Exploring the Efficacy of Language Modeling and Keyboard Dynamics in the Early Detection of Neurodegenerative Diseases

Rami Baffoun, Frederik Hasenbach, Rami Kallel,
Ahmed Badis Lakrach, Ali Reda, Malte Steingass

University of Bonn,
53117 Bonn, Deutschland

Abstract. With the rising prominence of artificial intelligence (AI) and generative pretrained transformers (GPTs) in various domains, our project taps into these technologies to address a specific need in healthcare. Utilizing Llama 2, an open-source AI model known for its robust data security and analytical capabilities, we have developed a chatbot aimed at engaging patients in conversations.

The primary objective of this chatbot is to initiate patient interactions, thereby facilitating the systematic collection of conversational data. Concurrently, a keylogger rigorously records this data, with an emphasis on the extraction of insightful observations.

An agile methodology was employed to refine data collection and analysis processes collaboratively. This flexible framework allowed for continuous improvement based on iterative feedback, ensuring the project's adaptability to evolving research needs.

The collected data accurately reflects users' typing behavior, offering valuable insights into potential cognitive impairments. This outcome demonstrates the efficacy of combining chatbot interactions and keyboard dynamics analysis for data generation in medical research. Furthermore, the project's results show that integrated Large Language Models (LLMs) are capable of conducting conversations with users.

By consistently utilizing follow-up questions, users are more inclined to continue these conversations, leading to a larger volume of data.

This comprehensive dataset, upon further analysis, can provide a more precise insight into the users' neural health condition, highlighting the effectiveness of using advanced AI techniques for early detection and analysis in the healthcare sector.

1 Introduction

The advent of artificial intelligence in healthcare promises transformative changes[1], especially in the early detection and management of chronic conditions. Among these, neurodegenerative conditions stand out due to their profound impact on individuals, families, and healthcare systems worldwide[2]. Early detection of these conditions remains a critical challenge, necessitating innovative approaches that are both effective and respectful of patient privacy[3]. This project investigates the potential of language modeling and keyboard dynamics as non-invasive, AI-driven tools for the early detection of dementia. With the surge in AI applications and growing concerns over data privacy, the need for secure and ethical AI solutions has never been more critical[4] Our choice of the open-source Lama2 model

underscores our commitment to data security[5]. Adopting an agile methodology, this project emphasizes adaptability and collaboration[6], crucial for navigating the complexities of AI applications in sensitive health data environments. This approach facilitated a productive partnership with the Computer Science Department and the Uniklinikum Bonn, enhancing our project's capacity to address the intricate challenges of dementia diagnosis.
Our methodology centers on leveraging a chatbot for engaging in natural dialogues and a keylogger for assessing keyboard dynamics, which are crucial for our data collection process. This data is meticulously organized within a PostgreSQL database framework. The project's primary goal is to collect data that is crucial for analyzing cognitive decline by examining subtle variances in typing and communication behaviors, thereby addressing our fundamental research question on the viability of large language models for building tools that aid in the collection and analysis of patient data in the sector of neurodegenerative diseases.

2 Project Background

Neurodegenerative diseases are complex processes that primarily affect the central nervous system. They typically have a gradual onset between the ages of 50 and 75 and lead to significant declines in the quality of life and independence of those affected. In advanced stages, individuals rely increasingly on support from caregivers or family members.[7]

Due to the aging population and persistent shortage of caregivers, there is increasing pressure to detect neurodegenerative diseases early and reduce the burden on both individuals and society.[7]

Until 2020, therapeutic options to reverse damage were unavailable, with or without medication. Currently, only two types of medication are available, aimed at slowing down the process.[8]

Early detection of neurodegenerative diseases is currently the most effective method to assist those affected. Neuropsychological tests are suspected to distinguish between individuals undergoing healthy neural aging and those with limitations that may suggest potential symptoms of neurodegenerative disease.[9]

The aim of our project is to develop an application that facilitates early diagnosis and treatment of neurodegenerative diseases by detecting possible anomalies and making this information available to medical personnel. One advantage of this application is that it can be used without direct supervision of healthcare professionals and during everyday activities, minimizing the user's effort. Additionally, the use of communicative LLMs could further motivate individuals to generate data that could ultimately contribute to an early diagnosis.

3 Related Work

Examining research on capturing typing behavior for medical purposes provides insights and direction for the project's future. It highlights areas of focus and challenges that need to be addressed.

Collecting data for medical purposes varies, and there is currently no standard method for obtaining useful information about health status in the neurodegenerative field.[10] The topic of typing methods, such as virtual keyboards on smartphones and physical keyboards on computers, requires a detailed approach. The autocorrection on smartphones also matters (Compare [10] and [11]).

Studying metadata, such as typing errors, keystrokes, speed, and rhythm,[11] provides opportunities to understand motor issues or typing mistakes related to the user's age.[10]

The possibilities for future work include developing advanced algorithms, setting standards for early detection of neurodegenerative diseases, integrating them into the project, ongoing monitoring of symptoms, and improvements in case of treatment.[11] These mechanisms should be integrated into everyday use instead of on a separate application.[10]

These insights provide a foundation for the project's future. They explain the valuable information that can be obtained from typing behavior, the potential benefits it offers, and the reasons why the project remains significant in the field of digital medicine and neurodegenerative diseases.

3.1 Large Language Models

A Large Language Model (LLM) can be described as an algorithm designed to comprehend human language. With the help of deep learning techniques and extensive datasets, LLMs learn how to process the meaning of textual input and generate coherent responses. These models, characterized by millions of parameters, undergo expensive pre-training on massive datasets followed by fine-tuning on specialized tasks. The iterative training process includes adjustment of the parameters which optimizes the mapping of input sequences, such as textual strings, to corresponding outputs, such as responses or predictions [12].
LLMs show remarkable results across a wide range of natural language processing tasks, including language translation, text summarization, question answering, text generation, and their utilization as conversational chatbots. It is the latter application that made them popular and it is what we will utilize them for. Recent years have been shaped by ChatGPT and nowadays it is a standard tool in many individuals' daily routines. It is widely accessible and user-friendly. However it is known, that it lacks Data security [13], an issue that that has arisen in the digital age, where the widespread use of the internet lead to the increased importance of

data, making it valuable for companies [14]. While ChatGPT remains a practical tool, it does not provide sufficient data security [13], and therefore is not suitable for applications involving the handling of personal data.

4 Project Structure

4.1 Componentdiagramm

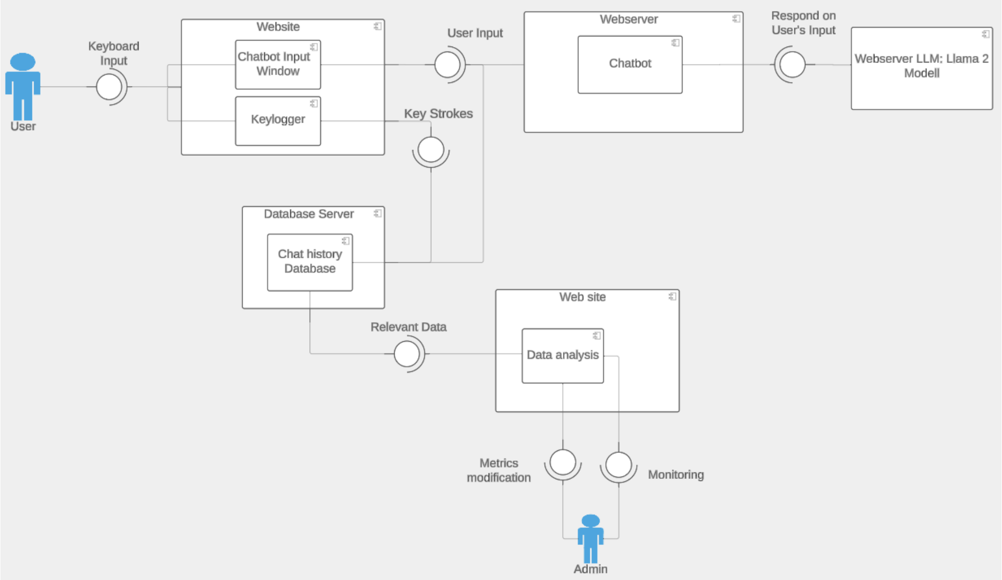


Fig. 1. Component Diagram of the System

Our system, as illustrated in 1, utilizes a chatbot interface integrated with an advanced language model, referred to as ”Lama2”, to enable user interaction via a website. A keystroke logging mechanism operates concurrently to capture typing patterns and conversational data, which may provide insights into cognitive function.

Data from these interactions are streamed to a central server hosting a database specifically designed for chat histories, where they are stored for relevance evaluation in subsequent analysis. The analytical process that follows is aimed at extracting patterns and anomalies from this data that may suggest cognitive impairment.

