point for the application.

4.2.1 Dependencies

The application's required packages are defined in a *package.json* file. For a React app to operate, the main dependencies are *react*, *react-dom* and *react-scripts*. In addition, the developed React application also makes use of the *react-drag-drop-files* package to handle file uploads through drag-and-drop functionality. When starting the app, the environment variable *REACT_APP_BACKEND_URL* must be defined to specify the backend server's URL.

4.2.2 State Management

The *useState* hook from React is used to manage component state. The App component utilises the *useState* hook to manage states of *tags*, *selectedNumberOfTags*, *selectedModel*, *error*, and *isLoading* (see Figure 6). These states are essential for tracking the uploaded file, selected parameters, error messages, and loading status.

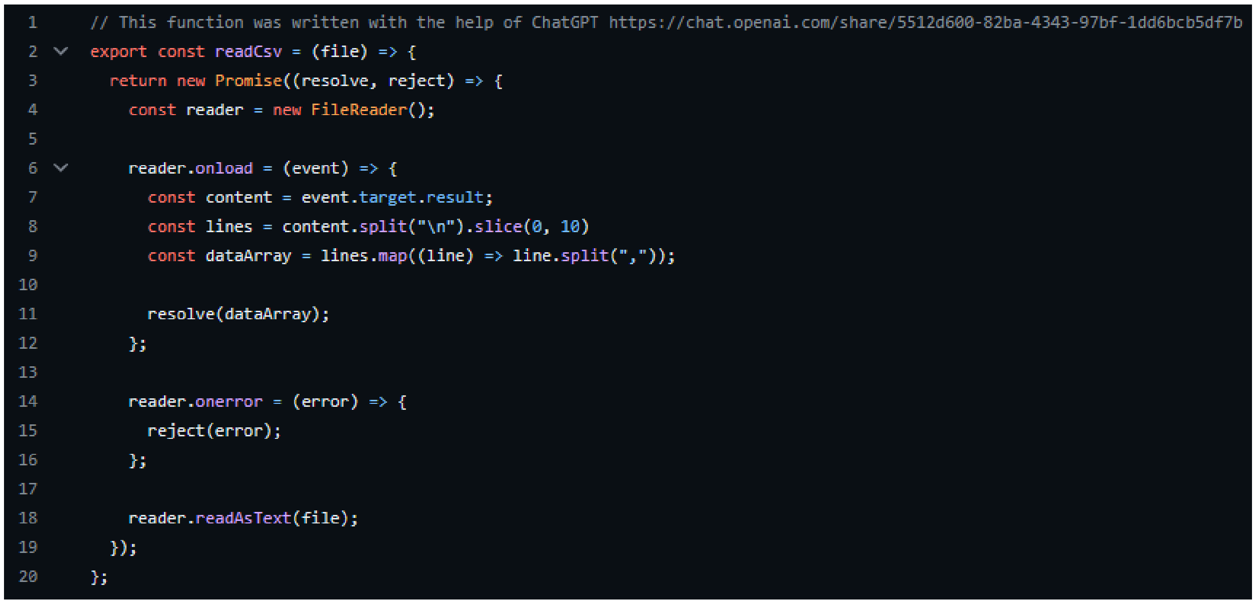


```
function App() {
  const [tags, setTags] = useState(null);
  const [selectedNumberOfTags, setSelectedNumberOfTags] = useState(NUMBER_OF_TAGS[0]);
  const [selectedModel, setSelectedModel] = useState(MODELS[0]);

  const [error, setError] = useState(null);
  const [isLoading, setIsLoading] = useState(false);
```

Figure 6. App state variables.

4.2.3 File Upload and Tag Generation

The *handleChange* function is invoked upon uploading a file. It utilises the *readCsv* utility function to extract data from the uploaded file. The *readCsv* utility function parses the CSV file uploaded by the user, preparing the data for transmission to the backend. This function accepts a single parameter *file*, representing the uploaded CSV file, and returns a *Promise* resolving to an array containing the first 10 rows of the parsed CSV file data (see Figure 7). Parsing CSV data in the front-end offers the advantage of bypassing the need to transfer large files to the back-end for processing. Consequently, this approach eliminates the need for a size limit on file uploads, with the maximum size being solely dictated by the browser (for instance, in Chrome, the limit is 4GB).



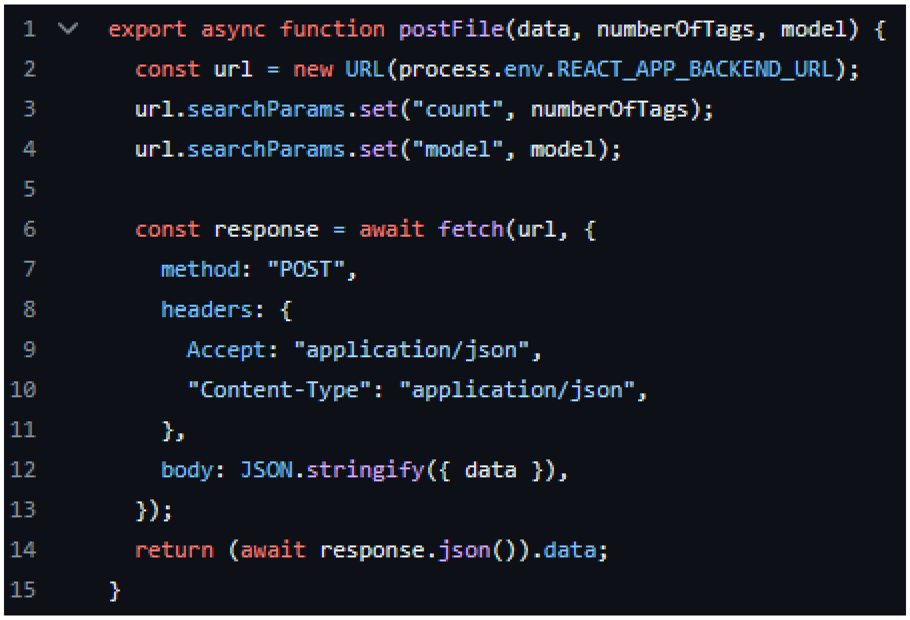
```javascript
// This function was written with the help of ChatGPT https://chat.openai.com/share/5512d600-82ba-4343-97bf-1dd6bcb5df7b
export const readCsv = (file) => {
  return new Promise((resolve, reject) => {
    const reader = new FileReader();

    reader.onload = (event) => {
      const content = event.target.result;
      const lines = content.split("\n").slice(0, 10)
      const dataArray = lines.map((line) => line.split(","));

      resolve(dataArray);
    };

    reader.onerror = (error) => {
      reject(error);
    };

    reader.readAsText(file);
  });
};
```

Figure 7. *readCsv* utility function.

