



Technical Report on the Development of the All-to-En-Tags Model with a Focus on Serbian, German, and English Language Integration

Authors: Saša Stjepanović (Matr. -Nr. 03756216), Leon Kogler (Matr. -Nr. 03758706)
and Oluseyi Stephen Adedara (Matr. -Nr. 03768481)

Subject: Seminar Computational Social Science: Generative AI and Democracy

Submission Date: August 31, 2024

Contents

1.	Introduction	1
2.	Data Collection.....	2
2.1.	Language Differences	2
2.2.	Effective Data Collection.....	2
2.3.	Tag Generation	3
2.4.	Mitigating Bias	4
3.	Training the Model	5
3.1.	Categorization-Based Tagging	5
3.1.1.	Data Preparation.....	5
3.1.2.	Model Training	5
3.1.3.	Outcomes	5
3.2.	Instruction-Based Finetuning	6
3.2.1.	Model Selection and Configuration	6
3.2.2.	Prompt Design and Formatting	6
3.2.3.	Training Process.....	6
3.2.4.	Outcomes	7
4.	System Design and Implementation.....	8
4.1	System Overview.....	8
4.2	Tag Generation	9
4.2.1	tag-gen-llama3-model	9
4.3	Embedding	10
4.3.1	mxbai-embed-large Model.....	10
4.3.2	Embedding Process	10
4.4	Search Mechanism.....	12
4.5	File Database	14
4.5.1	ChromaDB Structure.....	15
4.5.2	Tag and Embedding Storage	16
4.6	Use of Langchain.....	18
5.	Challenges, Issues, and Solutions	19
5.1	Technical Challenges and Solutions.....	19
5.1.1	Efficiency in Embedding and Search.....	19

5.1.2 Integration with ChromaDB	20
5.1.3 Limited Server Resources	20
5.2 Non-Technical Challenges and Solutions	21
5.2.1 Resource and Data Constraints	21
5.2.2 Knowledge Gap in LLMs	21
5.3 Reflection on Solutions and Lessons Learned	22
6. Conclusion and Outlook	23
6.1 Summary of Findings.....	23
6.2 Future Directions	24
7. Appendix	25
7.1 GitHub Repository:	25
7.2 Code Documentation	25
8. Declaration	27
References	28

1.Introduction

In today's evolving digital world, the ability to retrieve relevant information regardless of language is becoming increasingly valuable. As organizations and individuals are publishing data all over the internet, the need for a model that recognizes relevant content and can provide a quick response to relevant articles could be beneficial. The usage of such a model can vary from simple questions, e.g. "What are the best restaurants in Germany?" or "What is the best beer in the Czech Republic?" to even political ones, e.g. "How is the current situation in Ukraine?" or "What is happening in Israel?" Asking those political questions can get very different and even biased opinions which is why the model development gets more interesting at some point. That is why we need a universal model on an unbiased level that points us to trustworthy data that is focused on facts. This needs to be further researched and developed to have a data source that individuals can trust more.

In this technical report, we will discuss the model we developed as a team to face the above-mentioned issue. Our project does not cover all articles ever published and written in every language across the internet. However, the main idea of how the issue can be fixed with the limit of the Serbian, German, and English languages is presented.

2. Data Collection

2.1. Language Differences

Given that the languages used in our model are Serbian, German, and English, it is no surprise that they each have different alphabets and writing systems. Serbian uses both Cyrillic and Latin alphabets, with Cyrillic being the official script. It also has additional characters like 'š', 'č', 'ć', 'đ' and 'ž' in the Latin script or 'ш', 'ч', 'ћ', 'ђ', and 'ж' in the corresponding Cyrillic script. German and English on the other hand exclusively use the Latin alphabet, though German also has additional characters like "ü", "ä", "ö", and "ß" and is famous for compound words which can be an important issue.

Luckily, almost every article that is written in Serbian is also available in Latin and by using already well-trained AI models, e.g. ChatGPT, Meta AI, and Google Gemini to later deal with tokenizing the words and generating tags from articles in our approach, the problems with language differences are going to be efficiently fixed. Additionally, for our model, the Latin script of all languages will be used as it makes the process of tokenization a lot easier.