The role of the administrator is pivotal, with the ability to monitor the system's performance and modify the metrics of analysis. This monitoring ensures continuous system optimization and calibration to improve the quality of the collection of the data.

Our approach emphasizes the meticulous accumulation of user data, laying the groundwork for in-depth future analyses. By leveraging artificial intelligence.

4.2 Chatbot Server

Technology Stack: The chatbot server of this project is coded in Python language and makes use of its libraries. For instance, we decided on Chainlit[15] as a server for the HTTP requests. Then, to load and tokenize the large language models, in short LLMs, we used the transformers[16] library. FastAPI[17] is needed for its custom API endpoints, which we'll later explain in detail. Finally, we made use of the Python library typing[18] to modify the model's output strings. All of these modules are imported into the file models.py within the server directory.

System Prompts: System prompts are special messages used to steer the behavior of the chatbot. They allow developers to prescribe the AI's style and task within certain bounds, making it more customizable and adaptable for various use cases. In our case, the system prompt instructs the chatbot to have a friendly conversation with the user and ask them many questions about their daily lives, their feelings, and overall health. This encourages the user to interact more with the chatbot in order to collect information that can later be processed and evaluated in regard to neurodegenerative diseases.

Global Variables: Starting with the global variables in models.py, we have the boolean *models_loaded* that is false by default and becomes true once the models are loaded to ensure that it's done only once. This is called the singleton pattern because it's a global variable across all chainlit sessions. *english_pipe* and *german_pipe* are two separate pipelines that contain the chatbot's response in two different languages. The one being displayed will be chosen later by the user depending on their language preference.
This is in hindsight ensured by the boolean *german*, which displays the response in German if it's true and in English if it's false.

Functions: First, once the page is loaded, we call the Chainlit annotation **@cl.on_chat_start** which acts as the deployment function within the chatbot server. We display the starting message and pass on the system prompt in the required language. Then we try to load both models.
Loading and processing each model is ensured by the transformers library. This module provides useful functions like *AutoModelForCausalLM()* to automatically retrieve the relevant model given the name to the pretrained weights, *AutoTokenizer()* to prepare its parameters and then pass them onto the communication

pipeline, which is ensured by the function *pipeline()*. This process takes place within *load_english_model()* and *load_german_model()*, which are basically identical and differ only on the value of *model_id*.

Once the models are loaded, we set the *models_loaded* variable to true, ensuring the singleton pattern of the models and we display the welcome message in the chosen language. The chatbot then awaits the user message, sets the session variable *first_message* to true and starts saving the chatbot's conversation with the user.

Once the chatbot receives its first message, it starts the server's main function, defined by the Chainlit annotation **@cl.on_message**. This function is basically a loop that starts by reading the user's message. If it's the first one it creates the query using both the system prompt and the user's message, then it sets *first_message* to false. Otherwise, in the usual scenario, it creates the query simply by appending the new user's message to the saved log that already contains the system prompt and the complete conversation between the chatbot and the user. Then, depending on the model's language, we either call *send_query_to_german_model()* or *send_query_to_english_model()*, which do the same thing, i.e. take the query, define the minimum and maximum response length, and customize other parameters. The only difference here is that one function uses the English pipeline, and the other uses the German one. Finally, the model updates the query and sends its output, which is displayed to the user as the final response. But how exactly does the user choose the language?

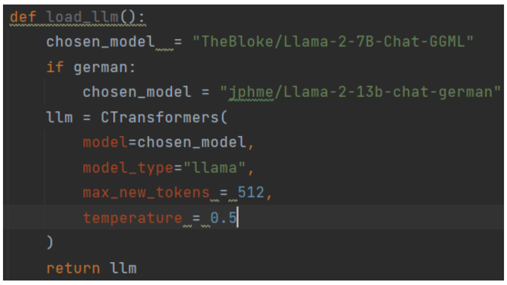
Localization: As mentioned before, we have two large language models. The English model is Llama-2-7b-chat-hf[19] by Meta and the German model is Llama-2-13b-chat-german[20] by jphme.

Both models are loaded onto their respective pipelines, but the one we send the query to is decided by the boolean *german*. What changes the value of this boolean is actually a FastAPI request, that is triggered by the user pressing the button "Swap Models".

This sends an HTTP request from the user interface to the server and directly swaps the value of *german* from false to true and vice versa. This process changes the language of the starting and welcome message as well as the system prompt and chatbot response. Although this doesn't translate an already displayed response with a different model, it remembers the user's conversation with itself even after a swap.

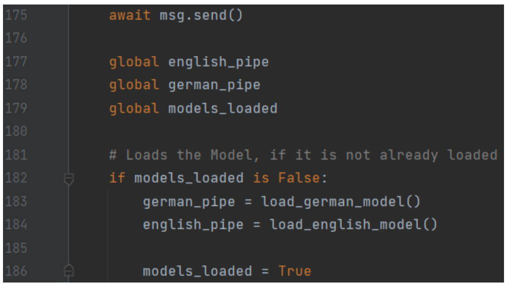
Progress: In the earlier stages of this project, we went through different ideas on how to implement certain features. Particularly with the chatbot server, there were some improvements to be made over time.

Language Change: The first idea of the model change was to decide which model to load depending on the variable *german* (Figure 2.). The benefit of this idea is less memory usage as only one model would be loaded at a time. Through trial and error, however, we quickly realized how inefficient this approach was. This meant that every time the user changes languages, they had to wait for the other model to completely load, which would take about 30 seconds for the English 7b model and around 4 minutes for the German 13b model. We decided on loading both models simultaneously on two separate pipelines and only controlling which output to display at the very end within the chainlit endpoint (Figure 3.). Even though it takes up significantly more VRAM, this combined method only managed to use up 12.8/24 GB of the RTX 4090 and it's much faster, because the models only needed to be loaded once with the server deployment. After that, the swap occurs immediately upon request.



```python
def load_llm():
    chosen_model  = "TheBloke/Llama-2-7B-Chat-GGML"
    if german:
        chosen_model = "jphme/Llama-2-13b-chat-german"
    llm = CTransformers(
        model=chosen_model,
        model_type="llama",
        max_new_tokens = 512,
        temperature = 0.5
    )
    return llm
```

Fig. 2. Load a single model at a time



```python
    await msg.send()

    global english_pipe
    global german_pipe
    global models_loaded

    # Loads the Model, if it is not already loaded
    if models_loaded is False:
        german_pipe = load_german_model()
        english_pipe = load_english_model()

        models_loaded = True
```

Fig. 3. Load both models simultaneously

Vectorstore: Faiss is a Python library for efficient similarity search and clustering of dense vectors. We used this module in the first implementation of model.py since it contains algorithms that search in sets of vectors of any size, up to ones that possibly do not fit in RAM. The downside of this idea is that we needed to have at least two separate files, one PDF file containing the data vectors in the form of a medical article to fine-tune the model's parameters on neurodegenerative diseases and a second file to load the database and retrieve its chains. This however meant more processing time and an extra file in German for the second model, so we decided on the current strategy instead.

4.3 Automatization

Technology Stack: The connection, deployment, and screening of this project are automated using Python for the scripting, git bash for sending the requests, and Linux Screen for screening the processes.

GPU Remote Connection: First we import the python library *subprocess* to open a git bash window. We then use *pyperclip* to copy the SSH credentials into the clipboard such as the server URL, password, port number, etc.

And we use *pyautogui*'s hotkey functionality to press CTRL+V and enter which passes the credentials into the git bash window, effectively sending the SSH request. Doing so not only enables us to connect to the GPU server, but also to connect to different port numbers used for the chatbot, user interface, server, and database on the localhost for testing purposes. This locally stored Python file is called login.py.

Server Deployment: To deploy the application, i.e. chainlit server as well as the custom frontend and uvicorn, we use a jupyter notebook file called deploy.ipynb stored within the server directory. This file provides functions that are used to launch or terminate each of the three aforementioned processes.
Executing the launch command starts with three separate git bash files also stored in the chatbot server directory. Each file starts by determining the right directory of the application and firing its respective command.

The chatbot deployment command is: **chainlit run models.py --port=8080**

To launch the frontend we use: **npm run dev**
And uvicorn: **uvicorn fastapi_server:app --reload --port 8082**

To terminate a process in Linux, we simply iterate over all open processes till we find its name. We get its ProcessID and we run the command: **kill -9 [processID]**

Process Screening: Screening a process enables starting a Screen session and then opening any number of windows inside that session. Processes running in Screen will continue to run when their window is not visible even after disconnecting.

In this process, we try screening all of the processes mentioned above, so that the application can still run in the background and can be accessed by any user on the internet, provided its public URL.