Upon successful CSV file reading, the *postFile* function sends the parsed data along with selected parameters (*selectedNumberOfTags* and *selectedModel*) to the server for tag generation. The *postFile* function handles the transmission of data to the backend server for tag generation. It accepts the following parameters:

- *data*, which represents the first 10 rows of the uploaded CSV;
- *numberOfTags*, which is the number of tags the user has chosen to be generated by the LLM;
- *model*, which is the LLM that the user has chosen to be used for tag generation.

The *postFile* function constructs the backend URL using the provided environment variable *REACT_APP_BACKEND_URL*, appending query parameters for *count* (number of tags) and *model*. It then performs a POST request to the constructed URL using the JavaScript *fetch* API, which returns a *Promise*. Finally, the *Promise* is resolved and generated tags are extracted from the JSON response (see Figure 8). The generated tags are then stored in the component state (*tags*), and any errors during the process are captured and displayed.



```javascript
export async function postFile(data, numberOfTags, model) {
  const url = new URL(process.env.REACT_APP_BACKEND_URL);
  url.searchParams.set("count", numberOfTags);
  url.searchParams.set("model", model);

  const response = await fetch(url, {
    method: "POST",
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json",
    },
    body: JSON.stringify({ data }),
  });
  return (await response.json()).data;
}
```

Figure 8. *postFile* function.

4.2.4 User Interface

The user interface consists of a card layout containing the application title, parameter selection dropdowns, file uploader, and sections for displaying generated tags, loading status, and error messages. Dropdown menus are provided for selecting the number of tags and the model to be used for tag generation. The *react-drag-drop-files* library provides a *FileUploader* component that enables users to upload files, restricting them to only CSV file types (see Figure 9).

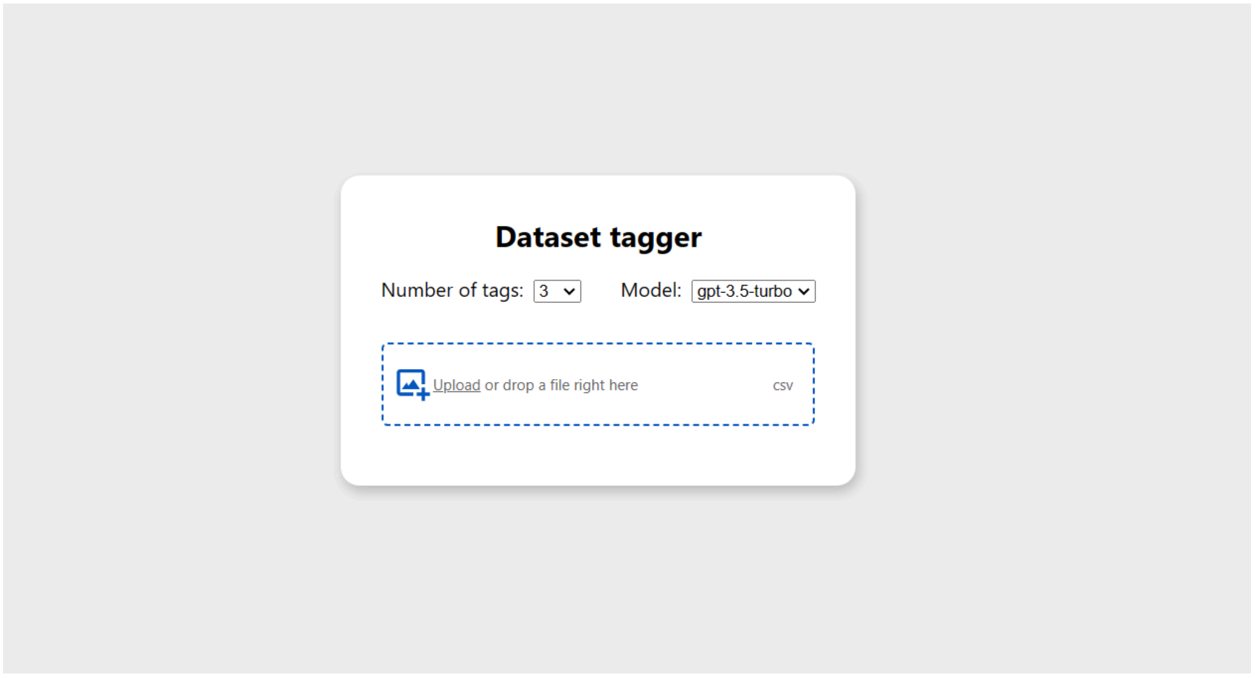
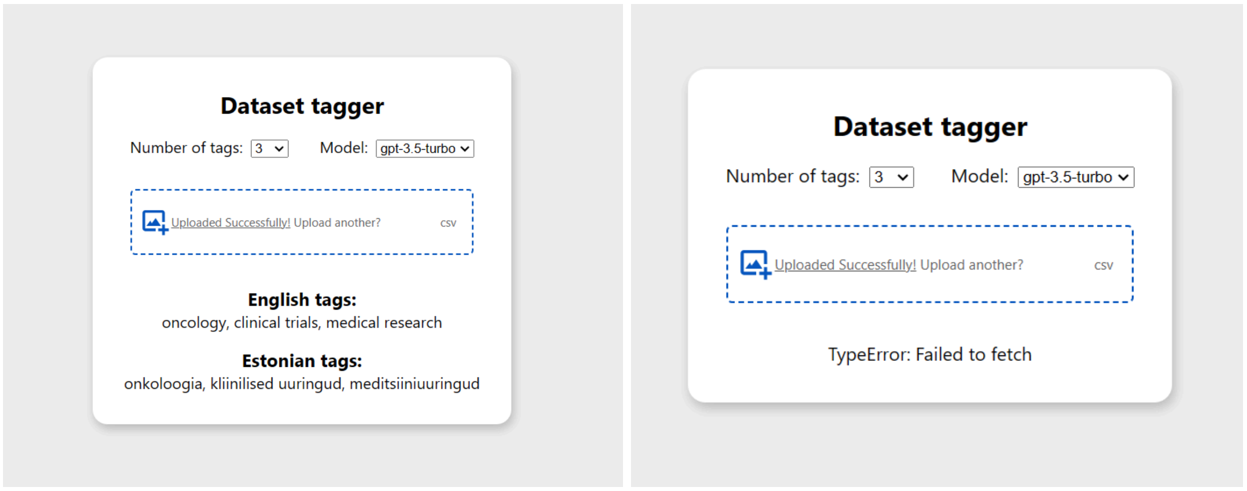


Figure 9. Front-end application interface.