2.2. Effective Data Collection

A common difficulty arises when data is being collected as efficient models require effective and good-quality data collection. A common technique is the retrieval of the data through an API or publicly available databases. However, traditional web scraping is also an option. The most significant issue with APIs and databases in general is that most of the time they do not offer the full content of articles that are needed for our purposes. Additionally, data retrieved from an API or databases may not always be up-to-date, especially if the API or databases are not maintained regularly. This can lead to incorrect

outputs which could significantly impact our model. Therefore, this approach is only used in a limited manner.

Our model requires each article to have a title, timestamp, and extensive content. Unfortunately, the third component is often, for the API and database approach, only available in the needed size if a large payment is made. Therefore, the traditional option of web scraping websites seems to be the most effective way of collecting the data as it is very flexible and provides most of the time the full content of a website.

However, due to the diverse layouts of websites, web scraping presents an additional challenge. To get the data from a website, a web scraping tool, e.g. BeautifulSoup in Python, can be used to make the process easier. The difficult part is the deep individual structure of websites, e.g. different news providers in several countries and languages have their own design and components. Therefore, the most effective way of collecting the data through web scraping for our model is to analyze the structure of different websites and then proceed to the next step which is the individual implementation of a scraping algorithm.

Referring to the Serbian language, the standardized version uses the Cyrillic alphabet which leads to a big issue regarding tokenization in the later process. Fortunately, almost every news provider offers a Latin version which for our purposes was considered and used when web scraping was done. As for the other two, namely German and English, no such issues can occur due to the Latin nature of the language.

2.3. Tag Generation

Our model uses tags to decide which articles are relevant to a user's query. Therefore, this part for generating tags is very important as it could lead to false results in the end. Using standard NLP (Natural Language Processing) libraries for generating tags often relies on predefined rules and keyword matching, which can limit their flexibility and

accuracy. These libraries are more likely to miss the needed context-specific tags because they work on generalized patterns rather than deep understanding. In contrast, generative AI models, e.g. ChatGPT, Google Gemini, and Meta AI have advanced techniques like contextual embeddings to generate tags. This difference makes the approach of using well-trained AI models instead of standard libraries for tag generation more attractive. Therefore, for our purposes, all three main generative AI models are being used to get the needed good-quality tags most efficiently.

For better understanding, the process of creating the tags is done by giving each main AI model the same prompt with the corresponding article. After generating tags from the named models, the most relevant and accurate tags are selected. These chosen tags are then saved for use in training or for querying the model.

2.4. Mitigating Bias

The model should generally give outputs that are unbiased and correct. To achieve this the model uses data not only from one but different news providers, even though a particular news provider may offer more data from which the model could benefit more. Referring to the Serbian language, five different news providers were used to mitigate the bias, and the number of articles was limited to a certain amount, e.g. five thousand per news provider to not give a certain data provider an advantage. As for the other two languages, the methods of retrieving the data from APIs and databases were also added.

For the process of generating tags from articles, several generative AI models were used, e.g. ChatGPT, Google Gemini, and Meta AI. By utilizing various AI models, we ensured that the tags were generated from multiple perspectives, reducing the risk of bias that could arise by relying on just one AI model. This approach created a more balanced and less biased set of tags.

3. Training the Model

3.1. Categorization-Based Tagging

In this approach, we treated the task of generating tags for an article as a multi-label classification problem. An article was categorized by assigning a set of predefined tags (labels) based on the content. The model, a variant of the DeBERTa-v3 architecture, was trained to predict the presence or absence of each tag from a set of potential tags.

3.1.1. Data Preparation

The dataset was filtered to retain only those samples that contained frequently occurring tags. This was done to ensure the model had sufficient data to learn from for each tag. The tags were mapped into a format where each tag was assigned a specific index, and if a tag was present in an article, that index was marked with a 1.

3.1.2. Model Training

The model was fine-tuned using the filtered dataset with a specific focus on predicting the correct set of tags for each article. The training was carried out over multiple epochs, with evaluation happening at the end of each epoch.