To start a screen with the name 'frontend' we run the Linux command:
screen -S frontend
We then start a process, similarly to what we did above. For instance, we launch the custom frontend with: **npm run dev** Finally we exit that screen (without terminating its process) using: **exit**

To actually terminate the process, however, we must know its process ID by calling **screen -ls** to get a list of all processes and their IDs. And once we get that PID we run: **screen -XS PID quit**

4.4 ChatBot UI

The chat UI of our project was mainly constructed with React.js, a widely-used JavaScript library known for its flexibility and effectiveness in creating contemporary user interfaces. React.js played a central role in our UI development, providing a strong foundation.

UI-Mockup Planning and designing user interfaces (UI) is a crucial aspect of software development. The UI is the first thing the user sees and determines whether they want to continue engaging with the application. Using UI mockups during the early stages of development allows for design decisions to be visualised and evaluated before implementation begins.

In this situation, the aim was to create a UI specifically tailored to potential user demographics. Individuals between the ages of 65 and 85 are over 15 times more likely to develop Alzheimer's disease than younger people.[21] Therefore, it is probable that this age group will utilise this website. As a result, the design should be adjusted to address potential issues faced by this demographic.

Age-appropriate UI should consider the following points in its design: clear, sans-serif fonts and font sizes that are as large as possible. This fragment of text discusses the importance of readability and contrast for older readers. It is recommended to use high contrast between the text and background to aid recognition, as older people may struggle with low contrast and similar shades. Additionally, it is important to be mindful of the use of certain colours, such as blue, violet, green, and yellow, as perception can shift towards yellow with age and the absorption of light waves is restricted. (See Fig. 4 and 5) The placement of essential web objects and content should be central to make them easier to find. Patterned backgrounds should be avoided. By following these principles, a UI can be made more accessible to older people by considering their age-related visual limitations and improving usability.[22] Following these principles, the final UI mockup appears as shown. The design emphasises the contrast between white and dark grey, with the most significant elements of each page positioned centrally to draw attention to them. Care was taken to ensure that the chat window layout is consistent with other chat windows in similar software. The message input is located at the bottom of the window, and the text bubbles are differentiated by color and position based on the sender. (See Fig. 6)

Unfortunately, when moving the UI mockup to a working HTML page with CSS, not all objects could be moved exactly, so there was a difference between the current product and the original plan. As a result, our design expectations were not fully met. Although the characteristics of age-appropriate use were still present, the criticism of modernity was the decisive factor in our decision to reject

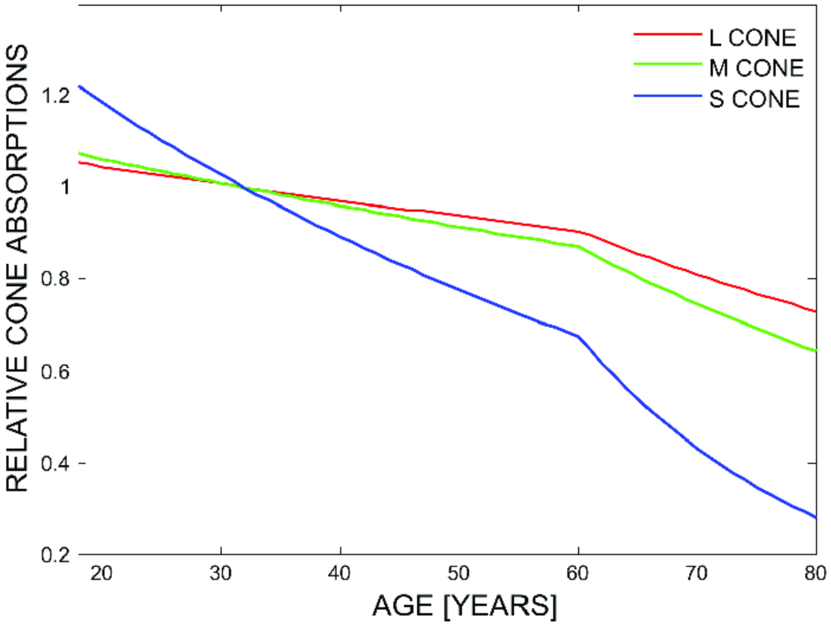


Fig. 4. Decrease of Wavelength Vision in Years[23]

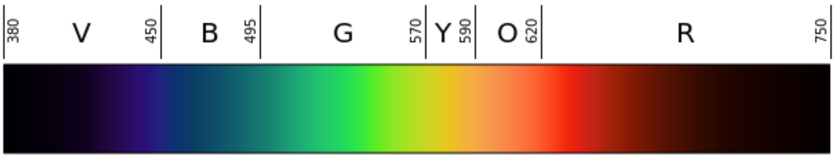
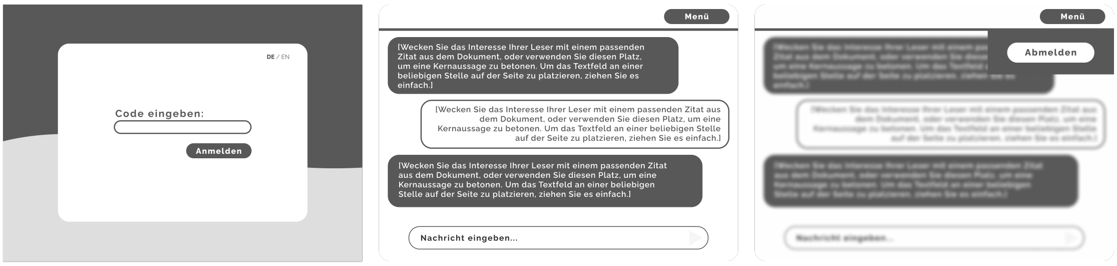


Fig. 5. Linear Visible Spectrum[24]



[Wecken Sie das Interesse Ihrer Leser mit einem passenden Zitat aus dem Dokument, oder verwenden Sie diesen Platz, um eine Kernaussage zu betonen. Um das Textfeld an einer beliebigen Stelle auf der Seite zu platzieren, ziehen Sie es einfach.]

[Wecken Sie das Interesse Ihrer Leser mit einem passenden Zitat aus dem Dokument, oder verwenden Sie diesen Platz, um eine Kernaussage zu betonen. Um das Textfeld an einer beliebigen Stelle auf der Seite zu platzieren, ziehen Sie es einfach.]

[Wecken Sie das Interesse Ihrer Leser mit einem passenden Zitat aus dem Dokument, oder verwenden Sie diesen Platz, um eine Kernaussage zu betonen. Um das Textfeld an einer beliebigen Stelle auf der Seite zu platzieren, ziehen Sie es einfach.]

Nachricht eingeben...

Fig. 6. UI-Mockup

the UI mockup during the project. The decision was made to choose an existing design and adapt it to our requirements.

Careful planning and execution of a UI mockup is a crucial step in software development. This process helps to identify the necessary considerations for programming and implementation of UI functions, as well as highlighting important aspects of usability and accessibility. At the same time, the deviation from our UI mockup demonstrates the importance of an agile approach to software development. This approach may result in work that does not always contribute to the finished product.

Technology Stack React.js: React.js[25] served as the base of our UI development, providing a framework for building user interfaces. Utilizing React's component-based architecture, we created reusable UI components, facilitating easier maintenance and scalability of the application. React's virtual DOM also optimized rendering performance, ensuring a responsive user experience. By utilizing React.js, our team implemented a wide range of features and functionalities that enriched the user experience within the project's environment. From dynamic content rendering to real-time updates, React.js provided the tools necessary to facilitate seamless user interactions and fluid navigation throughout the application.

Material-UI: Material-UI[26], a React UI library inspired by Google's Material Design principles, significantly contributed to enhancing the user experience of our project. It was easily integrated into our project through a straightforward installation process. By adding Material-UI as a dependency, we obtained access to a comprehensive suite of pre-designed components and styles, which expedited the UI development process. These components included various UI elements such as buttons, forms, and avatars (Figure 7.).

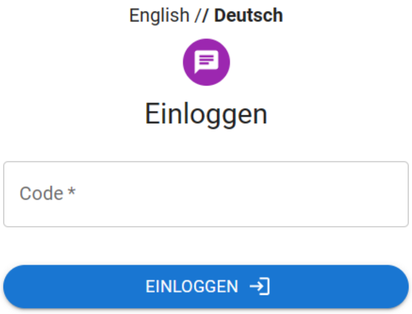


Fig. 7. login form with material UI pre-designed componenets

Design and User Experience: Design and user experience are crucial elements in the success of any application. Our project underwent a prototyping phase, which involved creating an initial iteration of the user interface design based on previously discussed UI mockups. This preliminary design was refined over time through systematic research and feedback from team members. This iterative process allowed us to identify areas for improvement, leading to design modifications aimed at enhancing usability and optimizing the user experience.