The component dynamically renders elements based on the current state. For example, it displays generated tags if available, shows loading indicators during file processing, and renders error messages if any errors occur (see Figure 10 & 11).



English tags:
oncology, clinical trials, medical research

Estonian tags:
onkoloogia, kliinilised uuringud, meditsiiniuuringud

TypeError: Failed to fetch

Figure 10 & 11. Successful file upload and unsuccessful file upload.

The application's styling is maintained through CSS, with style rules defined in the *App.css* file. These styles are designed to offer users a clean and intuitive layout, enhancing the overall interaction experience.

4.3 Hosting

Vercel facilitates automatic deployments triggered by changes to the respective front-end or back-end folders within the *main* branch of the source code's repository. Both the front-end and back-end source code are hosted in a single repository. A repository that contains multiple projects, such as the back-end and front-end, is called a monorepo [39]. The deployment of the application on Vercel is separated into 2 different Vercel projects: gpt-tagger and gpt-tagger-frontend. This is Vercel's recommended approach to deploying applications that use a monorepo [40].

In the case of the back-end project (gpt-tagger), a custom *vercel.json* configuration file is used to define information for Vercel to set up a Python runtime when deploying. For the front-end project (gpt-tagger-frontend), no configuration file is needed as Vercel can natively handle the configuration for a React application.

Vercel allows for the definition of environment variables specific to each project. As such, all necessary environment variables are defined for both projects inside the Vercel platform. The link to the hosted application can be found in Appendix II.

5. Evaluation of the Prototype

This chapter presents the evaluation of the prototype application developed within this study. The primary objectives of this testing were to assess the application's functionality, relevancy of generated tags, the quality of translations from English to Estonian, user-friendliness, and gather general feedback for further improvement of the prototype. In the following subchapters, the methodology and results of the evaluation are presented.

5.1 Prototype Evaluation Methodology

The evaluation of the prototype application involved conducting usability testing through a Google Forms survey. The survey was designed with three sections, each aimed at assessing specific aspects of the prototype, which are described in the further subchapters. Before taking the survey, participants were introduced with a brief description of the survey purpose (incl., its objective, brief overview of the process, and the length) and were provided with a link to the prototype, as well as informed about consent for further use of collected data, specifying that the first 10 rows of the dataset uploaded are processed according to OpenAI's enterprise privacy[8]. See Appendix III for the survey in its complete form.

5.1.1 Section 1: Tagging Accuracy and Parameters Evaluation with Predefined Sample Dataset

The first part of the survey aimed to evaluate tagging accuracy of the prototype with pre-defined sample datasets. Participants were provided with instructions on the prototype use and links to two sample datasets sourced from the Estonian Open Data Portal. Participants were asked to try out the prototype by following the instructions on its use provided within the survey, by uploading respective datasets to the prototype. One of these datasets was about ongoing clinical trials in Estonia[9] and the other was about vehicle statuses in Estonia[10]. In the survey, respondents were required to answer the same set of questions for each dataset provided.

[8] https://openai.com/enterprise-privacy
[9] https://avaandmed.eesti.ee/datasets/kaimasolevad-kliinilised-uuringud-eestis
[10] https://avaandmed.eesti.ee/datasets/soidukite-staatused-eestis

The first question was "*How relevant are the generated tags to the actual content of the datasets?*". Participants were asked to assess the relevance of the generated tags to the actual content, constituting an acceptability task, where answers were defined using 5-point Likert scale, where 1 point corresponds to "not relevant at all" and 5 to "very relevant". If a low score was assigned, the participant was followed up with an additional question asking for a justification for this score.

Then, participants were asked to evaluate how the parameter "number of keywords" affects the relevancy of generated tags. The answers were four predefined options, namely "Yes, improves relevancy significantly", "Yes, improves relevancy slightly", "No, does not improve or worsen relevancy" and "No, rather worsens relevancy". If a negative answer was given, the respondent was followed up with the open-ended question "*If tags relevancy worsens, how and at which number of keywords?*". Afterwards, the participants were asked which LLM produced more relevant tags with options being "GPT-3.5-turbo", "GPT-4" and "Both had results of similar relevancy".

Finally, the respondents were asked to assess the combination of different parameters, with the question being "*Which combination of the options "number of keywords" and "model" seemed to produce the most relevant results?*". This question was open-ended.

Additionally, Estonian speakers were asked to assess the accuracy of Estonian translations of tags. As being a native speaker of Estonian was not a mandatory prerequisite for participating in the survey, this question was optional.

5.1.2 Section 2: Evaluation of Tagging Process with User Dataset

The second section of the survey provided participants with the opportunity to try the prototype application with their own datasets. While this section was optional, participants were encouraged to test the application with a dataset of their own choice, while providing links to Estonian Open Data Portal and European Data Portal, from which open dataset could be selected by them. After testing their dataset, participants were asked to share any observations or feedback they have regarding the tagging process. This feedback was collected to map potential areas of the prototype for improvement.

5.1.3 Section 3: General Feedback on the Prototype

The third section of the survey focused on gathering general feedback on the prototype application. Participants were asked to provide feedback on the overall user-friendliness of the application, whether they would consider using it in their workflow (developing questions following the Unified Theory of Acceptance and Use of Technology (UTAUT)[11] and Technology Acceptance Model (TAM)[12] constructs for evaluating technology adoption), and if they encountered any prototype operation errors or issues during testing.

In the first question of this section, the participant was asked to rate how user-friendly the prototype is, representing an acceptability task, with the answers defined using a 5-point Likert scale, where 1 point corresponds to “Not user friendly” and 5 points corresponds to “Very user friendly”. If a low score was given, the respondent was followed up with an open-ended question to specify why they found the prototype to not be user friendly.

