D3.5 - Dialogue system capable of handling natural language inputs and outputs

Due date of deliverable: 29/02/2024

Responsible partner: SPXL

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Document Details

Title	Title
Deliverable No.	D3.5
Work Package	WP3
Task	T3.5
Deliverable Type	Other
Lead Partner	SPXL
Contributing Partner(s)	SPXL, AIT, SUPSI
Due date of deliverable	29/02/2024
Actual submission date	29/02/2024
Abstract	This document describes the vocal HMI implemented in task T3.5 and used to control the autonomous driving wheelchair engineered in the REXASI-PRO project
Keywords	Vocal HMI, Conformal Prediction, ChatGPT
Dissemination level:	
PU	Public
RE	Restricted to a group specified by the consortium (including Commission Services)
SEN	Sensitive, limited under the conditions of the Grant Agreement

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Change History

Version	Date	Changed by	Changes Made
1.0	15/01/2024	SPXL	First proposal of the table of contents
1.1	25/01/2024	SPXL	Table of contents consolidated
1.2	5/02/2024	SPXL	First three sections finilised
1.3	15/02/2024	VRS-SUPSI-SPXL-KCL	The rest of the document is finalised and made ready for internal review
1.4	28/02/2024	SPXL	The suggestion provided by DFKI and VRS during the internal review have been implemented

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Table of Contents

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

List of Figures

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

List of Tables

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Acronyms Table

Acronym	Expanded Form
REXASI-PRO	REliable & eXplainable Swarm Intelligence for People with Reduced mObility
AI	Artificial Intelligence
ChatGPT	Chat Generative Pre-trained Transformer
LLM	Large Language Model
STT	Speech To Text
TTS	Text To Speech
ASR	Automatic Speech Recognition
NLU	Natural Lanaugea Understanding
WER	Word Error Rate
RAG	Retrieval Augmented Generation
PDF	Portable Document Format
CP	Conformal Prediction
CRC	Conformal Risk Control
ECAPA	Emphasized Channel Attention, Propagation and Aggregation
TDNN	Time Delay Neural Network
API	Application Programming Interface
URL	Uniform Resource Locator
NLP	Natural Language Processing
ROS2	Robot Operating System 2
QA	Question Answering
HMI	Human Machine Interface

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

1. Overview of the Deliverable

1.1. Scope

This deliverable focuses on describing the hardware and the software that implements the Vocal HMI used to interact with the autonomous wheelchair developed in the REXASI-PRO project. It serves as a high level documentation of the code stored in the project's GIT repository [8] hosted at DFKI. This document also describes some future work that will be conducted outside of task T3.5, but that is strongly based on the developments carried out in that task.

1.2. Audience

This document is a reference for all partners interested in having a high level understanding of how the Vocal HMI has been implemented. Considering the configurability and modularity of the Vocal HMI, reading this document will allow you to use it in different contexts or to use some of its modules as building blocks of other voice-based components.

1.3. Summary

This document, after introducing the role of the Vocal HMI, describes its hardware and software architecture. Finally, it presents some future work that relates with the Vocal HMI.

1.4. Structure

Section 2 introduces the functionalities that the Vocal HMI provides to the user of the wheelchair; Section 3 describes the architecture of the Vocal HMI, both from a hardware and software perspective; Section 4 and 5 introduce future developments based on or related to the Vocal HMI that will be carried out in the REXASI-PRO project, but outside of Task T3.5. Section 4 describes the verification processes that will be implemented for the vocal HMI, while Section 5 focuses on trustworthiness aspects of using ChatGPT in giving assistance to the wheelchair users. Section 6 draws conclusions and 7 lists the functional and non-functional requirements listed in Deliverable D2.4 and referring to the vocal HMI, commenting those whose implementation diverges from the initial description.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

2. Introduction

Autonomous systems are advancing rapidly, and so is their ability to interact with humans to understand their needs and consequently perform the actions necessary to satisfy them. Task T3.5 focuses on giving this capability to the autonomous wheelchair developed in the REXASI-PRO project. The type of interface chosen is a vocal one, i.e. both the human and the autonomous system use voices to communicate: the former uses mouth and ears, the latter speakers and microphone.

The implemented HMI is not a pure vocal HMI, in fact the wheelchair is equipped with a small keyboard in addition to the microphone and speakers. Interaction with it is required for some actions, such as activating the vocal HMI, and accepting or rejecting the execution of a command. This choice allows increasing the reliability of the human-system interaction at a small cost in terms of interaction smoothness.

So far, the vocal HMI has been programmed to answer to three needs of the user:

- **Specify target destination:** the main task of an autonomous wheelchair is bringing the user from their current position to a desired destination. The vocal HMI, when listening to a user sentence, is able to understand that the pronounced command is requesting to reach a target destination, and it is also able to extract the name of this destination so that it can be passed to the controller of the wheelchair that computes the best route to reach it. The destination specified by a user can be of two types: a specific place (like *room 103*) a type of place (like *toilet*). In the second case, it is the controller that, given the type of place to be reached, guides the wheelchair to the closest place of that type.
- **Adjust the driving style:** the autonomous wheelchair is designed to have a human acceptable behaviour. This means that the trajectories and the travelling speed makes the user of the wheelchair (and also the surrounding people) comfortable. However, different users have different needs, some prefer to go faster than others, and some could be more comfortable having bigger distances from walls and other obstacles. These kinds of request can be issued by the user using the vocal HMI. Currently, the possible tuning regards the speed and the distance from the obstacles, however they could change in the future depending on which tuning parameter will be available in the final version of the neural network and the rest of the software stack controlling the wheelchair.
- **Ask information about the wheelchair:** finding the right information in a very dense manual written in small fonts is quite difficult, it would be much easier to ask our devices directly about their functionalities. This is what the vocal HMI of the wheelchair aims at, relying on the possibilities of ChatGPT (or other LLMs).

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Figure 1: *The hardware of the vocal HMI. From left to right: the LG H390 headsets, the Elgato Stream Deck Mini and the Seed Studio J4011*

3. Architecture

3.1. Hardware Architecture

The hardware stack for the vocal HMI consists of four main components:

- a microphone used by the vocal HMI to listen and capture the voice of the user;
- a speaker used by the vocal HMI to speak to the user;
- a small keyboard used to interact with the wheelchair when high reliability is required;
- a computing unit that processes the inputs (voice and key pressed) to generate the commands to be sent to the wheelchair controller and the responses to be spoken to the user.

In our implementation, the microphone and the speaker are integrated in the same device, an LG H390 headset (the first from the left in Figure 1). The choice of this model has been done to guarantee the interaction between the user and the wheelchair also in noisy environments thanks to its noise-cancelling features.

The chosen keyboard is an Elgato Stream Deck Mini (the second from the left in Figure 1). This device has 6 keys, each with a small screen that allows to change the image displayed on it. This allows us to:

- change the functionality of a key depending on the state of the vocal HMI (or more in general of the whole wheelchair);
- use the keyboard not only as an input device, but also as an output one.

The use of the keyboard as an output device is not exploited by the vocal HMI, but it is used extensively in other circumstances, particularly during the real-world data collection in Task T7.3. In this case, the keys on the keyboard are used to display the status of the data collection, so that the person collecting the data can be notified immediately if there is a problem.

As previously mentioned, the keyboard is used where high reliability is needed. Here is a list of the circumstances where the user is requested to interact with it:

- Activating the vocal HMI: the vocal HMI is not always listening to the user voice. This helps to avoid unintended interaction with it (e.g. when the user is speaking with another person). In order to activate the vocal HMI, the user is requested to press a key.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Figure 2: *Interactions between the hardware components*

- Confirming and refusing commands: before executing a command, the vocal HMI asks the user for a confirmation about the action it is going to actuate. Confirmation and refusal take place by pressing two dedicated buttons.
- Triggering the question-answer functionality: a dedicated button allows the user to activate the functionality to ask questions about the usage of the wheelchair.
- Trigger the profile creation: the vocal HMI only accepts vocal inputs from the person using the wheelchair. Its ability to distinguish voices, requires that a voice profile is generated each time that the user on the wheelchair changes. This profile acquisition process is triggered pressing a dedicated key on the Stream Deck.

The chosen processing unit is a Seed Studio J4011 (the third from the left in Figure 1) featuring a Jetson Orin NX with 8GB of RAM. This module, which delivers up to 70 TOPS AI performance, offers also a rich set of input outputs solutions including USB 3.2 ports, HDMI 2.1, M.2 key E for WIFI and M.2 Key M for SSD.