The final design decision was made with careful consideration of the project's nature and objectives. We selected a user interface that seamlessly aligned with our intended goals and target audience. Our focus was on crafting an intuitive and user-friendly interface, with a priority on simplicity and ease of use to enhance overall usability.

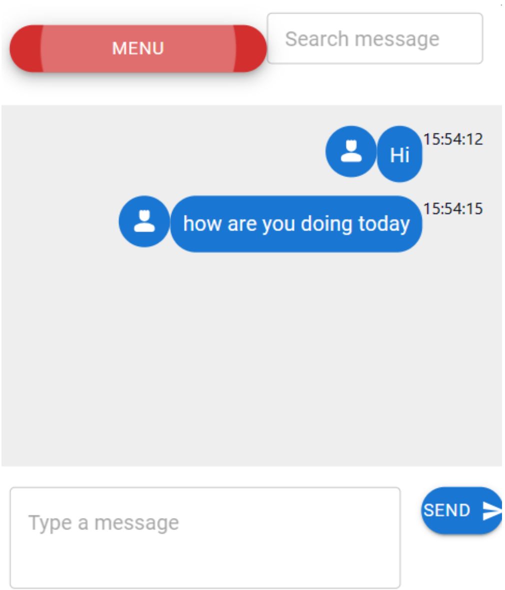
Implementation: After establishing our design principles and user experience objectives, our team proceeded to the implementation phase, aiming to translate these concepts into practical technical solutions. We selected React packages that matched our design vision while emphasizing smooth functionality and user-friendly navigation within the project interface.

To achieve this, we integrated React Router [27], a routing library tailored for React applications. This integration allowed us to define dynamic routes and manage navigation effectively, enabling users to easily transition between different pages and sections of the chatbot. The deliberate integration of React Router underscored our commitment to ensuring a smooth user journey within the chatbot interface, ultimately enhancing overall user satisfaction and engagement.

Language Support for Enhanced Accessibility: To address the diverse needs of our user base and promote inclusivity, our project was designed to accommodate both English and German languages within the chatbot interface. This implementation aimed to facilitate seamless language switching, thereby improving accessibility and usability for a broader audience.

The feature enabling language switching was achieved through the integration of the react-i18next package[28]. This package facilitated the translation of various interface elements, including placeholders, buttons, and textual content.

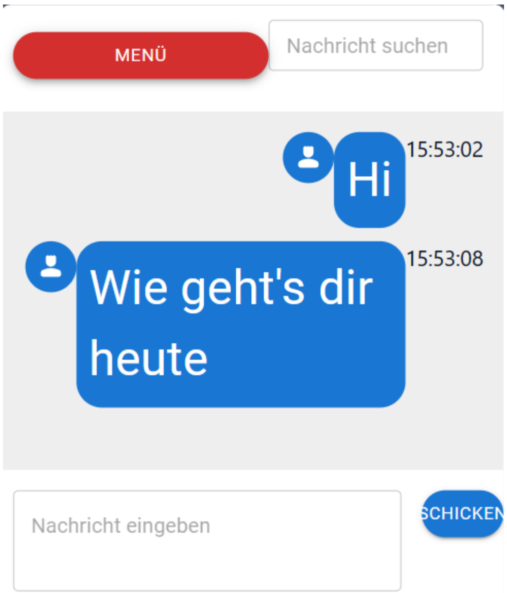
Enhancing Accessibility with Text Size Adjustment: In order to further enhance accessibility and accommodate users with varying visual preferences, our chatbot interface includes an Accessibility Mode option. This feature, represented by a ”Text Size” button, enables users to adjust the size of messages displayed within the chat interface. This button serves as a user-friendly mechanism for enlarging text, specifically catering to the needs of older or visually impaired individuals (Figures 8 and 9.). By providing the option to increase text size, our aim is to improve readability and enhance the overall user experience, ensuring that all users, regardless of age or visual acuity, can comfortably engage with the chatbot interface.



Hi 15:54:12

how are you doing today 15:54:15

Fig. 8. Accessibility Mode On



Hi 15:53:02

Wie geht's dir heute 15:53:08

Fig. 9. Accessibility Mode Off

Responsive Web Design: As previously discussed, our User Interface design was anchored on the utilization of the Material-UI (MUI) library. Material-UI offers a robust suite of pre-designed components engineered to prioritize responsiveness. This strategic choice ensured that our chatbot interface delivers consistent and optimized user experiences across various devices and screen sizes. Each Material-UI component is meticulously crafted to dynamically adapt its layout and presentation in response to the available screen real estate. This inherent adaptability eliminated the need for manual intervention or the implementation of media queries to ensure responsiveness.

Rami Baffoun

In our project, we primarily leveraged a combination of <Box>and <Grid>from the Material-UI library. These components played a key role in creating a responsive login page (Figures 10 and 11.) and a responsive chat page (Figures 12 and 13.).

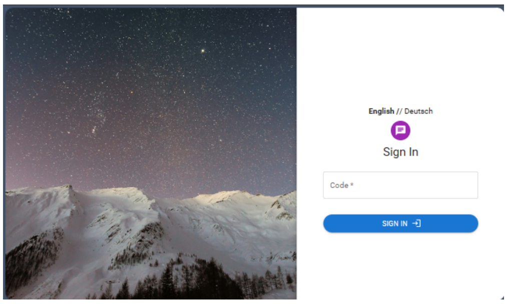


Fig. 10. Login page for larger screens

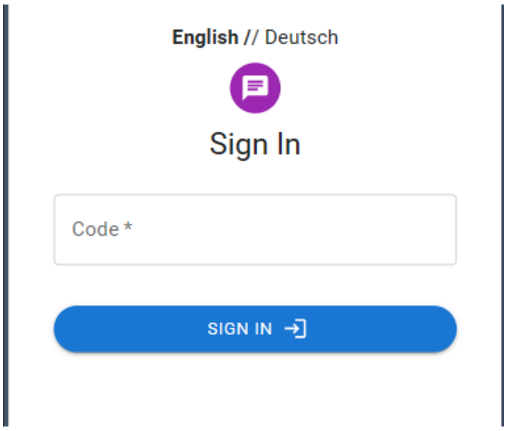


Fig. 11. Login page for smaller screens

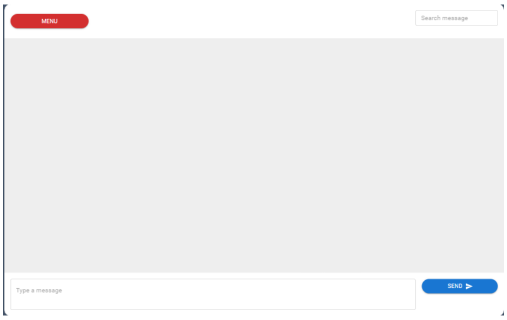


Fig. 12. Chat page for larger screens

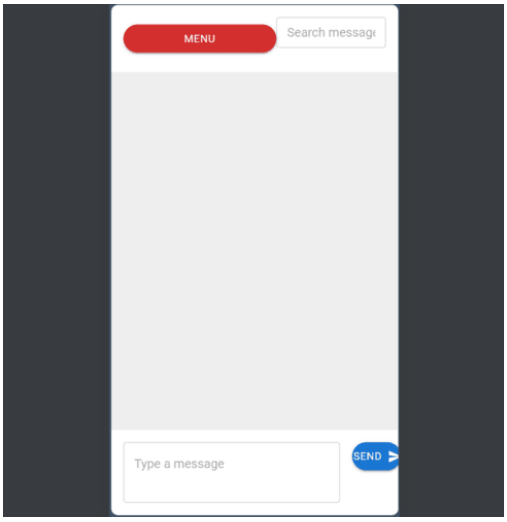


Fig. 13. Chat page for smaller screens

Integration with the Backend: In establishing a seamless connection between the chat UI and the server infrastructure, managed by ChainLit and FastAPI, we adopted a comprehensive integration approach. Our method involved utilizing the ChainLit React package[29], specifically designed to establish a WebSocket connection, facilitating seamless communication between users and the underlying models.

Upon users' login and access to the chat page, we employed the "useEffect()" hook, one of the frequently used React hooks. It was utilized to create a connection with the ChainLit server, which helped seamlessly integrate users into the chat environment, enabling a smooth exchange of messages within the chatbot interface.

Custom API Endpoints for Authentication: To ensure secure user authentication, we created custom API endpoints within FastAPI. The endpoints were customized to authenticate users via JSON Web Tokens (JWT)[30], utilizing HTTP calls to FastAPI for validating codes entered by the user against our database. A user is granted access and assigned a Token only if the entered code is confirmed to exist and valid.