Then, the participant was asked to rate the usefulness of the prototype, also constituting an acceptability task using a 5-point Likert scale, where 1 point corresponds to “Not useful at all” and 5 points corresponds to “Very useful”. If the respondent found the prototype to be insufficiently useful, they were followed up with a question asking them to justify their answer to the previous question.

The participant was then asked if they would use the prototype for the purpose of tagging datasets and if they ran into any unexpected behaviour or issues when using the prototype, with both questions being closed-ended with predefined answers “Yes” or “No”. If the respondent did run into unexpected behaviour or issues, they were followed up with an open-ended question asking them to describe the issue(s).

Finally, participants were given the opportunity to offer suggestions for improvement or features they would like to see implemented in future iterations of the application.

[11] https://www.jstor.org/stable/30036540
[12] https://pubsonline.informs.org/doi/10.1287/mnsc.46.2.186.11926

5.2 Evaluation Results

The survey was distributed through social media, emailing to Estonian Open Data Portal representatives and personal channels, gathering in total 22 responses. The survey was targeted at individuals who actively work or engage with datasets within their professional or personal domains.

5.2.1 Tagging Accuracy and Parameters Evaluation with Predefined Sample Datasets

The first mandatory question was "*How relevant are the generated tags to the actual content of the datasets?*". For both datasets, most respondents answered with a value of 4 or 5 (see Figure 12) with the average value being 4.4, i.e. predominantly relevant. There were no answers for the value 1 or 2. In cases where respondents found tags to be less relevant, they were asked the optional follow-up question on the reasoning behind low relevance score. Answers to this question revealed that while most tags were relevant to the dataset, some were overly specific, failing to encapsulate the broader essence of the datasets.

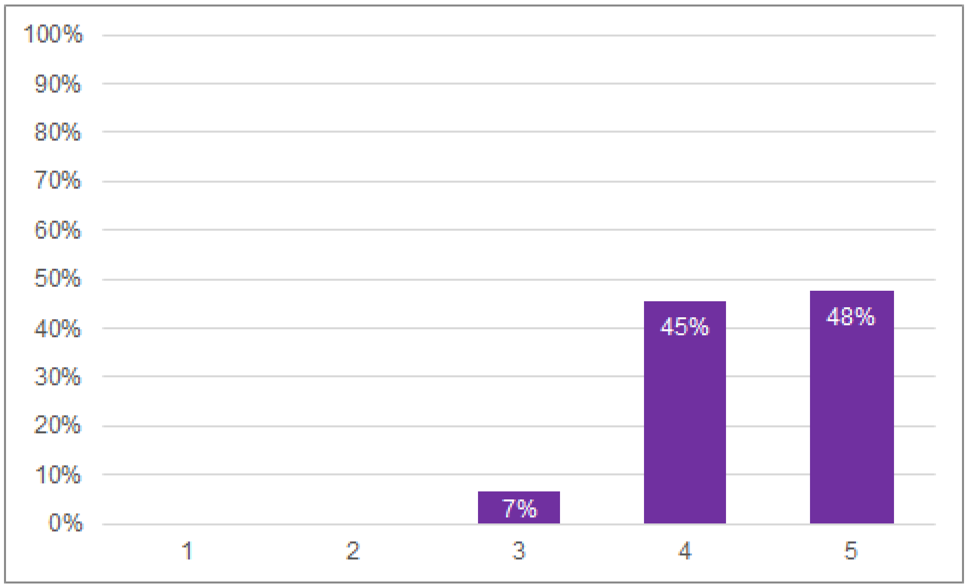


Figure 12. Ratings for relevancy of generated tags, based on answers from both datasets.

The second mandatory question was "*Does changing the "number of keywords" option affect the relevancy of the generated tags?*". About 74% of respondents reported that changing the number of tags to be generated improves relevancy with the largest share reporting that it improves slightly (see Figure 13). The question was succeeded by an optional question "*If tags relevancy worsens, how and at which number of keywords?*". From the obtained answers, a consensus emerged that increasing the number of tags generally enhanced accuracy or provided opportunities to discern more precise tags amid less accurate ones.

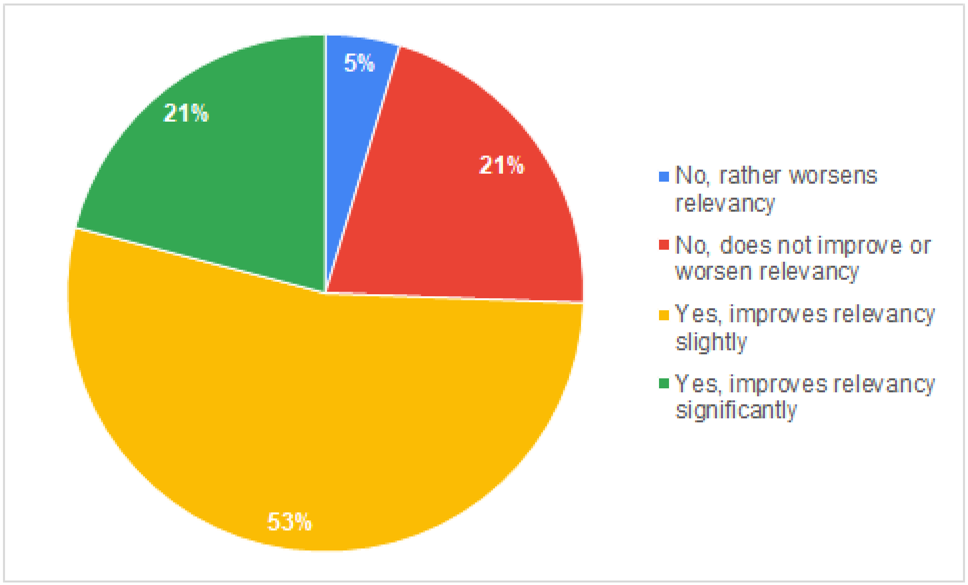


Figure 13. Does changing the "number of keywords" option affect relevancy of generated tags, based on answers from both datasets.

The third required question was "*Which model produced better tags?*". The majority of respondents (65%) highlighted that the best performing model was "GPT-4" (see Figure 14).