The headset and the keyboard are both connected to the processing unit via USB cable, and their interactions are summarized in Figure 2. The headset captures the user's voice and sends it to the processing unit, which generates a response that is sent as an audio signal to the headset's speakers, which play it back for the user to hear. The keyboard sends the code of the keys pressed by the user to the processing units, which use it to command the behaviour of the vocal HMI.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

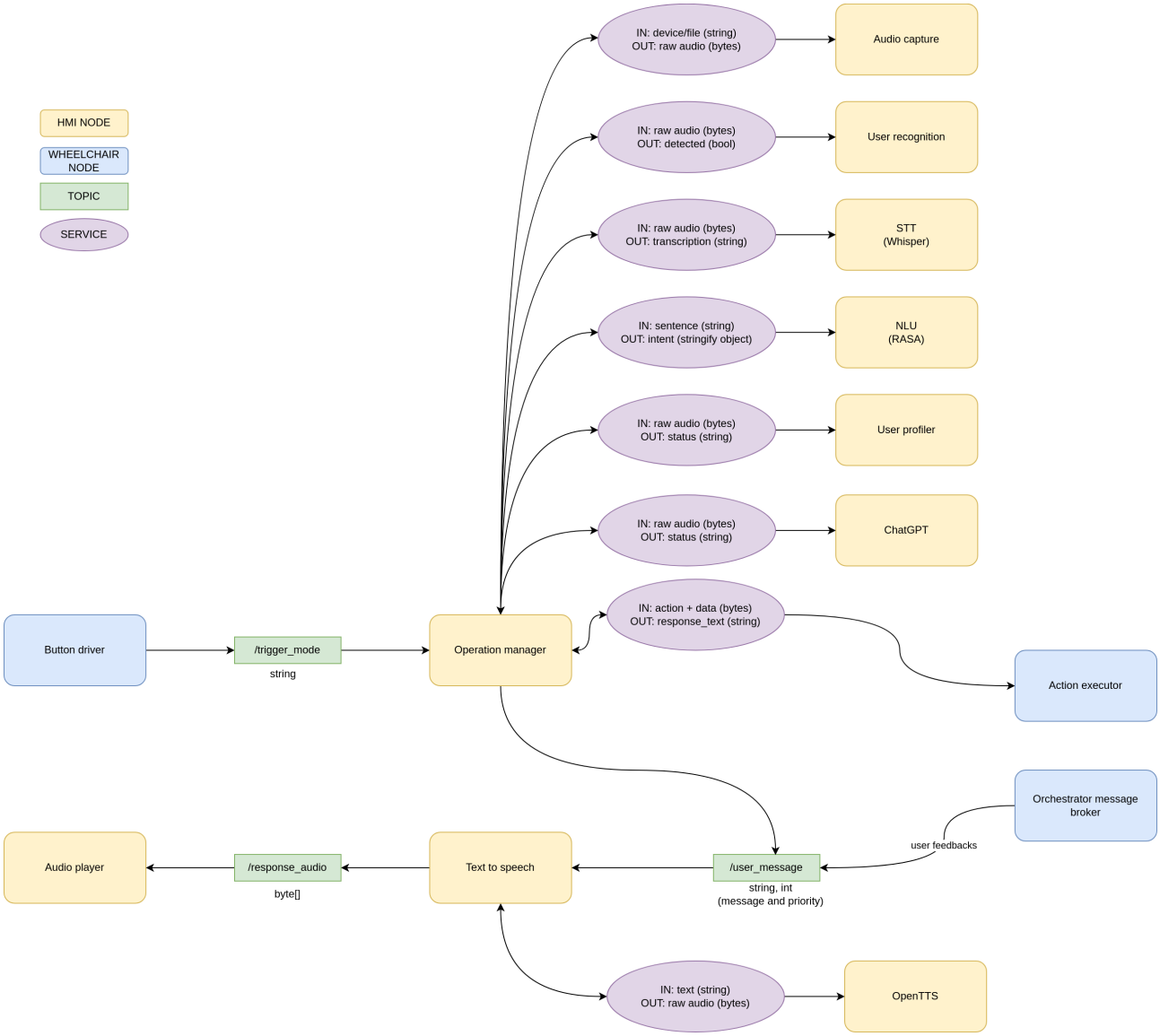

Figure 3: *Software architecture of the vocal HMI*

3.2. Software Architecture

The process of capturing an audio signal, interpreting it and taking as a consequence one or more actions, involves many software components. All of them have been implemented in the ROS2 framework [26] using Python and are shown in Figure 3. In the picture, we represented all the ROS2 nodes, services and topics required for the implementation of the vocal HMI. We give a short description of each of these ROS2 component types:

- **Nodes:** they are the unit of computation, where the logic takes place. Nodes can exchange information among them using services and topics (described later). Nodes can reside on the same machine, but ROS2 also allows distributing them to different machines.
- **Services:** a node can expose a service (service server). This means that it exposes an interface that allows other nodes (service clients) to ask it to accomplish a task. Usually, the interaction is synchronous and expected to be fast:

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

the service client sends an input and waits for the service server to precess it and answer with the proper output.

- **Topics:** nodes can use topics to communicate in a publish-subscribe model. This model is based on three concepts: channel, publisher and subscriber. A channel is an object where one or more publishers publish a message and one or more subscribers listen for such messages. This leads to the implementation of event-driven logics that are intrinsically asynchronous.

We now proceed describing the role of the nodes and the channels represented in Figure 3. We start with the nodes that are part of the vocal HMI (shown in orange in the picture), to continue later with those that are external but interact with those of the vocal HMI (shown in blue in the picture).

- **Audio capture:** this node, when requested, starts recording the input audio source (in our case the microphone of the headset). It stops the recording if the user become silent for some seconds.
- **User profiler:** this node receives an audio recording and uses it to extract a vectorial embedding of the voice of the user. For creating the voice embedding, the node relies on speechbrain [30], an open-source and all-in-one conversational AI toolkit based on PyTorch. It comes with pre-trained models for Speech Recognition, Text-to-Speech, Speaker Recognition, Speaker Verification, Speech Enhancement, Speech Separation, Spoken Language Understanding, Language Identification, Emotion Recognition, Voice Activity Detection, Sound Classification, Grapheme-to-Phoneme, and many others. This node uses the Speaker Verification component. It is composed of an ECAPA-TDNN model, a combination of convolutional and residual blocks. The embeddings of the voices are extracted using attentive statistical pooling. The system is trained with Additive Margin Softmax Loss.
- **User recognition:** this node takes as input a voice recording and tells if its voice belongs to the current user of the wheelchair. To do this, it computes the voice embedding of the recording, and computes its cosine similarity with three embeddings computed on three sentences previously pronounced by the user. The node answers that there is a match if at least one of the calculated cosine similarities is below a given threshold. This module, as the *User profiler* one, relies on the speechbrain [30] toolkit.
- **STT:** SST (Speech To Text) node takes as input an audio recordings with a sentence pronounced by the user and converts it to text. To accomplish this task, the node relies on Whisper [29], a transformer based encoder-decoder model pre-trained for automatic speech recognition (ASR) and speech translation. Trained on 680k hours of labelled data, Whisper demonstrates a strong ability to generalise to many datasets and domains without the need for fine-tuning.
- **NLU:** the NLU (Natural Language Understanding) node takes as input a string, and returns an intent and the entities necessary to complete the intent. For example, given the sentence "Go to office 105" the extracted intent is *provide_destination*, the entity is *office 105*. Given this structured information, it is easy to understand that the user has specified a destination and that this destination is office number 105. This node performs intent and entity extraction using Rasa [11], a natural language understanding module. It com-

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

prises loosely coupled modules combining a number of natural language processing and machine learning libraries in a consistent API. It aims a balance between customisability and ease of use. To this end, there are pre-defined pipelines with sensible defaults which work well for most use cases. This tool requires to be trained to recognize the set of intents and entities used in the context of application. The information to be supplied for the training is the list of intents. For each intent, it is necessary to supply a list of sentences that well represents it and highlight which part of the sentence are entities to be extracted. In Listing 1 you see how the *provide_destination* intent is defined.

```
version: "3.1"

nlu:
- intent: provide_destination
  examples: |
    - Can you take me to the [information desk](destination)?
    - I need to go to the [elevator](destination).
    - Please take me to the [restroom](destination).
    - Can you show me the way to the [cafeteria](destination)?
    - I want to go to the [conference room](destination).
    - Can you take me to the [office 24](destination)?
    - I need to go to the [administration office](destination).
    - Please show me where the [waiting area](destination) is.
    - Can you take me to the [parking garage](destination).
    - Show me where the [north exit](destination) is.
    - Can you take me to the [south way out](destination)?