Facilitating Message Exchange with Custom React Hooks: ChainLit's skilled team provided a suite of custom React hooks, including "useChatInteract," which defines methods to interact with the chat, such as sending messages, replying, and updating settings, and "useChatMessages," which provides access to the chat messages and the first user message. These thoughtfully crafted hooks were developed to optimize the flow of messages between users and our models, guaranteeing smooth communication and enhancing user engagement within the chatbot interface. Our project employed these hooks to enable seamless messaging between users and the models.

Dynamic Model Swapping Endpoint `/toggle_variable`, an additional custom endpoint was skillfully designed to manage the authorization process for switching between the German and English language models within the backend infrastructure. This endpoint facilitated the seamless exchange of variables between the chat UI and the server, enabling it to dynamically switch between the two loaded models based on user preferences. The intricate process of how the backend utilizes the incoming variable from the `/toggle_variable` endpoint to enable dynamic language switching within the chatbot interface was extensively explained in our dedicated backend section.

4.5 Large Language Models

Memory Requirements A significant hurdle for the use of LLMs is their sheer size. They are built on vast neural networks with billions of parameters and therefore require an enormous amount of memory and computational power. This poses a challenge for the everyday user, because these requirements are beyond the capabilities of the standard everyday user device. For example, the smallest 7B model of the Llama 2 series requires for inference around 28 GB of VRAM, if loaded in 32-bit precision. A really expensive GPU would be needed in order to use this model. Luckily, there is a method called quantization that minimizes the necessary resource requirements without losing a lot of the model's performance.
First let us understand how the memory requirements come together. The size of the model can be estimated by multiplying the number of parameters with the precision size of the parameters [31]. For a model with 7 billion parameters, where each parameter takes up 4 bytes (32-bit), the needed memory amounts to 7 billion $\cdot$ 4 bytes = 28 GB. If each parameter only took up 2 bytes, the needed memory

would be halved and amount to 14 GB. For 1 byte (8 bit) per parameter, it would be quartered and for 0.5 byte (4 bit) it would be one-eighth of the original memory requirement. This is just an approximate calculation, because in reality there is not a one-on-one correspondence between parameters and model size, but it gives a good estimation. It has to be noted that these calculations only apply to inference. For training, the required memory is much higher [32].

The downsizing of the model's original precision to something smaller is known as quantization. Quantization aims to represent the model's weights with a smaller precision, without losing a lot of the model's performance [33]. As shown, by using quantization techniques, the necessary memory requirements can be reduced by a multiple. This is a huge improvement, because it enables the use of LLMs for normal users with consumer hardware.

For example we used the Llama 2 13B model from "Hugging Face" [34] for inference on a GPU with 24 GB of VRAM. The original weights of the model are stored in 16-bit precision. Therefore the model needs around 13 billion · 2 bytes = 26 GB. This would not fit on our more than decent GPU. In theory, with 4-bit quantization, we could reduce the size of the model to around 13 billion · 0.5 bytes = 6.5 GB. In praxis, the model took up around 8 GB of VRAM. This number is a little higher than in theory due to some other necessary components like tokenizers. 8 GB is a very acceptable size and fits easily into our and even smaller consumer GPUs. If 8 GB is still too big, the smaller Llama 2 7B can be used. On our GPU it occupied around 4.6 GB of space, only 1.1 GB more than the 3.5 GB it should need in theory.

For quantization we used the Bitsandbytes library [35]. It performs all the necessary steps to load the model with a smaller precision. It is integrated into the Transformers library and is easy to use. Alternatively, there are some other quantization methods, like GPTQ, AWQ, GGUF, and GGML, which can be more performant and generate responses a little faster, depending on the use case. Diving into these techniques gets very technical and requires a lot of hardware knowledge. For our model, the quantization with Bitsandbytes gave really good results and it is not likely that these other methods would add a noticeable value, but could be tried in the future.

Conversational Memory In our exploration of Large Language Models, we initially encountered a surprising realization. The responses of the model seemed to be independent of the rest of the conversation, as if the model was only capable of answering the immediate question without retaining any knowledge of previous interactions. We realized that the model itself has no inherent memory. This was not expected, because the general use of ChatGPT gave the impression that LLMs had something like a memory. All of us have used ChatGPT before, but we never questioned how the model is able to remember previous messages. It just never

came to mind that this is something special, because it is such a basic requirement for having a natural conversation. So the model itself does not initially have this ability, but there had to be a way to make it possible. It turns out that LLMs are stateless, which means that every input will be processed independently from any previous input. This is due to the underlying transformers structure of the models. It does not contain any memory-keeping components, only constant weights. The key to unlocking conversational memory is the prompt that is fed to the model. In order for LLMs to know the previous conversation, the prompt has to not only include the new message of the user, but also the whole previous conversation. It will take all this input, process it and generate an answer based on all this information [36]. The difference can be shown with a simple example.
The conversation without providing the whole context in every prompt:

```
User:    Hi, ich bin Malte.
Chatbot: Hallo Malte...
User:    Weißt du wie ich heiße?
Chatbot: Oh, entschuldigung! Ich weiß nicht, wer Sie sind oder
         wie Sie heißen.
```

The conversation with providing the whole context in every prompt:

```
User:    Hi, ich bin Malte.
Chatbot: Hallo Malte...
User:    Weißt du wie ich heiße?
Chatbot: Ja, ich weiß, dass du Malte heißt. Du hast es in
         deiner ersten Nachricht erwähnt.
```

Depending on the model, there are different formatting styles. They depend on how the model was trained and are necessary to achieve the best results. For Llama 2 the recommended style is the following [36]:

```
<s>[INST] <<SYS>>
{your_system_message}
<</SYS>>

{user_message1} [/INST]
```

To append the answers of the model and continue a conversation it should look like this:

```
<s>[INST] <<SYS>>
{your_system_message}
<</SYS>>
```

```
{user_message1} [/INST] {model_reply}</s><s>[INST] {user_message2}
[/INST]
```

This approach enables the model to remember previous fragments from the conversation, but there are a couple of disadvantages that have to be kept in mind. If the whole message history is added to the new message from the user, the prompt of the model gets longer as the conversation continues. This can cause a couple of problems.
First of all, the model has to process more and more text, which leads to longer processing times. Also, the amount of tokens a model can process is limited. This amount is called the context window of a model. If the prompt is bigger than the context window, the model will truncate the first tokens of the input and only consider the rest that fits within its context window. This cuts off the system prompt, which is always at the beginning of the prompt, and then the oldest parts of the conversation, making the model forget the system prompt and these messages. There are a couple of approaches to handle this problem [37].
One of them is the idea to edit the prompt, so that only the oldest messages will be truncated when the context window is reached. This ensures that the system prompt, which is essential for the model's behavior, will remain and only the least relevant messages will be forgotten. Another approach is to not use the conversation as a whole, but to use a summarization of it. This way, the model still has access to the most relevant information, while the context window will not likely be reached [37].
Another known problem is that the model tends to forget the system prompt after a couple of messages. This might be because the system prompt is at the beginning of the prompt. As the prompt gets longer, the model has difficulties processing all the information and therefore does not follow the instructions of the system prompt anymore. One solution provided by the authors of the model, is to add a little reminder of what the model is supposed to do, to every new message. This method is called Ghost Attention [38].
For our use case it is not likely that the conversation gets long enough to exceed the context window. Llama 2 can handle up to 4096 tokens, which is estimated at around 3000 words [37]. This is quite a lot for a simple conversation. For that reason, we have not implemented these solutions. However, if needed, it can still be implemented later with a reasonable effort.

Process for choosing Llama 2 Llama 2 is a LLM from Meta. It was created in cooperation with Microsoft, with the aim to make it available for everyone. The model is free for research and commercial use, which makes it an amazing opportunity for businesses, entrepreneurs, and researchers who do not have the resources to build this technology by themselves [39]. It can be downloaded and run locally.