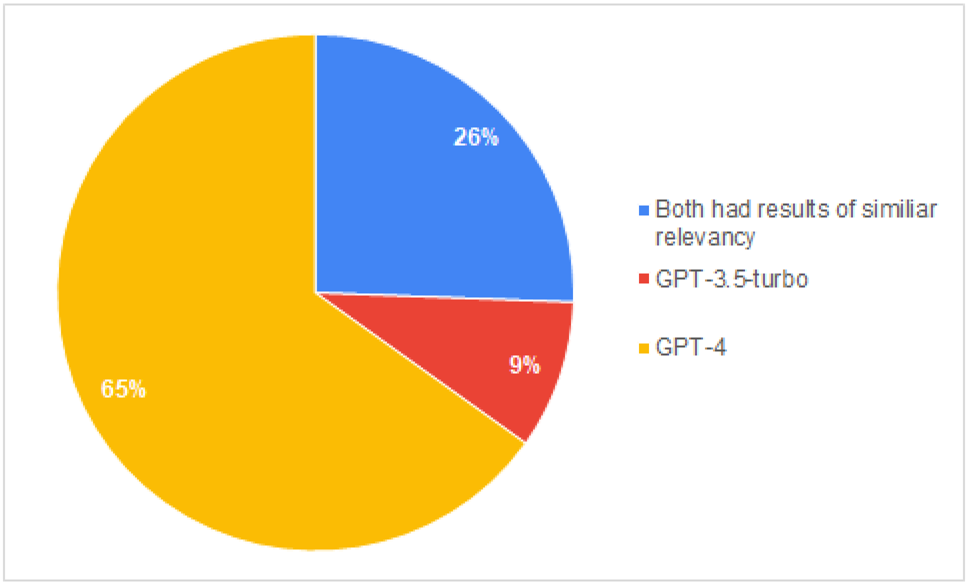


Figure 14. Which model produced better tags, based on answers from both datasets.

The final mandatory question in the first section was "*Which combination of the options "number of keywords" and "model" seemed to produce the most relevant results?*". The prevailing trend from these responses highlighted that the combination of GPT-4 and utilising five or more keywords appeared to consistently yield the most relevant outcomes for respondents.

Finally, Estonian speakers were asked to assess the accuracy of the Estonian tag translations. Most answers accumulated to the values of 4 and 5 (see Figure 15). Nobody answered with the values of 1 or 2.

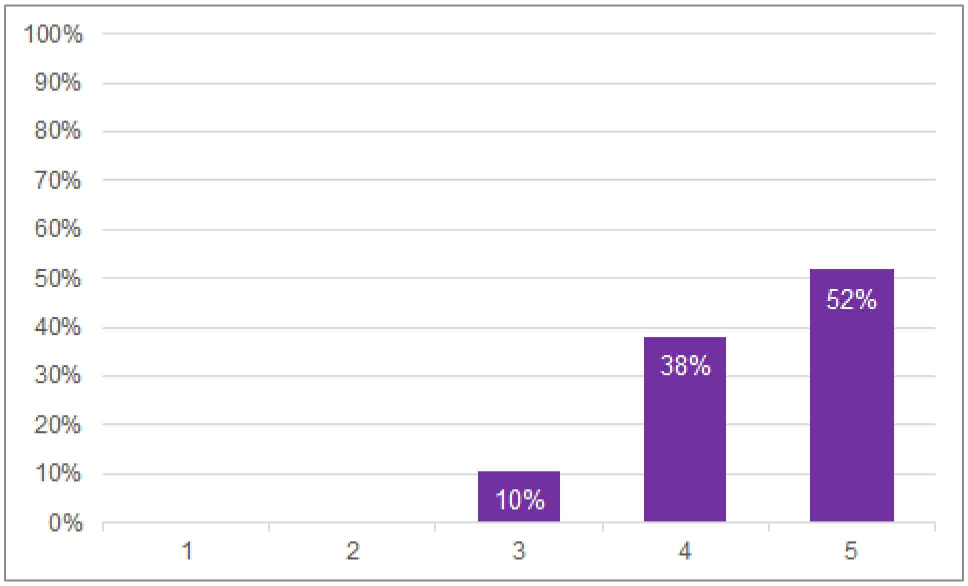


Figure 15. Accuracy of Estonian tag translations, based on answers from both datasets.

5.2.2 Evaluation of Tagging Process with User Dataset

Despite this part of the survey being optional, 12 respondents answered the question of this section. A variety of different answers were provided by participants, including comments that would be classified as feedback for further improvement of the prototype. Answers that can be categorised as feedback will be elaborated in chapter 5.2.3. Otherwise, several participants found that GPT-4 generally performs better with generating tags, also it was pointed out that in some rare cases the LLM returns incomprehensible output instead of relevant tags. Furthermore, several comments were made about the Estonian translations differing when using GPT-3.5-turbo and GPT-4, although these models were not used for translation, as translation service was used to translate the English tags to Estonian (refer to chapter 3 and 4). As such, an oversight that can be derived from this feedback is that users should have been informed that translations are performed by a separate service.

5.2.3 General Feedback on the Prototype

The first question of this section was "*How user-friendly is the prototype?*". 12 (54.5%) participants gave it a score of 4, whilst only 2 (9%) respondents gave the highest score of 5 (see Figure 16). For the justification of lower user-friendliness scores, participants pointed out the following concerns:

- files had to be reuploaded any time the user wanted to change parameters such as "number of keywords" or "model";
- prototype was limited to only one file type, namely .csv;
- file size limit was not specified.

Regarding the latter - absence of a specified size limit, the prototype was designed to operate without imposing an arbitrary file size restriction (refer to chapter 4.2.3), whereas users were not informed that there is no size limit, which caused this comment.

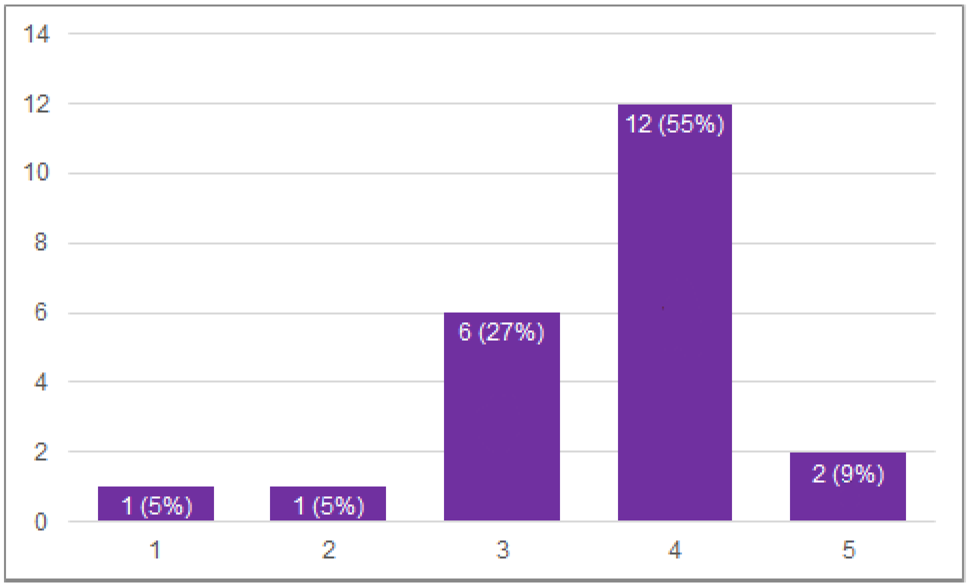


Figure 16. User-friendliness of the prototype.