3.1.3. Outcomes

Despite the structured approach, several challenges were encountered that led to suboptimal results. The primary challenge was the imbalance in the distribution of tags. Some tags appeared very frequently, while others were rare, resulting in the model being trained on a highly imbalanced dataset. After filtering, the dataset became relatively small. From originally over 6000 different tags, there were only 362 tags in the dataset that occurred more than 15 times. We tested the model with various prompts, but it consistently

produced similar results regardless of the input. There was no clear connection between the given sentence and the tags that the model generated with the highest likelihood.

3.2. Instruction-Based Finetuning

In this approach, we focused on instruction-based finetuning of a LLM to generate relevant tags for articles. Unlike the previous categorization-based method, this approach leverages the model's ability to follow detailed instructions, making it more adaptable to the task of generating tags based on the content of the articles.

3.2.1. Model Selection and Configuration

We used the unsloth/llama-3-8b-bnb-4bit model, a 4-bit quantized version of Llama-3, to reduce memory usage and improve efficiency. The model was further fine-tuned using Parameter-Efficient Fine-Tuning (PEFT) techniques, specifically LoRA (Low-Rank Adaptation), which allowed us to reduce the training parameters from 4.65 billion to just 41 million, optimizing specific layers for the task at hand.

3.2.2. Prompt Design and Formatting

We crafted detailed prompts to guide the model. The prompts included an instruction that the model should behave as a helpful assistant tasked with generating unbiased, English-language tags representing the core ideas of an article. These prompts were then formatted into a structure, ensuring that the response was limited to generating only the translated English tags we can further use in our application.

3.2.3. Training Process

We ultimately selected 700 articles across all three languages, each paired with corresponding English tags, for training. With a total batch size of 8 and training over 2

epochs on a NVIDIA T4 GPU with 16GB of memory, the process took approximately 2 hours. We deployed our model by saving the pretrained model, enhanced with the LoRA adapter, as a GGUF file with Q4 quantization. The resulting model, approximately 4.9 GB in size, can be efficiently run locally using Ollama.

3.2.4. Outcomes

During testing, we encountered issues where the model sometimes generated summaries of the article or exhibited other unintended behaviors, in addition to providing the required tags. Despite using a carefully crafted prompt designed to elicit a list of tags only, the model occasionally deviated from this instruction.

We employed specific techniques to address these issues, including setting constraints on the output format and using filtering methods. However, these solutions were not entirely reliable, and the model still occasionally produced outputs that did not align with our expectations.

We ultimately chose the Instruction-Based Finetuning approach over the multi-label classification method due to its significantly better performance and ease of training. The instruction-based approach provided more accurate and relevant results, even with a relatively small amount of data. It proved to be more effective in generating high-quality tags and was easier to implement, making it a more practical choice for our needs.

Future work should focus on improving the reliability of the instruction-based finetuning method by refining prompt design, tweaking the training parameters, and exploring more robust data handling techniques. Expanding the dataset and optimizing the fine-tuning process could further enhance performance.

4. System Design and Implementation

This chapter details the technical solutions developed to address the challenges identified in building a multilingual search engine. It covers our system, the models used for tag generation and embedding, the process of similarity search, and the integration of Langchain to manage these processes efficiently.

4.1 System Overview

The system was designed to address the core challenges of processing and retrieving multilingual content. The architecture comprises several key components, each responsible for a specific task in the pipeline, from data ingestion to result retrieval.