The goal of our project is to collect personal data about patients and therefore providing data security is a basic prerequisite we have to fulfill. Beeing able to run the model locally is what ensures that our institute has full control over the data of the users. This way, the data protection only depends on the security measures of our institute and not on any uncontrollable third parties.
The model currently is available with parameter sizes of 7B, 13B, and 70B (B stands for billions). A 30B model is also planned. There are two variants of the model available. A base model and a fine-tuned model for chat use cases[40]. Since we want to build a chatbot, we use the latter. Created by one of the top players in the LLM space, Llama 2 seems to be the most promising open-source model available. It delivers amazing results compared to its for consumer hardware manageable model sizes [41]. In comparison, ChatGPT 3 is with 175 billion parameters around 13 times bigger than the 13B model and 25 times bigger than the 7B model of the Llama 2 series. What stands out is the model's roleplaying ability. It does a really good job in performing the commands that are given to it, which makes it easy to adjust to the needed behavior [42] Selecting the right model was a long process. Since we were new to the field, we had to gather a lot of information on how to effectively utilize these models before we could start testing them. Issues such as hardware requirements, prompting techniques, proper model downloading procedures, and selecting the appropriate quantization method presented considerable challenges. Every time we learned something new, more questions emerged. Our primary criterion for the chosen model was its proficiency in both English and German, along with compatibility with our GPU with 24 GB of memory. We discovered that only small models, like 7B, 13B, and maybe 30B models could fit. In terms of speaking capabilities, we needed a model that could hold a natural and engaging conversation. The Llama 2 model looked the most promising, because it is available in manageable sizes, is free to use, looks like it could speak German, and is praised as one of the best open-source LLMs. When we first tested the 7B model in English, we quickly realized its impressive speaking abilities. The models generated responses were remarkable, leaving little room for complaint. What stood out the most was its adeptness at following instructions and its strong role-playing capabilities. For our project, these are very helpful features, because they allow us to influence the model's behavior to suit our needs. Also, the time to answer was in a normal time span. This is very important, because if it takes too long to answer, the user will be annoyed and stop the conversation. It is an advantage of a small model. With larger models, the process of generating an answer is more complex and can lead to longer response times.

Problem with languages The Llama 2 13B model performed well during testing in English, demonstrating the necessary abilities for our project. However, when tested in German, the model performed poorly, and it is recommended that further testing is carried out to improve the model's German skills. The model frequently switched between German and English, even mid-sentence, and the quality of its German responses was subpar despite understanding the input. During the research process, it initially appeared that the model had the ability to speak up to 20 languages, including German. When researching further it became clear that this was not the case with Llama 2. The model was trained on a significantly larger amount of English data (89.7%) compared to German data (0.17%). This explains why the model's German-speaking capabilities are insufficient. The model was trained on enough German data to understand the language, but it was not enough for proper speaking skills [38].

There are a couple approaches to handle this problem [43]:

1. Try to get the model to reply in the specific language with prompt engineering
2. Hope that the authors or people of the community fine-tune the model for that language and release it
3. Fine-tune the model by yourself

From the beginning, we attempted the first approach of using the system prompt to instruct the model to respond only in German. Without this prompt, the model would respond in English, even if the user's message was in German. We attempted system prompts in both languages. The English instructions yielded slightly better results, but as mentioned earlier, the model was not trained with the necessary prerequisites to speak proper German, resulting in inadequate answers.

Another option is to rely on a pre-trained model that has been fine-tuned on German data. We discovered a suitable Llama 2 13B model on Hugging Face that meets this requirement[20]. The German answers are of lower quality than the English answers in the original model. There are occasional errors in individual word choice and sentence structure, but the overall meaning is always understandable and accurate. For instance, consider the following conversation where one sentence is formulated awkwardly:

```
User:  Hallo, wer bist du?
Chatbot: Hallo! Ich bin ein KI-Sprachmodell namens LLaMA,
         und ich bin hier, um dir bei deinen Fragen zu helfen und
         dich auf deiner Reise durch das Leben zu unterstützen.
         Wie geht es dir heute? Hast du eine gute Woche gehabt
         oder gibt es etwas, worüber du nachdenken möchtest?
User:  Mir gehts gut. Ich hab eigentlich nichts zu erzählen
Chatbot: Gut zu hören, dass dir alles gut geht! Wenn du
         irgendwelche Gedanken oder Gefühle hast, die du teilen
         möchtest, stehe ich dir gerne zur Verfügung. Es ist
         wichtig, sich mit anderen über seine Emotionen und
```

```
Erfahrungen auszutauschen, da dies unsere Beziehungen
stärkt und unsere persönliche Entwicklung fördert.
```

Although the sentence "Gut zu hören, dass dir alles gut geht!" is not grammatically correct in German, its meaning is clear. Despite occasional answers like this, we believe that the model's overall speaking ability is impressive given its small size. We found only one other model that is promoted having great German-speaking capabilities [44]. The problem was that we could not get it to run properly. The answers were only generating nonsense. However, it looks promising and it seemed like a temporary problem that will be fixed in the future, so it still might be a place to keep in mind.

The fine-tuned model we found has sufficient speaking capabilities and other than that we found no other promising German models that were open source and that met our hardware requirements. Fine-tuning seemed to be too expensive and since the speaking abilities were already good, it was unnecessary.

System Prompts System prompts are essential for guiding the behavior of language models such as Llama 2. Essentially, a system prompt is a piece of text that is added at the beginning of each input given to the model. This helps direct the model's responses according to specific contexts or roles. For example, in order to develop a chatbot that always behaves in a certain way, it would be necessary to remind the model to adapt the specific behavior in every prompt. The system prompt can be used to establish the desired behavior without the need for tedious repetition. [37]. They can provide the model with instructions regarding desired response patterns and customize the model's outputs to suit various scenarios. This is helpful, because it makes the model flexible and allows for modification of the model's behavior to fit specific use cases, without expensive fine-tuning [37].

Llama 2 excels at interpreting and respecting system prompts, ensuring a consistent and appropriate dialogue experience. This ability is a result of training that intentionally focused on initial context-giving prompts. To encourage the user to generate more data, the model should engage in interesting conversations without writing lengthy responses. Asking questions is an effective way to keep the conversation flowing. Additionally, inquiring about the user's life and well-being can be beneficial. Various system prompts were tested to guide the model's behavior towards the desired outcome.

Discovering the optimal system prompt cannot be achieved through online research, but rather trough trial and error. However, evaluating various conversations and determining the most effective one is challenging.There is no measure for rating conversations, and comparing responses is not feasible due to the model's tendency to generate different replies, resulting in entirely distinct conversations. Therefore, the first answers are the only ones that can be compared, but they do not represent the quality of the whole model and therefore are not viable for a reliable evaluation. Another problem is that the evaluation of the interest level of a conversation is highly subjective. Knowing these problems, we based our evaluation of each system prompt on our subjective perception of the model's behavior after having several conversations with it. We analysed the frequency of the model's questioning, the length of its responses, and its inquiries into the user's personal life and well-being.

Initially, we tested the system prompt: "Du bist ein Chatbot auf einer Webseite. Dein Ziel ist es, ein interessantes Gespräch mit dem User zu führen, dass den User dazu bringt viel zu schreiben. Dementsprechend solltest du versuchen viele Fragen zu stellen." The model asked a few questions at the start, but stopped doing so after a few iterations. Overall, it appeared to have little effect. According to an article, English prompts may be more effective due to the system prompt being solely trained in English [43]. Therefore, we have begun utilising an English prompt.

For the second one we used the default prompt that was used in training: "You are a helpful, respectful and honest assistant. Always answer as helpfully as possible, while being safe. Your answers should not include any harmful, unethical, racist, sexist, toxic, dangerous, or illegal content. Please ensure that your responses are socially unbiased and positive in nature. If a question does not make any sense, or is not factually coherent, explain why instead of answering something not correct. If you don't know the answer to a question, please don't share false information.“ The responses provided were unremarkable and lacked depth. The model assisted as requested, but did not generate any interesting content.

For the next test we added the following to the default system prompt: "Your goal is to engage the user to write a lot. Write interesting things and ask a lot of questions. Ask a question, every time you give an answer. In the beginning, ask the user about his well-being. Be interested in the user's health and ask questions about it, if he talks about it.“ This led to noticeable improvements, with each response incorporating at least one question. However, the model occasionally fixated on specific topics, delaying transitions to new subjects. Additionally, the model initiated the conversation with small talk, inquiring about the user's well-being. This feature could be beneficial in situations where a doctor requires daily updates about their patient's well-being. The model asked follow-up questions when the user mentioned that he is not feeling well and stopped asking about it when the user reported otherwise.

The previous system prompt's results were satisfactory, and there is little room for improvement. However, we identified two areas that require attention. Firstly, we discovered that the default system prompt, which was used for training and is part of the model, is unnecessary and does not contribute to its performance. Therefore, we have removed it to reduce the model's processing load. Secondly,

we noticed that the model's responses can be excessively lengthy at times. The enumerations can feel excessively long as the model lists up to 10 lengthy options. To address this issue, we included the prompt: "Your answers should not be too long". However, this did not fully resolve the problem as the model may still need to provide lengthy answers in certain situations. One possible reason for this limitation is that the model may not have been fine-tuned on sufficient data for certain scenarios, resulting in lengthy responses.
In summary, system prompts can influence the model's behavior to a certain extent, but they are constrained by the model's inherent capabilities. The effectiveness of prompts may also vary based on the model's size, because larger models have better capabilities. This suggests that larger models could potentially enhance the influence that system prompts have in controlling the model.