The second question was "*How useful is the prototype?*". 12 (54%) respondents rated the usefulness of the prototype with the score 4, while 6 (27%) participants found it to be very useful, thus giving it a score of 5 (see Figure 17). Respondents gave the following reasons for lower usefulness scores:

- the LLM has a hallucination problem, which means it sometimes produces irrelevant tags;
- it is a standalone tool, it would be more useful if it was integrated into an open data portal;
- multiple different combinations of "number of keywords" and "model" must be tried in order to find optimal tags.

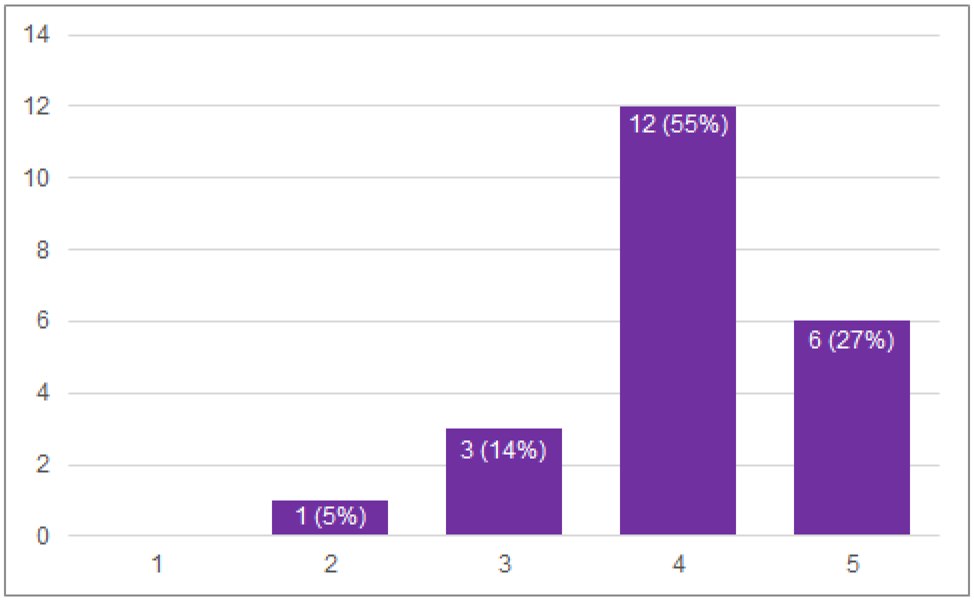


Figure 17. Usefulness of the prototype.

Then, participants were asked if they would use the prototype for the purpose of tagging datasets. 18 (82%) people answered “Yes”, while 4 (18%) answered “No”. The final mandatory question asked the participants if they encountered any unexpected behaviour or issues in the prototype, with 19 (86%) answering “No” and 3 (14%) answering “Yes”. The main issue users encountered was the prototype generating a different number of tags than was actually selected in the “number of keywords” option. Finally, participants left the following suggestions for further improvement of the prototype:

- possibility to approve or disprove the tags coming from the model;
- improvement of tagging accuracy;
- option to export results.

All of the points made in the question about rating the user-friendliness of the prototype can be considered as further improvements of the prototype. They are discussed in the next chapter.

6. Discussion

This chapter provides an overview of the results of the thesis. The feedback received from users is discussed and potential future improvements of the developed solution are outlined.

6.1 Feedback

The prototype developed within this thesis garnered positive feedback from participants in several key areas. Firstly, respondents generally rated the relevance of the generated tags highly, with an average rating of 4.4 out of 5. This indicates that the prototype effectively captured the essence of the datasets. Moreover, a significant portion of participants reported that adjusting the "number of keywords" option improved tag relevancy, suggesting flexibility in fine-tuning the tagging process. Additionally, the majority of respondents favoured the "GPT-4" model for its superior performance in tag generation compared to GPT-3.5-turbo that was originally selected for its superiority over other LLMs (refer to chapter 3.3.1). The combination of "GPT-4" with five or more keywords emerged as the most effective strategy for producing relevant tags consistently. Furthermore, Estonian speakers generally expressed satisfaction with the accuracy of the Estonian tag translations.

Despite the positive reception, user feedback identified areas for improvement in the prototype. Notably, some participants encountered instances, where the application produced irrelevant tags or incomprehensible output. In addition, some feedback highlighted that certain tags were overly specific, failing to encapsulate the broader content adequately. Furthermore, feedback regarding user interface and functionality emphasised concerns, such as the need to reupload files when adjusting parameters, limitations in supported file types and the standalone nature of the tool.

6.2 Future Improvements

In order to improve the usefulness of the application, issues with tagging accuracy (although pointed to by a minority of participants) must be addressed. These issues could be addressed by refining the initial system prompt provided to the LLM or by supplying more than just the first 10 rows of dataset content for the LLM to analyse. Although experimentation has shown that 10 rows is one of the lowest thresholds that still allows the LLM to generate relevant tags, where every additional row provided for analysis would increase the computational resources required, thus making the process more expensive (refer to chapter 3.1), increasing the amount of data the LLM processes allows it to make better generalisations based on the dataset.

Furthermore, feedback regarding user interface and functionality emphasised concerns such as the need to reupload files when adjusting parameters, limitations in supported file types and the standalone nature of the tool. These recommended improvements to the user interface can be implemented in the future to enhance user experience, particularly focusing on the interaction with the file upload logic. This includes expanding the file type support to common formats such as JSON, HTML, XLS, XLSX, and XML, and ensuring parameter values can be changed dynamically without the need to re-upload the dataset. The latter, namely, stand-alone nature of the tool stressed by evaluators, however, is due to the fact that the evaluated artefact is a prototype, which was made publicly available by hosting it as a stand-alone tool exclusively for its testing purposes. As such, once it is improved to meet evaluators expectations, it is expected to be integrated with existing open data portals, thereby broadening accessibility and utility for a wider audience.