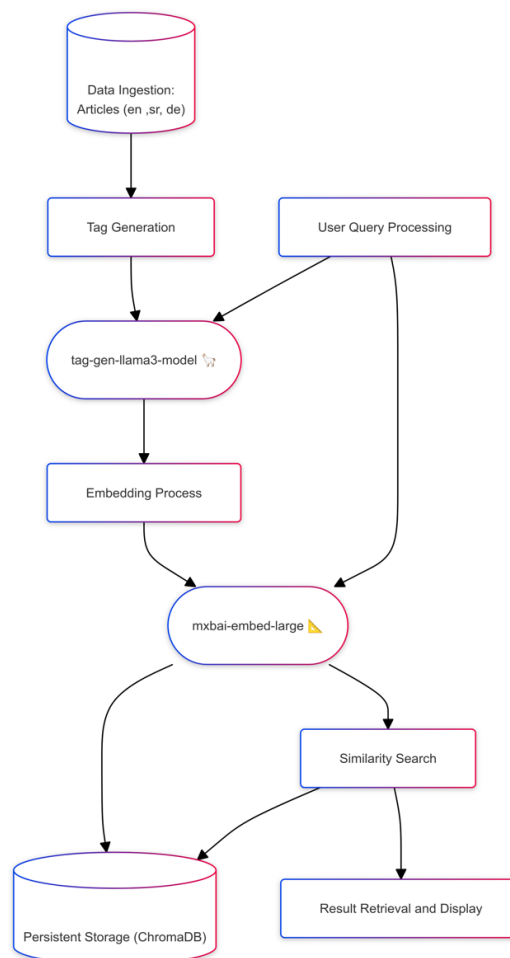


Figure 4.1 System Overview Diagram

The diagram above shows the flow of data through various stages:

- **Data Ingestion:** Articles in German, Serbian, and English are ingested into the system.
- **Tag Generation:** The *tag-gen-llama3* model generates English tags for each article.
- **Embedding Process:** These tags are then embedded using the *mxbai-embed-large* model.
- **Storage:** Embeddings are stored in ChromaDB, with data persisted to a directory for durability.
- **Query Processing:** User queries are processed in a similar manner—tags are generated, embedded, and compared to stored article embeddings using cosine similarity.
- **Result Retrieval:** The most relevant articles are retrieved and presented to the user based on the similarity scores.

4.2 Tag Generation

As discussed in the previous chapter, one of the core components of the system is the generation of tags for articles and user queries. This task was handled by what we will call the *tag-gen-llama3-model*, which was specifically fine-tuned for this purpose.

4.2.1 tag-gen-llama3-model

The *tag-gen-llama3-model* which we host on Ollama.com is based on the unsloth/llama-3-8b-bnb-4bit model. This model can generate accurate and relevant tags, which are essential for our embedding and similarity search processes. The model was

fine-tuned using a combination of categorization-based tagging and instruction-based finetuning techniques.

- **Categorization-Based Tagging:** Initially, a categorization-based tagging approach was attempted, treating tag generation as a multi-label classification problem. However, due to the challenges of imbalanced data distribution and suboptimal results, this approach was supplemented with instruction-based finetuning.
- **Instruction-Based Finetuning:** The final model was fine-tuned using detailed prompts that guided the model to generate relevant English-language tags. This approach allowed the model to better adapt to the diverse content of the articles, resulting in higher quality and more relevant tags [1].

4.3 Embedding

This section describes the process of generating embeddings for tags produced by a *tag-gen-llama3* based model for both articles and user queries. These embeddings are then used in a similarity search to retrieve relevant articles from a pre-existing database

4.3.1 mxbai-embed-large Model

The mxbai-embed-large model was used to convert the English tags into embeddings. This model was selected for its ability to generate embeddings that accurately capture the semantic content of the tags, ensuring that similar tags produce similar embeddings [2].

4.3.2 Embedding Process

The embedding process converts tags generated from articles and user queries into numerical vectors (embeddings). These embeddings are then used to

compare and retrieve relevant articles based on the similarity between the query and article content.

Embedding Articles:

- **Tag Generation for Articles:**

- All articles are processed using the *tag-gen-llama3* model to generate English tags. These tags summarize the key content of each article, regardless of the original language of the article (German, Serbian, or English).

- **Embedding Tags for Articles:**

- **Embedding Process:** Once the English tags are generated for each article, they are converted into embeddings using the *mxbai-embed-large* model. These embeddings are high-dimensional vectors that represent the semantic content of the articles.

- **Preprocessing Workflow:**

- **Normalization:** The tags are preprocessed to ensure they are in the correct format for embedding. This includes steps like normalization.
- **Batch Processing:** The tags from multiple articles are embedded in batches, optimizing the process for speed and efficiency.