```

Listing 1: *Definition of the* provide_destination *intent*

- **OpenTTS:** this node, given as input a sentence in string format, returns the same sentence in audio format. It relies on OpenTTS [5], a wrapper to multiple open source systems that gives support for many languages and voice tones.
- **Text to speech:** this node subscribes to the *user_message* channel, from which it receives text messages with a priority. These messages are queued depending on priority, converted to audio using the *OpenTTS* node, and published as a sequence of bytes to the *response_audio* channel.
- **Audio player:** this node subscribes to the *response_audio* channel and plays all the audio it receives on the output device (the speakers of the headsets in our case).
- **LLM:** this node, given a text message with a question, uses an LLM (Large Language Model) to answer. Currently, it is using gpt-3.5-turbo but it can be configured to use other models. The node is tailored to answer questions about the wheelchair usage and its characteristics, extrapolating the knowledge from the manufacturer manual of the device. The process to extract this data is explained in Section 3.2.1.

Among the nodes that are part of the vocal HMI, we have not yet described the *Operation Manager*. However, we will leave it as the last one of this section, since it interacts with almost all the nodes of the graph shown in Figure 3. Now we proceed with the description of the nodes that are external to the vocal HMI.

- **Button driver:** the button driver node reads all input from the keyboard and translates it into messages with the code of the keys pressed, which are published on the *trigger_mode* channel.
- **Action executor:** this node takes as input an action command to be executed

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

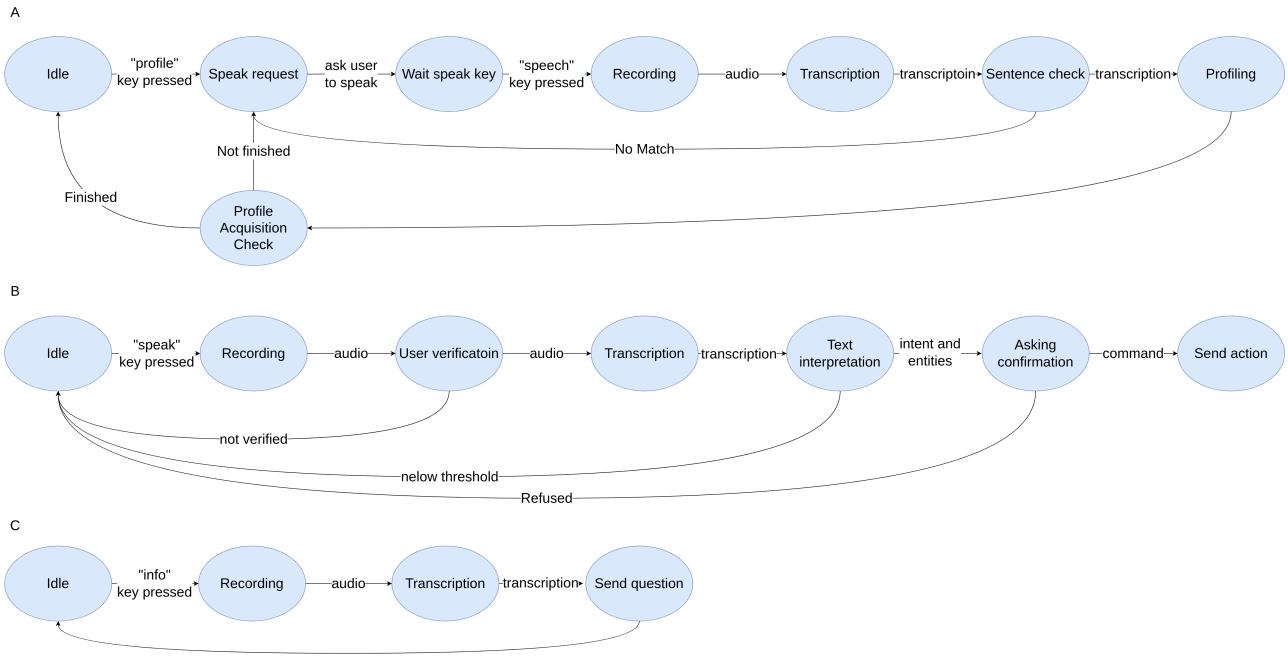

Figure 4: *State machine implemented in the* Operation manager *node*

on the wheelchair. For example, if the user asks "Go to office 105", this node takes as input the action code corresponding to the *provide_destination* action with the name of the destination to be reached. Optionally, it returns a message to be played to the user. For example, if office 105 is a non-existent destination, the node command returns a text message informing the user of this fact.

- **Orchestrator message broker:** the REXASI-PRO project includes the presence of an orchestrator that - having a global knowledge of the spaces in which the wheelchair is moving - provides feedback such as the best routes or the need to brake to avoid a moving obstacle. For the sake of explainability, the orchestrator communicates the reasons for its decisions using the vocal HMI. To do this, it sends a text message to the wheelchair, that is received by the *Orchestrator message broker* node that publishes it on the *user_message* channel. This action causes the message to be played through the user's headphones. Note that messages played on the *user_messages* channel have a priority that is used to order the messages in the queue where they wait to be played. The messages sent by the *Orchestrator Message Broker* always have high priority and are published before the other messages.

Finally, we describe the *Operation manager* node, which has the most interactions with the other nodes and is the main actor behind the orchestration of the vocal HMI. This node subscribes to the *trigger_mode* channel, where the *Button driver* node publishes the codes of the keys pressed by the user on the keyboard. When the *Operation manager* receives a message from the *trigger_mode* channel, it reacts as described by the finite state machine shown in Figure 4. When the vocal HMI starts, the *Operation manager* is in the *Idle* state of the state machine. It leaves this state as a result of three events that lead to three different action streams, which we describe below:

- The user presses the *profile* key on the keyboard (see Figure 4.A): the state

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

machine of the *Operation manager* switches to state *Waiting for profile acquisition* and takes the following actions:

- It publishes a message on the *user_message* channel, asking the user to press the *speak* key and say the first of the three phrases to be pronounced to complete the profile acquisition: "Hello wheelchair". This action triggers the *Text to Speech node* which, after a few passages, leads to the message being played through the user's headphones.
- The state machine changes its state to *Wait speak key* and the *Operation manager* waits until the *speak* key is pressed.
- At this point, the state machine jumps to state *Recording*, where the *Operation manager* queries the *Audio capture* node to capture the next sentence spoken by the user.
- Once the sentence is recorded, the state machine enters the state *Transcription* that leads the *Operation manager* to interact with the *STT* node to convert the voice of the user to a text message.
- The state machine jumps to the *Sentence check* state, where it checks if the sentence pronounced by the user is the expected one (i.e "Hello wheelchair"). The check is done with the library JiWER [3] computing the WER (Word Error Rate). This metric is computed as the Levenshtein distance between the recognized word sequence and the ground truth transcription divided by the number of words in the ground truth transcription. The Levenshtein distance is defined as the minimal number of substitution, insertion and deletion operations required to turn one word string into another.
- After this, if the WER score is above the threshold of 0.5, the state machine jumps to the *Speak request* state asking the user to repeat the sentence, otherwise it moves to the *Profiling* state where the *Operation manager* computes the embedding of the audio recording relying on the *Profiler* node.
- Once the embedding is computed, the state machine enters the state *Profile acquisition check*. If all the sentences required to complete the profile acquisition have been properly pronounced the profile acquisition is finished and the embeddings of the pronounced sentences are stored. Otherwise, *Operation manager* jumps to the *Speak request* state and asks the user to pronounce the next sentence. The complete list of sentences to be pronounced is "Hello wheelchair", "Bring me to the elevator" and "Go to the cafeteria".
- Finally, the state machine goes back to the *Idle* state and the *Operation manager* is ready to accept the next command.

- The user presses the *speak* key (see Figure 4.B): the *Operation manager* state machine jumps to the *Recording* state and takes the following actions:
  - By querying the *Audio capture* node, the *Operation manager* acquires the recording of the next sentence pronounced by the user.
  - The state machine changes its internal state to *User verification*, and the *Operation manager* sends the acquired recording to the *User verification* node. This node computes the embedding of the recording and com-

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

pares it with those of the profile of the current user of the wheelchair (those computed during the profiling procedure where the user pronounced the sentences "Hello wheelchair", "Bring me to the elevator" and "Go to the cafeteria").

- If there is no match between the embedding and the user profile, a message informs the user about it and the state machine jumps back to the *Idle* state. Otherwise, the state machine enters the *Transcription* node and the *Operation manager* sends the recording to the *STT* node that answers with its transcription.
- At this point, the state machine jumps to state *Text interpretation*, where the *Operation manager* asks the *NLU* node to extract from the transcriptions intent and entities.
- If such information is extracted with a too low confidence, the state machine jumps back to the state *recording* and the *Operation manager* asks the user to repeat the command. Otherwise, the state machine changes state and moves to the *Asking confirmation* state, in which the *Operation manager* sends a text message to the *user_message* channel asking the user to confirm the understood command.
- If the user presses the "refuse" button, the state machine goes back to the *Idle* state, otherwise, it moves to the state *sending action* where it sends the command to the *Action executor* node that executes the command.
- Finally, the state machine goes back to the *Idle* states and the *Operation manager* is ready to accept the next command.