Conclusion

This thesis aimed to address the challenge of poor data findability and metadata quality associated with open datasets. The principles of open government data and FAIR were explored and general data findability issues were presented, particularly those linked to dataset tags. An analysis of the Estonian Open Data Portal was conducted, which unearthed prevalent issues with data findability in the portal.

During the development of the thesis, a prototype application was developed that automates the tagging process by employing large language models - GPT-3.5-turbo and GPT-4. The development of such a tool presents significant benefits for both data publishers and consumers. Automatic tagging can reduce the risk for data publishers of publishing datasets that lack tags, which is a common issue on portals where tag inclusion is not mandatory, e.g., Estonian OGD portal, where 11% datasets have no associated tags and 26% datasets have only one tag assigned to them. Additionally, this automation minimises the association of datasets with incomplete or inaccurate tags. As a result, it improves the findability and accessibility of datasets, facilitating easier access for users.

The application was developed as a web service. This approach was chosen to ensure that the project is not limited to the Estonian Open Data Portal or OGD portals in general, making it environment-agnostic and interoperable with other products. The web service was implemented using FastAPI, an efficient framework for building APIs. Additionally, a web-based user interface was created using React to facilitate usability testing. The prototype was deployed into Vercel cloud, which spared users from the need to set up the application locally.

In assessing the prototype, a survey was administered, garnering 22 responses. Participants assessed various aspects of the application through its thorough examination, including the relevance of generated tags, user-friendliness, and overall usefulness, which were generally positively assessed by them. The feedback provided by respondents was used to identify areas for future improvement of the prototype.

The overarching goal of this thesis was realised - automatic tagging demonstrated its potential of providing relevant tags for datasets, thereby serving as a valuable tool for data publishers. In doing so, this thesis contributes to the realm of open data, paving the way for improved data findability and reusability.

References

[1] Sáez Martín A., Rosario A. H. D., Pérez M. C. C. 2016. An International Analysis of the Quality of Open Government Data Portals. *Social Science Computer Review*, 34(3), 298-311. doi: 10.1177/0894439315585734

[2] Skopal T., Klímek J., Nečaský M. 2020. Improving Findability of Open Data Beyond Data Catalogs. *In Proceedings of the 21st International Conference on Information Integration and Web-based Applications & Services (iiWAS2019)*. Association for Computing Machinery, New York, NY, USA, 413–417. doi: 10.1145/3366030.3366095

[3] de Castro B. P. C., Rodrigues H. F., Lopes G. R., Campos M. L. M. 2019. Semantic enrichment and exploration of open dataset tags. *In Proceedings of the 25th Brazillian Symposium on Multimedia and the Web (WebMedia '19)*. Association for Computing Machinery, New York, NY, USA, 417–424. doi: 10.1145/3323503.3349562 (04.12.2023)

[4] Riley J. 2017. Understanding metadata. National Information Standards Organization. http://www.niso.org/publications/press/UnderstandingMetadata.pdf (04.12.2023)

[5] Neumaier S., Umbrich J., Polleres A. 2016. Automated Quality Assessment of Metadata across Open Data Portals. *Data and Information Quality*, Volume 8, 1, Article 2. doi: 10.1145/2964909

[6] Alamo T., Reina D.G., Mammarella M., Abella A. 2020. Open data resources for fighting covid-19. arXiv, preprint. doi: 10.48550/arXiv.2004.06111

[7] Quarati A., De Martino M., Rosim S. 2021. Geospatial open data usage and metadata quality. *ISPRS international journal of geo-information*, 10(1), 30. doi: 10.3390/ijgi10010030

[8] Ubaldi B. 2013. Open Government Data: Towards Empirical Analysis of Open Government Data Initiatives. *OECD Working Papers on Public Governance*, No. 22, OECD Publishing, Paris. doi: 10.1787/5k46bj4f03s7-en

[9] Organisation for Economic Co-operation and Development. Open Government Data. https://www.oecd.org/gov/digital-government/open-government-data.htm (04.12.2023)

[10] Open Data Charter. ODC Principles. https://opendatacharter.org/principles/ (15.05.2023)

[11] Republic of Estonia Information System Authority. Estonian open data portal. https://www.ria.ee/en/state-information-system/data-based-governance-and-reuse-data/estonian-open-data-portal (08.05.2024)

[12] Wilkinson M., Dumontier M., Aalbersberg I. et al. 2016. The FAIR Guiding Principles for scientific data management and stewardship. *Sci Data*. doi: 10.1038/sdata.2016.18

[13] FAIR Data Maturity Model Working Group. 2020. FAIR Data Maturity Model. Specification and Guidelines. Zenodo. doi: 10.15497/rda00050

[14] GO FAIR. FAIR Principles. https://www.go-fair.org/fair-principles/ (28.02.2024)

[15] Jati P. H. P., Lin Y., Nodehi S., Cahyono D. B., van Reisen M. 2022. FAIR versus open data: A comparison of objectives and principles. *Data Intelligence*. 4(4), 867-881. doi: 10.1162/dint_a_00176

[16] Máchová R., Lněnička M. 2017. Evaluating the Quality of Open Data Portals on the National Level. *Journal of theoretical and applied electronic commerce research*, 12, 21-41. doi: 10.4067/S0718-18762017000100003

[17] Joni S., Yoganathan V., Corporan J. et al. 2019. Machine learning approach to auto-tagging online content for content marketing efficiency: A comparative analysis between methods and content type. *Journal of Business Research*, Volume 101, 203–217. doi: 10.1016/j.jbusres.2019.04.018

[18] Braunschweig K., Eberius J., Thiele M., Lehner W. 2012. The State of Open Data Limits of Current Open Data Platforms.

[19] Weigel T., Schwardmann U., Klump J., Bendoukha S., Quick R. 2020. Making Data and Workflows Findable for Machines. *Data Intelligence*. 2 (1-2), 40–46. doi: 10.1162/dint_a_00026

[20] Rajamäe-Soosaar K., Nikiforova A. 2024. Exploring Estonia's Open Government Data Development as a Journey towards Excellence: Unveiling the Progress of Local Governments in Open Data Provision. *In Proceedings of the 25th Annual International Conference on Digital Government Research*.