4.6 Database

Integrating databases into the chatbot is crucial for collecting and tracking data about the user. These databases serve as repositories for storing and organizing a wide range of relevant information, including user profiles, conversation histories, message content, and keystroke data. By leveraging this structured database architecture, we can access to a wealth of contextual information, which facilitates the generation of potential future quantitative metrics and analyses. On top of that, databases enable us to access specific data by utilizing specific commands and queries tailored to our requirements. In other words, the use of databases within the chatbot framework allows for efficient data collection and management. On the other hand, the databases can also be useful for the login functionality, in case we want the chatbot to have many users with each user having their own login data. In this context, PostgreSQL is a great choice since it is an OpenSource that provides support for writing database functions using SQL and Python along with many useful features, such as the ability to attach comments on tables, databases, columns, and other individual database objects which would allow us to document decisions or implementation details.

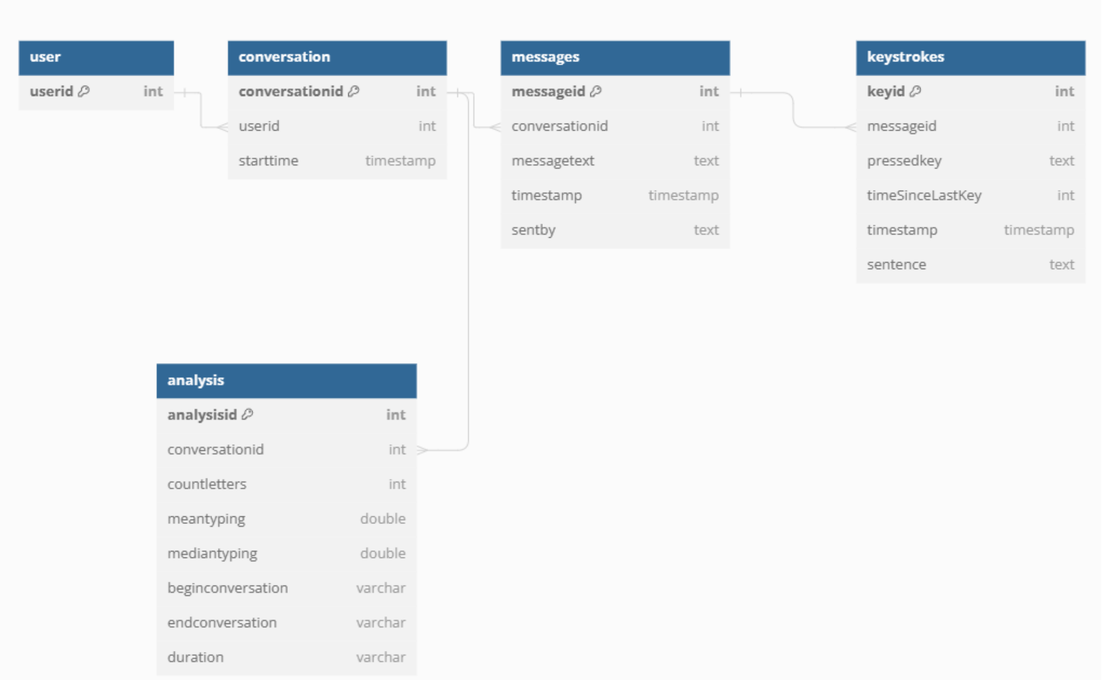


Fig. 14. UML Diagram of the databases

Figure 1 illustrates the database schema designed to support the chatbot. This structured layout encompasses several key tables, including ”user,” ”conversation,” ”messages,” ”keystrokes,” and ”analysis,” each playing a vital role in capturing and analyzing user interactions. These components are interconnected through defined relationships, ensuring efficient data retrieval and analysis for cognitive assessment purposes. The ”user” table serves as a foundational element, enabling the identification and tracking of individual users participating in conversations. It is composed of only one element, which is the User-ID, and we made sure that for each instance, the generated ID doesn’t already exist, which allows the data of the users not to get mixed up. The ”user” table is linked to the ”conversation” table, which records conversation details such as start time and user ID, it facilitates the organization and retrieval of contextual information, such as the starting time of the conversation. The ”messages” table stores the messages of the user during conversations, crucial for analyzing communication patterns, where each message gets saved instantly, along with the time at which it got sent. The current ”messages” table saves only the messages sent by the user, but its functionality can be extended to additionally store the messages generated by the chatbot. This could be used to analyze the behavior of the user through his reactions on the diverse messages of the chatbot. The reason why we have not merged the ”conversation” and the ”messages” tables into one table, is that one user can engage with the

chatbot in different conversations at different times, and it is important not to mix up the messages of the separate conversations. Additionally, the "keystrokes" table captures typing behavior, specifically to determine the time interval between the input of two consecutive characters. Finally, the "analysis" table consolidates derived metrics, calculating different values like the typing speed, based on the content of the "keystrokes" table like letter count and typing speed. The relationships established through references ensure data coherence and integrity, enabling seamless interactions between database components for efficient analysis.
However, it is important to note that we can always add/modify tables based on the type of data we want to capture, which reflects the flexibility of the databases. Overall we can say that the chatbot is simply a tool to make the user behave and to give us a chance to capture his behavior's characteristics, and that the databases are a great tool to store the data, since it provides diverse useful functionalities that help us to save and sort up the data according to our wishes.

4.7 Keylogger

A keylogger, fundamentally, is software that meticulously records keystrokes made on a keyboard. While traditionally associated with cybersecurity concerns, in the context of our project, it assumes a pivotal role in collecting behavioral data. Specifically, our project employs a keylogger designed not merely to log keystrokes but to analyze the temporal dynamics between them—measuring the latency before the initiation of the next keystroke.[10]

This approach of data collection is instrumental for our investigation into the early markers of neurodegenerative diseases. Research indicates that cognitive impairments can subtly influence motor functions, including typing behavior.[10]

n our student-led project, the incorporation of keylogging serves not just as a data collection tool but as a gateway to understanding the intersection of cognitive function and digital interaction.

5 Analysis

The data collected from the keylogger provides limited insights initially. Therefore, it is essential to bring this data together for later analysis and integration into the context of medical research. The flexibility in choosing summarization methods is intended to be improved in the future.

Currently, we are focusing on specific datasets, including the assignment of conversations to respective users for comparison. (See Fig. 14) This method allows for the identification of changes during interactions. Furthermore, it records the mean and median time between two letters during a conversation to gain insight into users' motor skills. Only values within a defined range are considered to prevent distortion caused by intentional fast typing or pauses.[10]

To frame these values, the number of typed keys per conversation, the start and end times of the conversation, and the resulting duration are captured. These data enable comparisons, such as whether the average time between keystrokes depends on the conversation duration, time of day, or textual length.

It is important to note that no statements can be made about how these values support the diagnosis process. As previously noted (See Related Work) in studies, there are no existing standards for evaluating such properties.

This analysis demonstrates the capability of our project to gain additional insights from the metadata of keystrokes promoted by the chatbot. These insights are intended to be adapted in future digital medical research.

6 Limitations

The limitations of our project were notably influenced by several constraints. Primarily, our access to the GPU server was not as flexible as required, which significantly restricted our ability to experiment extensively with the Llama2 model. This limitation hampered our capacity to conduct thorough testing and exploration of the model's full capabilities, ultimately confining the scope of our AI-driven solutions. Additionally, the restricted server access prevented us from evaluating a wider range of models. As a result, our selection of the Llama2 model was primarily based on existing studies rather and security reasons than direct comparative analysis, which could have potentially identified a more optimal or suitable model for our project's specific needs.

Moreover, our relative inexperience with prompt engineering posed another challenge. Effective prompt engineering is critical for leveraging the full potential of large language models, and our amateur proficiency in this area may have affected the efficiency and accuracy of the data generated and collected[0]. These limitations underscore the need for enhanced resources and expertise in future projects to fully explore and utilize the capabilities of AI in healthcare research.

7 Future Work

To improve the experience of the user when writing with the model, additional open-source models could be tried out. The area of LLMs is consistently evolving, which means that newer and better models are likely to be released in the future. This is particularly true for German-speaking models, which are currently limited in availability. Meta is currently developing a 30B Llama 2 model, which may be big enough to have improved German-speaking capabilities, making it an interesting option for testing. The larger size of the model may improve the quality of the answers and increase the influence of system prompts on the model's behavior, potentially enhancing its general usage and expanding its field of application. The model requires a GPU with 24 GB of VRAM.

The EM German model family shows potential and justifies further investigation as a viable option. [44].