- The user presses the *info* key (see Figure 4.C): the state machine of the *Operation manager* jumps to the *Recording* state and takes the following actions:
  - As the first step, it contacts the *Audio Capture* node to record the next sentence pronounced by the user.
  - Once the recording is finisched the state machine changes state to *Transcription* and the *Operation manager* contacts the *STT* asking for the transcription of the just captured recording.
  - Then the state machine jumps to state *Send question* and the *Operation manager* sends the question of the user to the *LLM* node that answers with a text message that is played in the headphones of the user.
  - Finally, the state machine goes back to the *Idle* states and the *Operation manager* is ready to accept the next command.

3.2.1. Pipeline to answer questions using an LLM

The vocal HMI allows the user to ask questions about the characteristics and functioning of the wheelchair. This is achieved by relying on an LLM, specifically GPT 3.5 turbo, which is instructed to extract its answers from a specific source: the wheelchair manual. Currently, we are using the original manual of the wheelchair, provided by Ottobock, because a manual describing the wheelchair with the additional features provided within the REXASI-PRO project is not available.

Basic notions on LLMs: the used LLM is based on a transformer model and works by receiving an input, encoding it, and then decoding it to produce an out-

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

put prediction. To achieve its capability to receive an input and generate an output prediction, the model requires training, so that it can fulfil general functions, and fine-tuning, which enables it to perform specific tasks. Therefore, in a first stage, LLMs are pre-trained using large textual datasets from sites like Wikipedia, GitHub, or others. These datasets consist of trillions of words, and their quality will affect the language model's performance. At this stage, the large language model engages in unsupervised learning, meaning it processes the datasets fed to it without specific instructions. During this process, the LLM's AI algorithm can learn the meaning of words, and of the relationships between words. It also learns to distinguish words based on context. For example, it would learn to understand whether "right" means "correct," or the opposite of "left." To make the LLM perform a specific task, such as translation or question answering, it must be fine-tuned to that particular activity.

Focus a question answering LLM on a specific topic: the dataset used to train LLMs is large, but it cannot cover all the knowledge needed to answer any input of a question-answering task. For example, the LLM cannot answer questions about events that occurred after training or about very specific topics. To solve this problem, as a first approach one could think of retraining the model with documents containing the missing information, but this approach is not always feasible due to the complexity of the training procedure, the resources required to perform it, the time it takes, and last but not least, the availability of the training pipeline. A more common approach to this problem is to properly engineer the input (also called *prompt*) to the model, adding the context from which it is expected to extract the information. Thus, if we want to get an answer about the fairy tale *Little Red Riding Hood*, the input to the model would be:

> Considering this fairy tale: "Once upon a time ..." could you tell me what animal eats Little Red Riding Hood?

and not:

> Could you tell me what animal eats Little Red Riding Hood?

This technique is also very useful to limit the number of hallucinations generated by the model, i.e. situations where it produces an output that is coherent and grammatically correct but factually incorrect.

Handling large contexts with RAG: over time, the dimension of the prompt accepted by the LLMs has grown, however it is always limited to a fixed number of characters. For example, in the case of ChatGPT it is about 3000 characters. It often happens that the dimension of the documents that we want to pass as prompt context (hereafter *knowledge base*) is much larger, and this is the case also for the manual of the wheelchair that is about 6 million characters long. To address this problem, a widely used approach is called Retrieval Augmented Generation (RAG).

This approach combines the usage of an LLM with a retrieval tool whose purpose is, given a question, to extract those parts of the knowledge base that satisfies the following:

- to have fewer characters than the maximum accepted by the LLM prompt
- to contain the information required to give an answer to the question

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

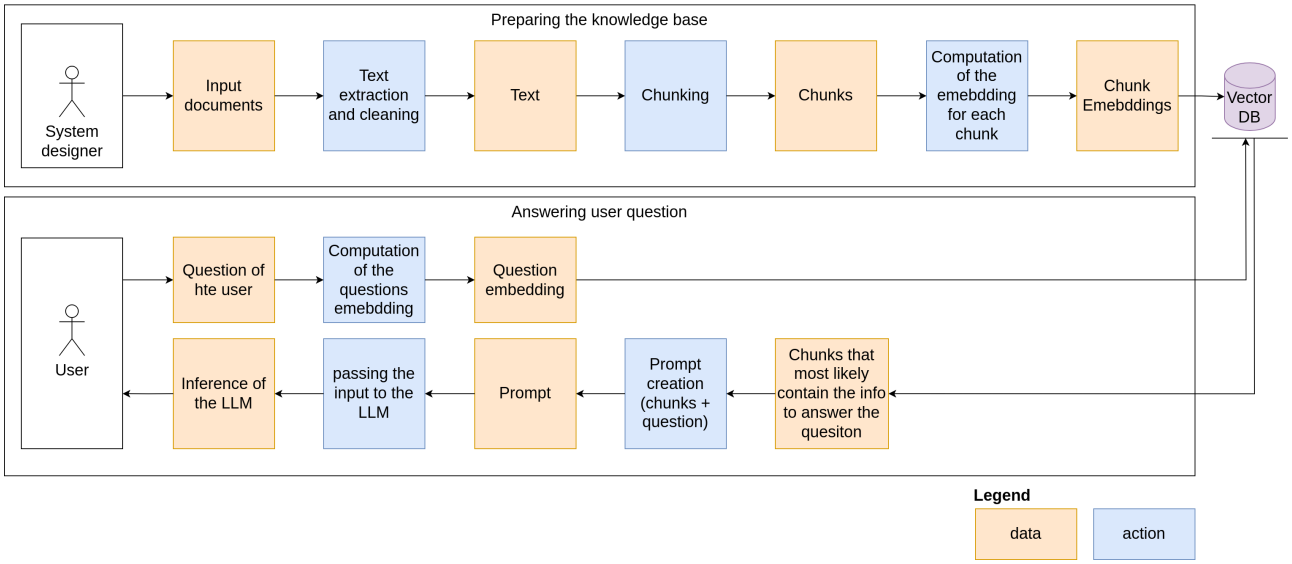

Figure 5: *Descriptions of the RAG approach: above the pipeline to import the knowledge base in the vector database, below how to exploit this information to answer the question of a user*

In order to be able to accomplish this task, the retrieval tool needs to preprocess the knowledge base taking these steps (shown in the top part of Figure 5):

- The knowledge base needs to be imported in a machine-readable format. For example, in the case of a PDF, text needs to be extracted from paragraph and tables.
- The knowledge base is divided in chunks, i.e. the whole text is divided in parts such that some of them (a reasonable number is between 5 or 10) can fit in the prompt of the LLM.
- For each chunk, an embedding is computed. This can be done with an LLM but also with other approaches.
- The so computed embeddings are stored with the corresponding chunk in a vector database, where they can be easily retrieved using different metrics (e.g. by cosine similarity with a given embedding)

Once the embedding of the chunks are in the vector database, these are the steps to take to answer questions based on the information contained in the knowledge base (shown in the bottom part of Figure 5):

- Compute the embedding of the question posed by the user with the same method used to compute the embeddings of the chunks of the knowledge base.
- Query the vector database passing as input the embedding of the question. The database finds the chunks with the higher probability to contain the information to answer the question. There are many approaches to accomplish this task, the simplest consists in extracting the chunks having the embeddings with the highest cosine similarity with the embedding of the question.
- Compose the prompt of the LLM by concatenating the chunks corresponding to the embedding extracted in the previous step and then asking the LLM to answer the questions only relying on the information contained in that chunks.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Implementation of the RAG approach within REXASI-PRO: so far we gave a general description of the RAG approach, here we describe how we instantiated it in the REXASI-PRO project mentioning tools and methods used to implement the blue boxes represented in Figure 5:

- **Text Extraction and Cleaning:** the extraction of text data from the document was made possible by the Python library pdfplumber [7], which allows extraction of text and relevant metadata from pdf file; this tool also offers a visual debugger, which lets you easily inspect the result of the algorithm, testing its ability to correctly parse every character in the file. In this sense, each document was scanned, and the information associated to each page were extracted, such as page content, page number, annotations, URLs, and document title. The pages content were used for extracting the information for the LLM, while the other information were used as metadata. A cleaning phase was applied using a tool called cleantext [1]: it is a collection of functions that lets you parse, clean and format text data. This phase was applied for normalize the text outputted by the pdf parser and translate to the same encoding of all the characters.
- **Chunking:** for the chunking phase the RecursiveSplitterFunction from Langchain library [4] was used, which is a well-known resource for managing LLMs. The aforementioned function takes a text as input, and it recursively splits it according to specific delimiters, that denote the last chars of text's phrases.
- **Chunk and question embedding computation:** for the embedding part, a transformer model was applied, the 'all-MiniLM-L6-v2' [2], which is a embedder model that maps sentence and paragraphs to a 384 dimensional dense vector space, which is well suited for task like clustering or semantic search.
- **Chunk retrieval:** given an input question, we compute its embedding with the same method used to compute those of the chunks. Then we ask the vector database to return the 5 chunks whose embedding is closest to that of the question. For this step we use the Euclidean distance.
- **Prompt creation:** the prompt used consists of three parts: PREFIX, CONTEXT, and SUFFIX, described as follows:
  - PREFIX: defines the general statement in the input to the model. Describes the scope of the query, the model's response behaviour, and the generation constraints and/or requirements.

```
1  You are an intelligent wheelchair powered by artificial intelligence that was
   made for helping users answer their questions.
2  Your goal is to give concise and informative answers based solely on the given
   DOCUMENTS.
3  Please reply to the question using only the information present in the DOCUMENTS
   below.
4  If the answer is not contained within the text above, reply politely that the
   information is not in the knowledge base.
5  You must only use information from the given DOCUMENTS.
6  Use an unbiased and journalistic tone and answer only with the essential
   information .
7  Do not repeat text.
8  If you don't know the answer, just say that you don't know, don't try to make up
   an answer.
9
```

Listing 2: *PREFIX of the prompt*

  - CONTEXT: contains the chunks extracted by the vector database that are

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

inherent to the user query. It has a specific format: it starts with *CONTEXT* and lists every piece of information associated to the chunk.

```
DOCUMENTS:
- not use the wheelchair in case of ...
- the intended use of the wheelchair is ...
- do not apply external, not tested devices to ...
- refer to the Section 14.0 for understanding...
- Section 14.0: Usage of brake system in ...

```

Listing 3: *CONTEXT of the prompt*

  - SUFFIX: represents additional information regarding the prompt, or/and defines an output format for the LLM. Generally is the last piece of information ingested by the model, and can drive the output of the generation, manipulating the generation after the PREFIX and CONTEXT phase. We will give more details on this in the next paragraph, where we explain the strategy implemented to increase the explainability of the answer received from the LLM.

- **Passing the input to the LLM:** The LLM used is chatGPT based on GPT 3.5 turbo. The inference is done using the OpenAI APIs [6], where both the generated prompt and specific additional parameters are entered. The additional parameters play a critical role in ensuring reproducibility, which is essential for the model to produce identical responses to identical queries. The parameters include
  - *random-seed*: Seed value for the random number to make the results reproducible.
  - *temperature*: metric that controls the amount of randomness in the generated output: we set it to 0 in order to get the same output when we provide the same input multiple times.

Explainability of the answer: the described approach can fail to provide the correct answer for multiple reasons, these are some examples:

- the chunks extracted from the knowledge base do not contain the information required to answer the question
- the LLM misinterpreted the information contained in the chunks
- the LLM generates non-factual or nonsensical answers (i.e. hallucinations)

One mitigation strategy for such events is to provide the user with the portion of text in the knowledge base from which the LLM extracted its answer so that the user can verify its reliability. A first step in this direction is to provide the user with the chunks from which the answer was generated. However, the chunks may consist of several sentences, and it is quite difficult for the user to go through all of them. A better approach would be to highlight the sentences that were more relevant for the LLM to generate its answer. To achieve this result, we modified the pipeline shown in Figure 5 in two points:

- We modified the prompt generation step by adding a SUFFIX to the prompt asking the LLM to assign a score between 0 and 1 to each sentence of the chunks, depending on how useful the sentence was for generating the answer. Finally, we asked the LLM to return only the sentences with a score greater or equal to 0.1 in a table with two columns: the first with the sentence and the second with the score associated with the sentence.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

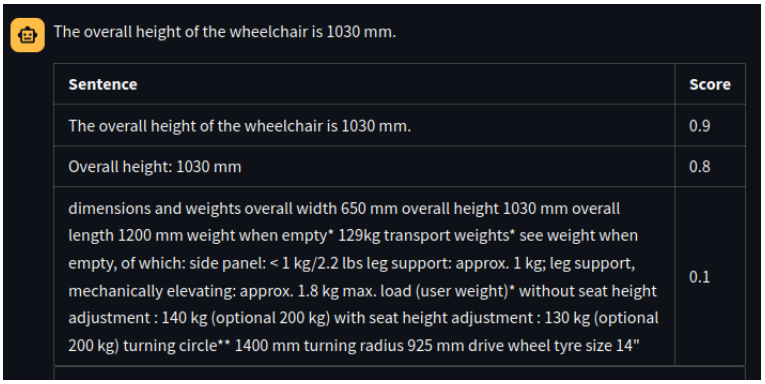

Figure 6: *Example of answer provided by the model to the question "What is the overall height of the wheelchair?"*

```
For every sentence inside DOCUMENTS, detect each sentence and assign a score between
0 and 1 that reflects how informative it is to answer the question.
Return the answer to the question and a markdown table that shows the sentences with
a score greater than 0.1, where each row
has two data entries, one is the sentence, and the other is the score associated with
 it.

```

Listing 4: *SUFFIX of the prompt*

- Since the inference now consists of two parts (the answer to the question and the table with the most important sentences used to infer the answer), we added an extra step at the end of the pipeline to: first, extract the answer and send it to the vocal HMI to be played in the user's headphones; second, get the table with the sentences used to infer the answer, search for these sentences in the pdf file from which the knowledge base was extracted (in our case, the wheelchair manual), and finally highlight such sentences in the original pdf.

  In the case of the wheelchair, the resulting PDF cannot be shown to the user, but in a use case where a screen is available, it could be and would provide useful insights to understand if the user can trust the LLM's answer.

You can see an example of inference provided by the model with the described prompt in Figure 6.

This preliminary work regarding the explainability and reliability of ChatGPT in the context of REXASI-PRO will be extended outside of task T3.5 as described in Section 5.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

4. Dialogue-based Guidance Verification

By the end of the second year of the REXASI-PRO project, as the result of task T.4.5, the system for the dialogue-based guidance is going to be analyzed with the objective of reducing or mitigating accidental or adversarial threats. More specifically, the verification process will tackle two main challenges: (i) the consistency of the speech w.r.t the predicted actions ($S \rightarrow A$), and (ii) the causality relation between action and speech ($S \leftarrow A$). The motivation for this analysis is that the speech-to-action sub-system, that associates a wheelchair command to the voice commands (given by the user), is complex and may be exploited to negatively impact the safety of the user.

4.1. Speech to Action ($S \rightarrow A$)

This analysis will ensure that the system reliably predicts actions, or intents, from a given speech signal. Based on the architecture of Figure 3, we have two relevant prediction tasks to verify: 1) predicting text from speech, and 2) predicting action/intent from speech (the latter prediction is the sequential composition of the STT and NLU models).

To this purpose, we will resort to the frameworks of conformal prediction (CP) [38, 9] and conformal risk control (CRC) [10]. In the context of intent prediction, CP would derive prediction regions, i.e., sets of plausible actions, guaranteed to include, with arbitrary probability, the true (unknown) action for a given input speech signal. Such a region directly quantifies the uncertainty of the prediction for the input speech. When the region is not a singleton, i.e., the prediction is uncertain and we cannot reliably exclude alternative output intents, the user is asked to repeat the command.

CRC generalizes CP by allowing to control the expected value of any loss function (in particular, monotone loss functions that shrink as the size of the prediction region grows). This is especially useful in settings where we do not require the prediction region to cover the ground truth output *exactly*, but in a looser and more flexible manner. CRC will be used to verify the speech-to-text model: as done in [13], we can select our loss function in such a way that the resulting prediction regions are guaranteed to include, with a given probability, at least one sentence whose Word Error Rate (WER) w.r.t. the ground truth text is below a given threshold. For the same settings, CP instead would require that the region includes at least one sentence equal to the ground truth text, which is arguably too strict of a requirement for this prediction task.

Our approach will also ensure that these probabilistic guarantees hold despite distribution shifts that arise between training and test data [37]. In particular, we will consider covariate shifts (i.e., deviations in the distribution of inputs) occurring due to 1) different levels and kinds of environmental noise (e.g., crowded spaces, wind, traffic) and 2) the characteristic voiceprint of the test subject that, due to anatomical and acoustic factors, is distinguished from the voiceprints of the speech signals used for training.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

4.2. Action to Speech ($S \leftarrow A$)

The methodology that will be adopted for the verification of the causality relation between actions and speech is as follows.