- **Storage in Database:**

- The embeddings generated from the article tags are stored in a Chroma File DB, making them readily accessible for future similarity searches.

Generating Embeddings for User Queries:

- **Tag Generation for Queries:**

- When a user submits a search query, it is processed using the same *tag-gen-llama3* model to generate English tags that reflect the core meaning of the query.

- **Embedding Tags for Queries:**

- **Embedding Process:** The tags generated from the user query are embedded using the *mxbai-embed-large* model, resulting in a vector representation of the query's content.
- **Efficiency Considerations:** Since query processing must be done in real-time, the embedding workflow is optimized to ensure rapid response times.

4.4 Search Mechanism

Cosine Similarity Search:

Cosine similarity is used to measure the similarity between the embedding generated from a user query and the embeddings generated from the article tags. This process identifies which articles are most relevant to the user's search based on the similarity of the tags.

Implementation:

- **Similarity Calculation:**

- **Cosine Similarity Computation:** After generating the embedding for the user query, the system calculates the cosine similarity between this query embedding and each of the article embeddings stored in the database. This similarity score determines how closely the articles match the user's query.

- **Comparison with Pre-Existing Embeddings:** The system efficiently compares the query embedding against all stored article embeddings using vectorized operations, which are optimized for quick calculations.
- **Ranking and Filtering Results:**
 - **Sorting by Similarity:** Articles are ranked based on their cosine similarity scores relative to the query embedding. The highest-scoring articles are considered the most relevant and are presented to the user.
 - **Relevance Threshold:** A threshold may be applied to filter out articles with low similarity scores, ensuring that only the most relevant articles are shown to the user.

Integration with User Queries:

- **User Interface and Query Handling:**

The search interface allows users to input queries, which are then processed through the embedding and similarity search pipeline. The steps include:

Example Workflow:

1. **User Input:** The user enters a query (e.g., “Impacts of climate change in Europe”).
2. **Tag Generation:** The query is processed by the *tag-gen-llama3* model to generate English tags (e.g., “climate change impacts Europe”).
3. **Embedding:** The tags are embedded into a vector using the *mxbai-embed-large* model.

4. **Similarity Search:** The query embedding is compared against the article embeddings stored in the database.
5. **Result Display:** The most relevant articles, based on cosine similarity scores, are returned and displayed to the user.

Evaluation:

- **Accuracy:**

- **Relevance Testing:**

The search mechanism's accuracy was evaluated by testing it with a variety of user queries. The goal was to ensure that the system accurately captures the intent behind user queries and retrieves the most relevant articles.

- **Performance:**

- **Response Time:**

The system's performance in terms of query response time was measured. The embedding and similarity search processes were optimized to maintain fast response times, even with large datasets or complex queries.

4.5 File Database

This section details the storage and management of tags and embeddings within the ChromaDB, with a focus on the persistence of data to a directory instead of storing it in memory. This approach ensures that embeddings are stored efficiently and can be retrieved quickly during similarity searches.

4.5.1 ChromaDB Structure

Overview of ChromaDB:

ChromaDB is used to store the embeddings generated from the English tags of articles. It is a vector store designed to handle high-dimensional vector data efficiently, making it an ideal choice for managing and querying the embeddings used in similarity searches [3].

- **Persistence Mechanism:**

- **Persistent Storage:** Instead of storing data in memory, ChromaDB is configured to persist data to a directory. This ensures that all embeddings and associated metadata are stored on disk, allowing for easy recovery and scalability.
- **Directory Setup:** The embeddings are stored in a directory specified by the *persist_directory* parameter. For example, the embeddings might be stored in a directory like *“./chroma”*, ensuring they are saved to disk.

- **Database Schema:**

- **Tag Storage:** Each article’s English tags, generated by the llama-7b-based model, are stored along with the article in the ChromaDB. These tags are indexed for quick retrieval and used in the embedding process.
- **Embedding Storage:** Embeddings generated from these tags are stored as high-dimensional vectors. The ChromaDB manages these

embeddings, indexing them to support fast and accurate similarity searches.