8 Conclusion

In summary, LLMs offer good capabilities to utilize for passive data generation and collection from users. Our tests demonstrate that users can engage in comprehensive conversations with a small and free-of-charge model. While the capabilites of LLMs are already impressive, there is still room for improvement, particularly with open-source German-speaking models. The extent to which they motivate users to write more and thus generate more data also depends on the use case. The implementation of a survey-like application alongside conversations may enhance the quality of the generated data. It would be appropriate for applications such as a survey chatbot, designed to engage individuals in brief conversations. However, to consistently motivate users to daily engagement might be challenging overtime, due to a missing interest in the conversation. To counteract this effect, implementing the chatbot as a journal application could be used to obtain insights into the user's status and still keep the engagement on a higher level. Ideas such as this should be implementable, particularly with the continuous improvements in LLM technology.

Our project provides a solid foundation for further research in this area. The adaptability of LLMs allows for easy modification of behavior by changing the system prompt, making our prototype versatile for various scenarios.

References

1. Vivek Kaul, Sarah Enslin, and Seth A Gross. History of artificial intelligence in medicine. *Gastrointestinal endoscopy*, 92(4):807–812, 2020.
2. Innocenzo Rainero, Amalia C Bruni, Camillo Marra, Annachiara Cagnin, Laura Bonanni, Chiara Cupidi, Valentina Laganà, Elisa Rubino, Alessandro Vacca, Raffaele Di Lorenzo, et al. The impact of covid-19 quarantine on patients with dementia and family caregivers: A nation-wide survey. *Frontiers in aging neuroscience*, 12:625781, 2021.
3. Renjie Li, Xinyi Wang, Katherine Lawler, Saurabh Garg, Quan Bai, and Jane Alty. Applications of artificial intelligence to aid early detection of dementia: a scoping review on current capabilities and future directions. *Journal of biomedical informatics*, 127:104030, 2022.
4. Sandeep Reddy, Sonia Allan, Simon Coghlan, and Paul Cooper. A governance model for the application of ai in health care. *Journal of the American Medical Informatics Association*, 27(3):491–497, 2020.
5. Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

6. Anastasia Grass, Julia Backmann, and Martin Hoegl. From empowerment dynamics to team adaptability: Exploring and conceptualizing the continuous agile team innovation process. *Journal of Product Innovation Management*, 37(4):324–351, 2020.
7. KA Jellinger. Neurodegenerative erkrankungen (zns) – eine aktuelle Übersicht. *Journal für Neurologie, Neurochirurgie und Psychiatrie*, 6(1):9–18, 2005.
8. Zeinab Breijyeh and Rafik Karaman. Comprehensive review on alzheimer's disease: Causes and treatment. *Molecules*, 25:5789, 12 2020.
9. C. G. Gottfries, W. Lehmann, and B. Regland. Early diagnosis of cognitive impairment in the elderly with the focus on alzheimer's disease. *Journal of Neural Transmission*, 105(8):773–786, November 1 1998.
10. Maximilian Kapsecker, Simon Osterlehner, and Stephan M Jonas. Analysis of mobile typing characteristics in the light of cognition. In *2022 IEEE International Conference on Digital Health (ICDH)*, pages 87–95. IEEE, 2022.
11. Teresa Arroyo Gallego, María Ledesma-Carbayo, Ian Butterworth, Michele Matarazzo, Paloma Montero-Escribano, Verónica Puertas-Martín, Martha Gray, Luca Giancardo, and Alvaro Sánchez-Ferro. Detecting motor impairment in early parkinson's disease via natural typing interaction with keyboards: Validation of the neuroqwerty approach in an uncontrolled at-home setting. *Journal of Medical Internet Research*, 20:e89, 03 2018.
12. cloudflare. Was ist ein großes sprachmodell (llm)? `https://www.cloudflare.com/de-de/learning/ai/what-is-large-language-model/`. [Accessed 10-03-2024].
13. Justyna Obara. Is chatgpt safe? `https://nordpass.com/blog/is-chatgpt-safe/`, 2023. [Accessed 12-03-2024].
14. Usercentrics. Data is the new gold – how and why it is collected and sold. `https://usercentrics.com/knowledge-hub/data-is-the-new-gold-how-and-why-it-is-collected-and-sold/`, 2021. [Accessed 10-03-2024].
15. Dan Constantini. Chainlit. `https://docs.chainlit.io/get-started/overview`.
16. Hugging Face. Transformers Library. `https://pypi.org/project/transformers/`.
17. Sebastián Ramírez. FastAPI. `https://fastapi.tiangolo.com`.
18. Jelle Zijlstra & Alex Waygood. typing Library . `https://docs.python.org/3/library/typing.html`.
19. Meta. Llama-2-7b-chat-hf . `https://huggingface.co/meta-llama/Llama-2-7b-chat-hf`.
20. jphme. Llama-2-13b-chat-german . `https://huggingface.co/jphme/Llama-2-13b-chat-german`.
21. Denis A. Evans, H. Harris Funkenstein, Marilyn S. Albert, Paul A. Scherr, Nancy R. Cook, Marilyn J. Chown, Liesi E. Hebert, Charles H. Hennekens, and James O. Taylor. Prevalence of Alzheimer's Disease in a Community Population of Older Persons: Higher Than Previously Reported. *JAMA*, 262(18):2551–2556, 11 1989.
22. Shirley Ann Becker. E-government visual accessibility for older adult users. *Social Science Computer Review*, 22(1):11–23, 2004.
23. Sophie Wuerger. Colour constancy across the life span: Evidence for compensatory mechanisms. *PloS one*, 8:e63921, 05 2013.
24. Wikimedia Commons. File:linear visible spectrum.svg — wikimedia commons, the free media repository, 2024. [Online; accessed 20-February-2024].
25. Dan Abramov and Rachel Nabors. Reactjs. `https://react.dev/blog/2023/03/16/introducing-react-dev`.
26. MUI. Material UI. `https://mui.com/material-ui/getting-started/`.
27. Inc. Remix Software. React Router. `https://reactrouter.com/en/main/start/overview`.
28. i18next. react-i18next? `https://react.i18next.com`.
29. chainlit. chainlit react client package. `https://docs.chainlit.io/customisation/react-frontend`.
30. Auth0 by Okta. JSON Web Tokens. `https://jwt.io`.

31. Ivan Reznikov. How to fit large language models in small memory: Quantization. `https://www.linkedin.com/pulse/how-fit-large-language-models-small-memory-ivan-reznikov#:~:text=However%2C%20LLMs%20are%20also%20very,by%20the%20chosen%20precision%20size.`, 2023. [Accessed 11-03-2024].
32. Ryan Stewart. A simple, practical guide to running large-language models on your laptop. 2024. [Accessed 11-03-2024].
33. Maarten Grootendorst. Which quantization method is right for you? (gptq vs. gguf vs. awq). `https://www.maartengrootendorst.com/blog/quantization/`, 2023. [Accessed 11-03-2024].
34. Hugging face – the ai community building the future. `https://huggingface.co/`. [Accessed 11-03-2024].
35. bitsandbytes. `https://huggingface.co/docs/bitsandbytes/main/en/index`. [Accessed 11-03-2024].
36. Llama 2 is here - get it on hugging face. `https://huggingface.co/blog/llama2#how-to-prompt-llama-2`, 2023. [Accessed 11-03-2024].
37. A guide to prompting llama 2. `https://replicate.com/blog/how-to-prompt-llama#how-to-format-chat-prompts`, 2023. [Accessed 11-03-2024].
38. Touvron et al. Llama 2: Open foundation and fine-tuned chat models. 07 2023.
39. 8 Top Open-Source LLMs for 2024 and Their Uses. https://www.datacamp.com/blog/top-open-source-llms.
40. Meta. Llama 2 — llama.meta.com. `https://llama.meta.com/llama2`. [Accessed 10-03-2024].
41. Sunil Ramlochan. How does llama-2 compare to gpt-4/3.5 and other ai language models. `https://promptengineering.org/how-does-llama-2-compare-to-gpt-and-other-ai-language-models/`, 2023. [Accessed 11-03-2024].
42. Thomas Gebhardt. Meta hat llama 2 veröffentlicht – was kann der chat-bot im vergleich zu bard und chatgpt? `https://de.linkedin.com/pulse/meta-hat-llama-2-ver%C3%B6ffentlicht-kann-der-chat-bot-im-zu-gebhardt`. [Accessed 11-03-2024].
43. Niklas Heidloff. Language support for large language models — heidloff.net. `https://heidloff.net/article/llm-languages-german/`, 2023. [Accessed 11-03-2024].
44. Github - jphme/em$_{g}erman : Repositoryfortheemgermanmodel - - - github.com.$. [Accessed 11-03-2024].

Simran Arora, Avanika Narayan, Mayee F Chen, Laurel Orr, Neel Guha, Kush Bhatia, Ines Chami, and Christopher Re. Ask me anything: A simple strategy for prompting language models. In *The Eleventh International Conference on Learning Representations*, 2022.