1. Threat Identification and Assessment: The speech-to-action subsystem (described beforehand in this deliverable) is abstracted into a functional and physical architecture. The functionalities (e.g., the STT and the NLP components) and the physical components (e.g. the headset) are associated with a number of known generic threats. Those threats are then articulated as a (threat) scenario detailing how the threats could be exploited to negatively impact the causality relation between actions and speech.
2. Mitigation Strategy: The threat scenarios will be mitigated by analyzing the implementation of the speech-to-action subsystem (e.g., the software architecture based on ROS2) and its cybersecurity, ultimately refining the system implementation and cybersecurity architecture.
3. The threat scenarios will also be evaluated against an impact and likelihood scale for the prioritization of the effort for the implementation of the mitigation strategy.

One of the challenges identified in the project proposal is that the user authentication on the HMI may not be sufficient to guarantee that the speech commands are actually given by the user. Both accidentally or maliciously, commands could be given not by the user but by another user in proximity to the microphone of the wheelchair, “hijacking” the authenticated session of the user. Therefore, techniques for the continuous authentication of the user will be investigated as mitigation of this threat.

This project has received funding from the European Union’s Horizon Europe research and innovation program under grant agreement **No 101070028-2**

5. Trustworthiness of ChatGPT

The recent boost of innovation in Large Language Models (LLMs), like ChatGPT from OpenAI [28], is impacting our society as a whole. For instance, in the workplace, more and more text authoring tasks are supported or even completely automatized by LLMs. More natural and intuitive speech-based Human-Machine interfaces (HMI), like those embedded in most current operating systems, are another prominent application of LLMs.

In the context of REXASI-PRO, we investigate novel methods to include ML components in trustworthy-by-design architectures for assistive technologies. In this deliverable, we focus on using LLMs to enhance the interaction between users and smart wheelchairs. In particular, can we leverage ChatGPT to answer questions about the wheelchair's functioning? Can we trust the answers that it provides?

This section presents the current state and future plan of our investigation of these questions, whereas technical details about the integration of ChatGPT in the REXASI-PRO HMI software architecture are presented in Section 3.2.1. The section is organized as following: Section 5.1 introduces the investigation in more details, Section 5.2 describes related works, and Section 5.3 introduces the data analysis method and the collected dataset. The evaluation on the dataset is ongoing and results will be part of a scientific contribution in preparation. Finally, Section 5.4 illustrates our plans to monitor/improve trustworthiness at runtime using auxiliary modules based on machine-learning, which will be part of the next scientific investigation.

5.1. Trustworthiness of Large Language Models

Like most Deep Learning methods, LLMs represent black-boxes for their users. Anecdotally, they work well, yet they do not provide guarantees on their performance when answering domain-specific questions, like during the workflow presented in Section 3.2.1. How can we trust that their answers are correct and useful for the user asking them? How can we trust them to declare their ignorance when not provided with enough information to answer a specific question?

Trustworthiness of a software component relies on many factors, some of which are subjective to the users [35]. We focus on the three most relevant features for our application context:

Accuracy : are answers correct?

Robustness : are answers robust to small variations that generate semantically similar questions, like it would be expected from human interlocutors?

Explainability : can LLM-based modules explain why they provide a particular answer, and in particular, identify the source of information in the original context?

In order to answer these questions and assess how well an LLM performs in a specific domain, we need to:

1. define appropriate metrics for the target properties listed above;
2. define a dataset of representative questions from users;

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

3. evaluate the metrics on the dataset.

While the dataset (see Sections 5.3.1) and the related results are specific to the application domain, we present a general methodology to generate them, which can be applied to assess the performance of LLMs-based Q&A in other domains.

5.2. Related work

Different works have already addressed trustworthiness-related topics with respect to LLMs, either from the high-level perspective [25] or discussing a particular aspect focusing on a precisely defined set of research questions [20, 33, 34, 17]. Although trustworthiness as an umbrella term is a bit fuzzy and encompasses different things, the research mainly focalizes on topics of reliability [33, 43, 20, 18, 34, 24], robustness [43], consistency [17, 24, 36] and finally, accuracy [18, 36] as a way to measure the performance of LLMs in question answering setup. As the focus of this work is on a QA task, we limit the scope and leave out of consideration other studies focusing e.g. on robustness and reliability checks of LLM code generation [43]. Although different LLMs have been considered in QA-related works on the topic, the cornerstone of quite some of the studies remains to be ChatGPT [33, 18, 36, 17].

To assess model reliability, initial studies performed a straightforward QA evaluation [18, 36], while more advanced studies typically resort to prompting, trying to use prompts for improving reliability [34] or simply investigating how much different prompt formulations influence the answer of a question [20, 33, 34]. As an alternative to prompting for assessing model reliability, adversarial examples were also used, exploiting how much perturbed but semantic-preserving questions impact the answers [33]. Some authors even provide more precise definitions of particular aspects, e.g. reliability is defined as the ability to generalize, reduce social bias, calibrate output probabilities and update factual knowledge [34].

Current findings reveal that: 1) ChatGPT's correctness varies across question domains [33]; 2) prompting is extremely important as even the minor prompt changes might lead to drastic differences in the answer [20] affecting both the accuracy in the answering as well as the identification of unanswerable questions (e.g. when the prompt is used for assigning a particular role to ChatGPT [33]); moreover, the more subjective the prompt is, such as asking for opinion or agreement with the user, the more likely it is that the obtained answer is incorrect [20]; 3) ChatGPT is vulnerable to sentence-level and character-level attacks while synonyms attack are mostly ineffective [33], however, the adversarial attack impact is assessed only on yes-no and multiple choice questions. Another remarkable finding is that ChatGPT tends to make meaningless guesses rather than to reject to answer, identifying only 27.8% of unanswerable questions [33].

Most of the studies rely on available datasets (either entirely [34] or sampling from these randomly [33] or curating the specifically tailored datasets e.g. [20] do it based on different levels of truth using Wikipedia as a source, or augmenting them using paraphrasing [17]). These datasets encompass diverse domains related to general topics (as found in Wikipedia) [20] as well as particular domains such as recreation, medicine, (social) science, technology, law, history [33].

Very rarely the datasets were constructed from scratch and are domain-specific

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

such as in [18, 36]. In contrast to our study, these specific purpose and domain datasets were typically designed for the medical domain [18, 36]. In [36], a dataset of clinical yes-no questions in endodontics was generated, with questions classified in three different levels of difficulty based on the complexity of the endodontic procedure involved, the level of knowledge required and the likelihood of an incorrect response by a general dentist. The dataset in [18], constructed to assume binary or descriptive answers, was also particular as the questions originated from different specializations and were categorized based on their difficulty in easy, medium and hard, but the judgement on difficulty was entirely relying on author's subjective opinion. This is opposite to our case, where the categorization was done in a more systematic way. It is also worth mentioning that the results of these studies are contradictory, as one states that the overall accuracy of ChatGPT was fairly high across question types and difficulty (although being a bit reserved towards specific specializations and undoubtably reliability) [18] while the other claims significant differences in accuracy depending on question difficulty [36].

Evaluation metrics used in current literature for characterizing the quality of the answers vary with respect to the evaluation goal. For correctness, studies have used accuracy for yes-no and multiple-choice answers, F1-score and Rouge-L for extractive and abstractive answers [33] as well as a six-point Likert scale evaluation [18] or exact match [36]. Additionally, for completeness, a three-point Likert scale was used [18]. Consistency was evaluated based on the proportion of matching responses additionally employing confidence intervals and binomial Wald method. Perplexity was used for evaluating fluency [33]. For evaluating the impact of adversarial attacks, attack success rate measuring the fraction of adversarial examples that ChatGPT answers incorrectly was used [33].

With the recent expansion of large pre-trained models for NLP, the need for more data becomes more and more critical. Given that many tasks and domains have small training dataset, there is a need to increase the variety of examples used to train these models. This is exactly where data augmentation steps in the game. In NLP, data augmentation could be done using different interventions on character (e.g. any token-level random perturbation operations including random insertion, deletion, and swap [40]), word (e.g. synonym or antonym replacement [19, 42, 40, 39], context augmentation [21]) and sentence level (e.g. using dependency tree morphing [31], paraphrasing [16, 23, 22, 15] etc.). However, data augmentation should be done carefully as it should preserve the original data distribution in order to prevent over-fitting or under-fitting [14]. Additionally, for the task at hand, any data augmentation technique which changes the semantics of the original phrase or imposes substantial obstacles in understanding the meaning, such as antonym replacement as well as extensive random insertions, deletions and swaps, is not acceptable. That is why unlike many applications of data augmentation techniques which simply tend to augment the size of the training dataset, we also need to check the preservation of semantics as a post data augmentation step.