- **Indexing Mechanism:**

- **Vector Indexing:** The embeddings stored in ChromaDB are indexed using appropriate vector-based algorithms, like HNSW (Hierarchical Navigable Small World) with cosine similarity as the distance metric. This indexing allows for efficient similarity searches, ensuring quick response times even with large datasets.

4.5.2 Tag and Embedding Storage

Tag Management:

- **Tag Generation and Storage:**

Tags are generated using the *tag-gen-llama3* model and stored within ChromaDB as part of each article's record. These tags are critical for both the embedding process and for understanding the content of each article in a human-readable format.

Embedding Management:

- **Storing Embeddings:**

Once tags are generated and embedded using the *mxbai-embed-large* model, the resulting vectors are stored in ChromaDB. These embeddings are saved persistently to a directory, ensuring they are not lost between sessions and can be quickly accessed when needed.

- **Updating Embeddings:**

Like tags, embeddings must be updated if the content of an article changes or if improvements are made to the embedding model. ChromaDB supports updating existing records to reflect these changes.

Database Maintenance:

- **Data Consistency:**

Ensuring data consistency is crucial, especially when updating tags and embeddings. ChromaDB uses mechanisms like transactions to maintain consistency, ensuring that all related data is synchronized and accurate.

- **Backup and Recovery:**

Regular backups of the ChromaDB directory are essential to prevent data loss. By persisting data to disk, the system supports easy backup and recovery, ensuring that no data is lost in case of system failures or updates.

Scalability Considerations:

- **Scaling Up:**

As the number of articles grows, ChromaDB must scale to accommodate increasing volumes of data without degrading performance. Strategies like data partitioning, efficient indexing, and query optimization are implemented to ensure that the database remains performant as it scales.

- **Performance Tuning:**

Regular performance tuning is necessary to maintain the efficiency of the database. This includes optimizing the configuration of ChromaDB, updating indices, and managing the directory structure to ensure that similarity searches remain fast and accurate even with many stored embeddings.

4.6 Use of Langchain

Langchain was integrated into the system to streamline the processes of embedding, storage, and similarity search.

- **Integration with ChromaDB:** Langchain provided tools to efficiently manage the interaction between the embedding models and ChromaDB, ensuring that embeddings were stored and retrieved efficiently.
- **Efficiency Gains:** The use of Langchain allowed for optimized query handling and fast similarity searches, even with large volumes of data. This integration was crucial for maintaining the system's performance under real-world conditions.

5.Challenges, Issues, and Solutions

This chapter discusses the technical and non-technical challenges encountered during the development of the multilingual search engine, along with the solutions implemented to address these issues. The chapter also reflects on the effectiveness of these solutions and the lessons learned.

5.1 Technical Challenges and Solutions

5.1.1 Efficiency in Embedding and Search

- **Challenge:** The process of embedding large datasets and performing similarity searches posed significant performance challenges. Given the high dimensionality of the embeddings generated by the *mxbai-embed-large* model, ensuring efficient storage and fast retrieval was crucial to maintaining the system's overall performance.
- **Solution:** The embeddings were stored in ChromaDB, configured to persist data to a directory instead of memory. This setup allowed for efficient management of large volumes of data while ensuring quick access during similarity searches. Langchain was integrated to streamline the embedding and search processes, optimizing query handling and storage.
- **Outcome:** The integration of ChromaDB and Langchain successfully improved the system's efficiency, allowing for scalable storage and fast similarity searches. This solution enabled the system to handle larger datasets while improving performance.

5.1.2 Integration with ChromaDB

- **Challenge:** Integrating Chroma for persistent storage of embeddings introduced challenges related to data consistency and performance. Ensuring that the database could efficiently handle the high volume of embeddings and support real-time queries.
- **Solution:** The team configured ChromaDB to use vector indexing techniques, such as HNSW (Hierarchical Navigable Small World) with cosine similarity as the distance metric. This configuration allowed for faster similarity searches.
- **Outcome:** The optimized integration of ChromaDB with vector indexing significantly improved the system's ability to handle large-scale data while maintaining fast query response times.