[21] Estonian Open Data Portal. API manual. https://avaandmed.eesti.ee/instructions/api-manual (08.05.2024)

[22] Trabelsi M, Cao J, Heflin J. 2020. Semantic Labeling Using a Deep Contextualized Language Model. doi: 10.48550/arXiv.2010.16037

[23] OpenAPI. Prompt engineering. https://platform.openai.com/docs/guides/prompt-engineering/strategy-write-clear-instructions (22.04.2024)

[24] Oracle. What Are RESTful Web Services?

https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html (22.04.2024)

[25] Bigelow S. J., Lewis S. 2024. Web services. Tech Target. https://www.techtarget.com/searchapparchitecture/definition/Web-services (16.04.2024)

[26] Refuel. 2023. LLMs can label data as well as humans, but 100x faster. https://www.refuel.ai/blog-posts/llm-labeling-technical-report (16.04.2024)

[27] FastAPI. https://fastapi.tiangolo.com/ (16.04.2024)

[28] FastAPI. Benchmarks. https://fastapi.tiangolo.com/benchmarks/ (02.05.2024)

[29] OpenAI. API reference https://platform.openai.com/docs/api-reference (16.04.2024)

[30] Desai J. 2023. Web Application Vs Desktop Application: Pros and Cons. Positiwise. https://positiwise.com/blog/web-application-vs-desktop-application-pros-and-cons (16.04.2024)

[31] Vailshery L. S. 2023. Most popular web frameworks among developers worldwide 2023. Statista. https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/ (02.05.2024)

[32] Node.js. https://nodejs.org/en (02.05.2024)

[33] React. https://react.dev/ (16.04.2024

[34] Hungbo M. 2024. Node vs. React: The Difference and Which Framework to Choose. Ninetailed. https://ninetailed.io/blog/node-js-vs-react-js/ (02.05.2024)

[35] Intento. Independent Multi-Domain Evaluation of Machine Translation Engines. 2021. https://fs.hubspotusercontent00.net/hubfs/3317859/Intento%20State%20of%20Machine%20Translation%202021.pdf (22.04.2024)

[36] Ahmed A. 2023. What is Vercel and Why You Should Use It? Fishtank. https://www.getfishtank.com/blog/what-is-vercel (23.04.2024)

[37] Vercel. Functions. https://vercel.com/docs/functions (13.05.2024)

[38] Watts S. 2023. The Serverless Functions Beginner's Guide. Splunk. https://www.splunk.com/en_us/blog/learn/serverless-functions.html (23.04.2024)

[39] Narwhal Technologies. https://monorepo.tools/ (08.05.2024)

[40] Vercel. Monorepos. https://vercel.com/docs/monorepos (08.05.2024)

Appendix

I. Prototype Source Code

The source code of the prototype application can be found on the following link: https://github.com/kevinkliimask/gpt-tagger

II. Prototype Application

The prototype application can be found on the following link: https://gpt-tagger-frontend.vercel.app/

III. Survey

Introduction

Automated datasets tagging for an improved dataset findability

Hello!

My name is Kevin Kliimask. I am a 3rd year Computer Science student currently working on my Bachelor's thesis. The goal of my thesis is to improve data findability on open government data portals.

I've created a prototype website where you can have a Large Language Model (LLM) automatically tag your dataset. You can choose how many tags you wish to receive for the dataset and which model to use for the tagging.

I would like to ask you to evaluate the developed prototype by filling in this survey. The prototype is available at https://gpt-tagger-frontend.vercel.app/. Filling the survey is expected to take about 10-15 minutes

NOTE: Datasets uploaded to the website are not stored by the website's author. Only the first 10 rows of the dataset uploaded are processed according to OpenAI's enterprise privacy.

* Indicates required question

__First section (questions repeat twice for each dataset)__

Please answer the following questions based on the results retrieved from the website by using 2 sample datasets.

The first sample dataset can be found and downloaded from here: https://avaandmed.eesti.ee/datasets/kaimasolevad-kliinilised-uuringud-eestis (scroll down and click "Download" for the variant with CSV format)

The second sample dataset can be found and downloaded from here:
https://avaandmed.eesti.ee/datasets/soidukite-staatused-eestis (scroll down and click "Download" for the first variant in the table)

How relevant are the generated tags to the actual content of the datasets? You can compare them to the title, description and keywords of the dataset.*
5-point Likert scale, where 1 point corresponds to “not relevant at all” and 5 to “very relevant”

If the generated tags were not relevant, why?
Open-ended question

Does changing the "number of keywords" option affect the relevancy of the generated tags?*

- Yes, improves relevancy significantly
- Yes, improves relevancy slightly
- No, does not improve or worsen relevancy
- No, rather worsens relevancy

If tags relevancy worsens, how and at which number of keywords?
Open-ended question

Which model produced better tags?*

- GPT-3.5-turbo
- GPT-4
- Both had results of similar relevancy

Which combination of the options "number of keywords" and "model" seemed to produce the most relevant results?*
Open-ended question

(for Estonians) How accurate were the Estonian tag translations?
5-point Likert scale, where 1 point corresponds to “inaccurate” and 5 to “accurate”

<u>Second section</u>

(Optional) Give feedback on the prototype based on your own selected datasets.

You can use any csv format datasets. For example, you can find datasets to use on the Estonian open data portal (https://avaandmed.eesti.ee/datasets) or on the European data portal (https://data.europa.eu/data/datasets).

Did you make any observations based on your experimentation you would like to share with us?
Open-ended question

<u>Third section</u>

General feedback on the prototype

How user-friendly is the prototype?*
5-point Likert scale, where 1 point corresponds to “not user friendly” and 5 to “very user friendly”

If you found it to be insufficiently user-friendly, please specify
Open-ended question

How useful is the prototype?*
5-point Likert scale, where 1 point corresponds to “not useful at all” and 5 to “very useful”

If you found it to be insufficiently useful, please specify
Open-ended answer

Would you use the prototype for the purpose of tagging datasets?*

- Yes
- No

Did you encounter any unexpected behaviour or issues in the prototype?*

- Yes
- No

If you answered yes, what did you run into?
Open-ended question

Do you have any suggestions to improve the prototype?
Open-ended question

IV. Licence