5.3. Method

5.3.1. Dataset

In our application context, LLMs answer users' questions about the functioning of their smart wheelchairs. Users are free to ask anything but should not expect

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Question	Representative of
Can we drive backwards?	A closed-form yes-no question
How often do I need to check if the wheels are securely fastened?	A closed-form question with non-binary answer
How do we stop as soon as possible in case of a problem?	A simple open question
What should I pay attention to?	A more complex open question (answer cannot be found in a single paragraph)
How should I behave in crowded spaces?	A difficult open question
What should I do if the green light still doesn't stop flashing even after 30 minutes?	A question with negation
How to behave in case of any health-related issues, such as skin problems, while using the wheelchair?	A question with appositives
Is it normal that the battery level changes as a bolt from the blue?	A question with idiomatic expressions

Table 3: *Some of the questions in the dataset.*

informative answers when these would require information not provided by the wheelchair manual. We consider state-less interaction, where users ask single questions without follow-ups.

The dataset needs to be representative of possible user questions. In particular, questions should:

1. touch different topics
2. exhibit different complexity wrt answer availability (that is, being answerable using concrete information that can be found in a single paragraph or being answerable using cross-paragraph information or only partially answerable or even unanswerable)
3. exhibit different complexity wrt question formulation (that is, represent different levels of complexity from a linguistic perspective)

According to these criteria, we design a dataset composed of a total of 60 questions; Table 3 lists a few of them.

5.3.2. Data augmentation

From a single question, we generate 10 variations, that cover semantically equivalent ways for users to ask for the same information. For examples, see Table 4.

We use the following methods to generate variations:

- Different **synonym-replacement** approaches: Synonym replacement [19] is a lexical transformation technique in NLP that involves substituting words in a text with their semantically similar counterparts. The objective is to maintain the overall meaning of the sentence while introducing variation in vocabulary. This substitution can be done using different strategies. We restrict ourselves here to approaches relying on lexical databases such as Wordnet [27] to determine synonyms (as done in [42]) or using embedding-based word similarity

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Version	Rephrased Question
v_0	How do we stop as soon as possible in case of a problem?
v_1	How do we stop as shortly as possible in case of a problem?
v_2	In case of a problem, how do we stop?
v_3	How do we stop if there is a problem?

Table 4: *Example of question variations, with v_0 denoting original question.*

([39]) leveraging both embeddings from traditional models (e.g. word2vec, GloVe) and contextual embeddings (e.g. BERT, DistilBERT, RoBERTa).

- **Back-translation**: The back-translation method is a popular technique in natural language processing and machine translation, introduced by Sennrich et al. in 2016 [32]. Back-translation serves as a form of data augmentation, creating synthetic examples by translating source sentences into target languages and then back into the source language. This process results in a larger and more diverse training dataset, helping the model generalize better to various linguistic patterns.
- Different models specifically pre-trained/fine-tuned for **paraphrasing** task: Paraphrasing aims to capture the inherent meaning of a sentence while presenting it using alternative words or structures. In NLP, paraphrasing models are typically either trained using large datasets that contain pairs of sentences with equivalent meanings or the already pre-trained models are additionally fine-tuned for the paraphrasing tasks. We employ both well-known paraphrasing framework Parrot [12] and a specifically fine-tuned version of PEGASUS [41] model for the paraphrasing task.[1]

By augmenting the dataset, we are able to

1. perform a more robust analysis with larger statistical significance;
2. measure the robustness of LLMs when answering semantically equivalent questions.

For every instance obtained using any of the data augmentation techniques, we investigate semantic similarity with respect to the original sample and keep it only in the case of highly preserved semantic similarity (threshold to be determined empirically).

5.3.3. Metrics and Descriptors

For each question in the dataset, we record the following information:

- the question category (see Section 5.3.1)
- the answer from expert humans,
- the location of the relevant sources, identified by such experts,
- the answer of the LLM (for each question variation),
- the location of the most relevant sources (for each variation) as reported by the LLM as described in Section 3.2.1,

[1] https://huggingface.co/tuner007/pegasus_paraphrase

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

- the confidence score reported by the LLMs.

From this information, for each question, we compute the following metrics:

correctness how semantically similar is the answer to the original question with respect to the expected answer;

robustness how similar is the answer to different versions of the original question with respect to the obtained answer (scalar in [0, 1], mean, min, max, and standard deviation over the variations);

faithfulness how similar is the answer to different versions of the original question with respect to the expected answer (scalar in [0, 1], mean, min, max, and standard deviation over the variations);

question-variability robustness how variable is the answer with respect to question variations (scalar in [0, 1]); this is robustness which takes into account question variations;

question-complexity robustness how variable is the answer with respect to question complexity (scalar in [0, 1]);

relevance how relevant is the answer with respect to the actual question variation (scalar in [0, 1], mean, min, max, and standard deviation over the variations);

confidence how confident is the LLMs (scalar in [0, 1], mean and standard deviation over the variations);

explainability how accurate does the reported information sources match the expected sources (scalar in [0, 1], mean error, and error standard deviation over the variations);

explanation variability how variable are the reported information sources over question variations (scalar in [0, 1], standard deviation over the variations);

quality a single score to summarize the quality of an answer — high score when high correctness, high explainability, (scalar in [0, 1], mean, min, max, and standard deviation over the variations).

The mentioned metrics will be calculated on different levels: 1) question level, 2) question class level, 3) data augmentation method and 4) overall for the complete dataset per each of the proposed prompts and eventually other parameters used for configuring ChatGPT (e.g. temperature).

Note that some of these metrics can be used as descriptors at run-time when the ground truth (expected answer and explanation) is not known. We discuss how we make use of them at run-time in Section 5.4.

5.4. Runtime enhancements

In general, engineers, besides *assessing* using the methods presented in Section 5.3, may want to *improve* trustworthiness of LLMs to answer questions in a domain. They could use LLM-specific methods, like tuning parameters and prompts or providing a context with more information, to optimize performance.

We plan to focus instead on a more generic alternative, where we complement the LLM with other ML modules to automatically monitor the quality of an answer and expose this information to the user. This way, we aim to prevent overconfident wrong answers that correlate very negatively with trust. Moreover, by

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

leveraging the methods used in Section 5.3.2 for data augmentation, we show how to increase robustness, picking the best answer to a set of semantically similar questions.

5.4.1. Increased transparency from automatic quality monitoring

Using the same dataset that we used for assessment, we train a regressor (e.g., a multi-layer perceptron) that maps the descriptors computable at run-time (i.e., variability, confidence, and explainability variability) to the quality score. We measure the R^2 score of the regressor using k-fold validation. This will tell us how good these descriptors predict the LLM output. If the score is high enough, we will deploy the regressor to filter out low quality answers and enhance transparency by communicating the current answer quality to the user.

5.4.2. Increased robustness from question augmentations

At run-time, for each user question, instead of a single answer, we generate a set of answers using automatically generated variations of the question. We compute the descriptors and the predicted quality for each answer; we then pick the best answer. We measure how much this strategy improves the quality and the trade-off between a number of variations (i.e. computational cost) and answer quality. We hopefully learn that generating a few variations is enough to improve the [average] answer significantly.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

6. Conclusions

In this document, we summarised the work carried out in task 3.5 to implement the vocal HMI. We started presenting the role of the vocal HMI in the REXASI-PRO process, and then we proceeded giving the details about its hardware and software architecture. The description of the software architecture focused on three aspects: the ROS2 nodes and services created to implement the vocal HMI, the description of the internal state machine that drives the behaviour of the vocal HMI in response to external inputs and how we integrated ChatGPT in order to allow the vocal HMI to answer open questions elaborating the answers based on the information present in the wheelchair manual. In the last part of the document we have described the verification procedures that will be implemented for the vocal HMI in the context of Task T4.5 and finally we have presented the work that has been partially completed and that we will complete in the future to create a framework to verify and improve the explainability and reliability of the use of ChatGPT in a given context.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

7. Annex A

In this annex, we list the functional and non-functional requirements listed in Deliverable D2.4, commenting those whose implementation diverges from the initial description.