5.1.3 Limited Server Resources

- **Challenge:** Initially, the server had insufficient RAM to run our models with Ollama. This limitation prevented us from running our application on the test server
- **Solution:** The issue was resolved by increasing the RAM on the server by our instructor. This upgrade provided the necessary memory resources to efficiently run the LLM.
- **Outcome:** With the increased RAM, the system was able to run the *tag-gen-llama3* model effectively.

5.2 Non-Technical Challenges and Solutions

5.2.1 Resource and Data Constraints

- **Challenge:** The project faced constraints related to hardware resources and data availability. The limited availability of high-quality multilingual datasets for training and testing was a particular challenge, affecting the robustness of the model's performance.
- **Solution:** To overcome data constraints, the team focused on curating a high-quality dataset of articles across three languages. The use of efficient quantization techniques also helped optimize model performance
- **Outcome:** Although the curated dataset was relatively small, the quality of the data and the efficiency of the training process allowed the model to perform well. However, expanding the dataset and improving data handling techniques would likely enhance future iterations of the project.

5.2.2 Knowledge Gap in LLMs

- **Challenge:** Initially, the team lacked extensive experience with large language models (LLMs), which created a significant knowledge gap that hindered early progress.
- **Solution:** To address this gap, the team underwent training on LLMs from our instructor and we were also provided additional study resources.
- **Outcome:** The training significantly improved the team's ability to work with LLMs, leading to more effective implementation of the *tag-gen-llama3* model and other components of the system. The knowledge gained also allowed the team to troubleshoot issues more effectively and optimize model performance.

5.3 Reflection on Solutions and Lessons Learned

- **Evaluation of Solutions:** The solutions implemented addressed many of the challenges faced during the project. The instruction-based finetuning approach for tag generation significantly improved accuracy, while the integration of ChromaDB and Langchain optimized performance and scalability. The increase in server RAM resolved the resource limitations, and the team's LLM training filled critical knowledge gaps.
- **Areas for Improvement:** If starting over, the team would focus on further refining prompt design, exploring more robust data handling techniques, and investing in better server hardware from the outset. Expanding the dataset and experimenting with additional model fine-tuning strategies could also enhance the system's performance.

6. Conclusion and Outlook

This final chapter summarizes the key outcomes of the project, reflects on the overall effectiveness of the approaches taken, and on potential improvements in this field of multilingual search engines.

6.1 Summary of Findings

- **Multilingual Tag Generation:**

The project successfully developed a system capable of generating relevant tags for articles in Serbian, German, and English using the *tag-gen-llama3* model. This method outperformed the initial categorization-based tagging approach, providing more accurate and contextually relevant tags. The use of detailed prompts allowed the model to adapt better to the diverse content of the articles, ensuring that the generated tags effectively represented the core ideas of the text.

- **Embedding and Similarity Search:**

The use of the *mxbai-embed-large* model to embed these tags into high-dimensional vectors, stored in ChromaDB, enabled efficient similarity searches. By leveraging cosine similarity, the system could accurately retrieve, and rank articles based on user queries.

- **Challenges and Solutions:**

Throughout the project, several challenges were encountered, including hardware limitations, data quality issues, and initial knowledge gaps in LLMs. These were addressed through a combination of technical solutions (e.g., server RAM upgrade, efficient quantization, structured training for the team) and methodological

adjustments. These solutions proved effective in overcoming the obstacles and enabling the successful implementation of the multilingual search engine.

6.2 Future Directions

- **Expanding the Dataset:**

Future work could focus on expanding the dataset to include more languages and a broader range of topics. This would not only enhance the model's versatility but also improve its ability to generalize across different linguistic and cultural contexts.

- **Refining Model Training and Prompt Design:**

Further research could explore more sophisticated fine-tuning techniques and prompt engineering. This could involve experimenting with different LLMs, ensemble methods, or even hybrid approaches that combine rule-based and machine learning techniques.

- **User Interaction and Feedback Loops:**

- Incorporating user feedback mechanisms to continuously refine the search engine's accuracy and relevance could be a valuable addition. Exploring new ways to enhance user interaction, such as more intuitive query handling or adaptive interfaces that adjust to user preferences, could further improve the user experience.