Requirement ID	Description	Comment
WC-NFUNC-1	The wheelchair is equipped with a headset with microphone	
WC-NFUNC-3	The wheelchair is equipped with a button to accept the confirmation of an HMI command	
HMI-FUNC-1	When enabled, the Dialogue-based HMI communicates its readiness to received commands	Not implemented, it was slowing down the experience without bringing any improvement
HMI-FUNC-2	The user presses a dedicated button before pronouncing a command	
HMI-FUNC-3	The dialogue-based HMI allows the user to specify the desired destination	
HMI-FUNC-4	The dialogue-based HMI prompts the user for confirmation for each input by the user	
HMI-FUNC-5	Before executing a command given via the vocal HMI, the wheelchair communicates the command it is going to execute	
HMI-FUNC-6	When the wheelchair asks for the confirmation of the understood command the user can reject the command by pressing a rejection button	

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

HMI-FUNC-7	After receiving a command confirmation with the dedicated button, the dialogue-based HMI confirms the execution of the command with an audio message	
HMI-FUNC-8	The users press a button to ask the dialogue-based HMI to start the procedure to create their voice profile	
HMI-FUNC-9	When the voice profile acquisition is finished, the dialogue-based HMI notifies it with a message	
HMI-FUNC-10	The vocal HMI accepts destinations specified by name	
HMI-FUNC-11	The vocal HMI accepts destinations specified by type	
HMI-FUNC-12	The vocal HMI accepts driving style commands	Not yet, but the vocal HMI can be easily extended once the driving style command accepted by the wheelchair controller will be defined
HMI-FUNC-13	The vocal HMI answers questions about the usage of the wheelchair and the location	it only answers question related to the content of the manual of the original wheelchair because we do not have a manual of the wheelchair developed in REXASI-PRO or documentation describing the location
HMI-NFUNC-1	The vocal HMI pipeline executes in no more than 5 seconds (to give an answer and/or send a command)	
HMI-NFUNC-2	The vocal HMI can recognise a command in an environment with a noise of 60db	
HMI-SP-1	The vocal HMI only executes commands of pronounced by a voice that matches the current voice profile	

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

HMI-XAI-1	The vocal HMI notifies the user when a command is pronounced by a voice that does not match the current profile	

Table 5: *Requirements listed in D2.4 and implemented in T3.5*

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

References

[1] clean text. https://github.com/jfilter/clean-text/tree/main.

[2] Hugging Face. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2.

[3] JiWER. https://github.com/jitsi/jiwer. Accessed: 2024-01-25.

[4] langchain. https://www.langchain.com.

[5] Open Text To Speech Server. https://github.com/synesthesiam/opentts. Accessed: 2024-01-25.

[6] OpenAI. https://platform.openai.com/docs/api-reference.

[7] pdfplumber. https://github.com/jsvine/pdfplumber.

[8] Vocal HMI GIT repository. https://git.hb.dfki.de/rexasi-pro-eu/ros2/hmi.

[9] Anastasios N Angelopoulos and Stephen Bates. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. *arXiv preprint arXiv:2107.07511*, 2021.

[10] Anastasios N Angelopoulos, Stephen Bates, Adam Fisch, Lihua Lei, and Tal Schuster. Conformal risk control. *arXiv preprint arXiv:2208.02814*, 2022.

[11] Tom Bocklisch, Joey Faulkner, Nick Pawlowski, and Alan Nichol. Rasa: Open source language understanding and dialogue management, 2017.

[12] Prithiviraj Damodaran. Parrot: Paraphrase generation for nlu., 2021.

[13] Fares Ernez, Alexandre Arnold, Audrey Galametz, Catherine Kobus, and Nawal Ould-Amer. Applying the conformal prediction paradigm for the uncertainty quantification of an end-to-end automatic speech recognition model (wav2vec 2.0). In *Conformal and Probabilistic Prediction with Applications*, pages 16–35. PMLR, 2023.

[14] Steven Y Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy. A survey of data augmentation approaches for nlp. *arXiv preprint arXiv:2105.03075*, 2021.

[15] Varun Gangal, Steven Y Feng, Malihe Alikhani, Teruko Mitamura, and Eduard Hovy. Nareor: The narrative reordering problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10645–10653, 2022.

[16] Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. Adversarial example generation with syntactically controlled paraphrase networks. *arXiv preprint arXiv:1804.06059*, 2018.

[17] Myeongjun Jang and Thomas Lukasiewicz. Consistency analysis of chatgpt. *arXiv preprint arXiv:2303.06273*, 2023.

[18] Douglas Johnson, Rachel Goodman, J Patrinely, Cosby Stone, Eli Zimmerman, Rebecca Donald, Sam Chang, Sean Berkowitz, Avni Finn, Eiman Jahangir, et al. Assessing the accuracy and reliability of ai-generated medical responses: an evaluation of the chat-gpt model. *Research square*, 2023.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

[19] Michał Jungiewicz and Aleksander Smywiński-Pohl. Towards textual data augmentation for neural networks: synonyms and maximum loss. *Computer Science*, 20, 2019.

[20] Aisha Khatun and Daniel G Brown. Reliability check: An analysis of gpt-3's response to sensitive topics and prompt wording. *arXiv preprint arXiv:2306.06199*, 2023.

[21] Sosuke Kobayashi. Contextual augmentation: Data augmentation by words with paradigmatic relations. *arXiv preprint arXiv:1805.06201*, 2018.

[22] Ashutosh Kumar, Kabir Ahuja, Raghuram Vadapalli, and Partha Talukdar. Syntax-guided controlled generation of paraphrases. *Transactions of the Association for Computational Linguistics*, 8:330–345, 2020.

[23] Ashutosh Kumar, Satwik Bhattamishra, Manik Bhandari, and Partha Talukdar. Submodular optimization-based diverse paraphrasing and its effectiveness in data augmentation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3609–3619, 2019.

[24] Jian Li, Li Wang, Xi Chen, XiangWen Deng, Hao Wen, Mingke You, and Weizhi Liu. Are you asking gpt-4 medical questions properly?-prompt engineering in consistency and reliability with evidence-based guidelines for chatgpt-4: A pilot study. 2023.

[25] Yang Liu, Yuanshun Yao, Jean-Francois Ton, Xiaoying Zhang, Ruocheng Guo Hao Cheng, Yegor Klochkov, Muhammad Faaiz Taufiq, and Hang Li. Trustworthy llms: a survey and guideline for evaluating large language models' alignment. *arXiv preprint arXiv:2308.05374*, 2023.

[26] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.

[27] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[28] OpenAI. Chatgpt. `https://chat.openai.com/auth/login`.

[29] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision, 2022.

[30] Mirco Ravanelli, Titouan Parcollet, Peter Plantinga, Aku Rouhe, Samuele Cornell, Loren Lugosch, Cem Subakan, Nauman Dawalatabad, Abdelwahab Heba, Jianyuan Zhong, Ju-Chieh Chou, Sung-Lin Yeh, Szu-Wei Fu, Chien-Feng Liao, Elena Rastorgueva, François Grondin, William Aris, Hwidong Na, Yan Gao, Renato De Mori, and Yoshua Bengio. SpeechBrain: A general-purpose speech toolkit, 2021. arXiv:2106.04624.

[31] Gözde Gül Şahin and Mark Steedman. Data augmentation via dependency tree morphing for low-resource languages. *arXiv preprint arXiv:1903.09460*, 2019.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

[32] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. *arXiv preprint arXiv:1511.06709*, 2015.

[33] Xinyue Shen, Zeyuan Chen, Michael Backes, and Yang Zhang. In chatgpt we trust? measuring and characterizing the reliability of chatgpt. *arXiv preprint arXiv:2304.08979*, 2023.

[34] Chenglei Si, Zhe Gan, Zhengyuan Yang, Shuohang Wang, Jianfeng Wang, Jordan Boyd-Graber, and Lijuan Wang. Prompting gpt-3 to be reliable. *arXiv preprint arXiv:2210.09150*, 2022.

[35] Andrew Silva, Mariah Schrum, Erin Hedlund-Botti, Nakul Gopalan, and Matthew Gombolay. Explainable artificial intelligence: Evaluating the objective and subjective impacts of xai on human-agent interaction. *International Journal of Human–Computer Interaction*, 39(7):1390–1404, 2023.

[36] Ana Suárez, Víctor Díaz-Flores García, Juan Algar, Margarita Gómez Sánchez, María Llorente de Pedro, and Yolanda Freire. Unveiling the chatgpt phenomenon: evaluating the consistency and accuracy of endodontic question answers. *International endodontic journal*, 57(1):108–113, 2024.

[37] Ryan J Tibshirani, Rina Foygel Barber, Emmanuel Candes, and Aaditya Ramdas. Conformal prediction under covariate shift. *Advances in neural information processing systems*, 32, 2019.

[38] Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic learning in a random world*, volume 29. Springer, 2005.

[39] William Yang Wang and Diyi Yang. That's so annoying!!!: A lexical and frame-semantic embedding based data augmentation approach to automatic categorization of annoying behaviors using# petpeeve tweets. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 2557–2563, 2015.

[40] Jason Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*, 2019.

[41] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization, 2019.

[42] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

[43] Li Zhong and Zilong Wang. A study on robustness and reliability of large language model code generation. *arXiv preprint arXiv:2308.10335*, 2023.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**