In conclusion, this project we developed a multilingual search engine capable of handling queries across multiple languages. The successes achieved and the lessons learned provide a solid foundation for future research and development, offering valuable insights for the continued advancement of multilingual information retrieval technologies.

7. Appendix

7.1 GitHub Repository:

- **Location:** The complete project code, including all scripts, data structures, and the training notebook, is available on GitHub at the following location:
 - **Repository:** GitHub (<https://github.com/50Hertz/genai-all-en-tags>)
 - **Access:** This repository is publicly accessible and includes all the files necessary to replicate or build upon the project.

7.2 Code Documentation

Project Overview:

- **genai-democracy-en-tags:** This is the root directory of the project, containing all the necessary components of the search engine.
 - **chroma:**
 - **chroma.sqlite3:** This is the SQLite database file used by Chroma to store embeddings and associated metadata.
- **data:**
 - **articles:** Contains the original articles before preprocessing.
 - **preprocessed:** Stores preprocessed versions of the articles, ready for embedding.
 - **embedded:** Files are moved to this folder after embedding.
- **output:**

- **delete**: Temporary area for outputs that are pending review or deletion.
- **test_data**: This folder contains articles in our three supported languages (German, Serbian, and English).
- **training**:
 - **AiAndDemocracy.ipynb**: A Jupyter Notebook used for training the tag-gen-llama3-model.
- **util**:
 - **library**: Houses utility scripts used across the project:
 - **util_common.py**: Contains common utility functions used throughout the project, such as text processing or data manipulation.
 - **constants.py**: Defines constant values, such as paths and configuration parameters.
- **requirements.in** and **requirements.txt**: Specify the Python dependencies required to run the project, ensuring all necessary libraries and their versions are installed.

Key Python Scripts:

- **main.py**: Contains methods to import test data articles to the ChromaDB.
- **user_inference.py**: Handles the inference pipeline, including processing user queries, and performing similarity searches to retrieve relevant articles.
- **user_preprocess.py**: Conducts preprocessing of articles, which generates tags for an article and outputs to configured JSON format.
- **test.py**: Contains test cases to validate the functionality of different components in the system, ensuring the correctness and reliability of the code.

8. Declaration

We, the undersigned, hereby declare that the technical report “Development of the All-to-En-Tags Model with a Focus on Serbian, German, and English Language Integration” has been completed as a group effort. Each member of the team has contributed to the research, analysis, and writing of this report. We affirm that this work is the result of our own efforts, except where properly referenced, and that no part of this report has been copied or plagiarized from other sources. We take full responsibility for the content of this report and confirm that it accurately reflects our research findings and conclusions.

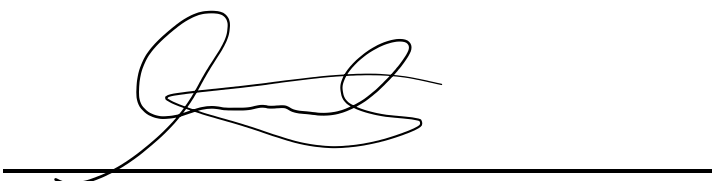
Signed,

A handwritten signature in black ink, appearing to read 'Stjepanovic', written over a horizontal line.

Saša Stjepanović

A handwritten signature in black ink, appearing to read 'Leon Kogler', written over a horizontal line.

Leon Kogler

A handwritten signature in black ink, appearing to read 'Oluseyi Stephen Adedara', written over a horizontal line.

Oluseyi Stephen Adedara

Date: August 31, 2024

References

- [1] S. Zhang *et al.*, “Instruction Tuning for Large Language Models: A Survey,” Mar. 13, 2024, *arXiv*: arXiv:2308.10792. Accessed: Aug. 30, 2024. [Online]. Available: <http://arxiv.org/abs/2308.10792>
- [2] Mixedbread AI Inc., “mxbai-embed-large-v1 - Docs,” Mixedbread. Accessed: Aug. 22, 2024. [Online]. Available: <https://www.mixedbread.ai/embeddings/mxbai-embed-large-v1>
- [3] “Chroma Docs.” Accessed: Aug. 30, 2024. [Online]. Available: <https://docs.trychroma.com/>