Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Consiglio Nazionale delle Ricerche

ISTI Technical Reports

Janet: a prototype of a conversational agent for a virtual research environment

Ahmed Salah Tawfik Ibrahim, ISTI-CNR, Pisa, Italy
Leonardo Candela, ISTI-CNR, Pisa, Italy

ISTI-TR-2023/006

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Consiglio Nazionale delle Ricerche

Janet: a prototype of a conversational agent for a virtual research environment

Ahmed Salah Tawfik Ibrahim, Leonardo Candela

ISTI-TR-2023/006

Conversational agents have been undergoing a lot of development where they have reached their best nowadays thanks to the large language models that have been developed over the past years. Thanks to this, the capabilities of such agents have become more diverse and they are now able to perform a wide range of tasks which can be harnessed to reduce or even replace the human effort in performing those tasks. Therefore, in an effort to equip scientific communities with such a powerful tool, we developed a prototype conversational agent, Janet, and deployed it into one virtual research environment (VRE) with the aim of supporting the scientists in exploiting the knowledge contained within these environments. Janet is still in the preliminary phase where there is a need for extensive testing and improvement; however, this report shows the main aspects of Janet including the architecture, implementation and deployment.

Keywords: Virtual Assistant, Virtual Research Environment, AI Chatbot, Conversational Agent.

Citation
Ibrahim A.S.T., Candela L. *Janet: a prototype of a conversational agent for a virtual research environment*. ISTI Technical Reports 2023/006. DOI: 10.32079/ISTI-TR-2023/006.

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Area della Ricerca CNR di Pisa
Via G. Moruzzi 1
56124 Pisa Italy
http://www.isti.cnr.it

Janet: a Prototype of a Conversational Agent for a Virtual Research Environment

Ahmed Salah Tawfik Ibrahim, Leonardo Candela*

Abstract

Conversational agents have been undergoing a lot of development where they have reached their best nowadays thanks to the large language models that have been developed over the past years. Thanks to this, the capabilities of such agents have become more diverse and they are now able to perform a wide range of tasks which can be harnessed to reduce or even replace the human effort in performing those tasks. Therefore, in an effort to equip scientific communities with such a powerful tool, we developed a prototype conversational agent, Janet, and deployed it into one virtual research environment (VRE) with the aim of supporting the scientists in exploiting the knowledge contained within these environments. Janet is still in the preliminary phase where there is a need for extensive testing and improvement; however, this report shows the main aspects of Janet including the architecture, implementation and deployment.

Keywords

Virtual Assistant — Virtual Research Environment — AI Chatbot — Conversational Agent — VRE

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", Consiglio Nazionale delle Ricerche, Via G. Moruzzi 1, 56124, Pisa, Italy
*__Corresponding authors__: leonardo.candela@isti.cnr.it

Disclaimer

This report contains selected excerpts from the thesis project [8] it is based on, which was authored by the same authors of this report.

Contents

1. Introduction

Conversational agents, also referred to as virtual assistants or chatbots, are now a default service offered by most websites and companies. They are used to fulfill a wide range of needs depending on the sector where they are deployed. For example, banks use them on their websites in order to assist customers in selecting the products that are suitable for them. The capabilities of these conversational agents are always improving thanks to the recent advances in the field of natural language processing (NLP) which aims at making machines able to understand and/or generate human language. In this work, we explore the possibility of implementing a conversational agent that could be helpful to the users of science gateways; in particular, the D4Science gateway and its virtual research environments (VREs) [3].

Generally, the users of these VREs are researchers or scientists and they are usually working with research artifacts like papers and datasets. They work collaboratively on their research of interest while exploiting the resources of the VRE which include datasets, papers, a social network and computing resources. Therefore, it is essential to explore the possibility of providing a conversational assistant that can make it easier for a VRE user to navigate through the content of the VRE. In other words, this work is motivated by the need to have a way to assist scientists while reducing the human effort involved.

This prototype represents a first step in a larger project that aims at using artificial intelligence in supporting VRE users. As such, we focus on an initial set of tasks that the agent is going to be able to perform. More specifically, the core outcome of this project is a conversational agent, Janet[1], that is capable of:

1. Answering questions based on the VRE content;
2. Summarizing papers shared by VRE co-workers;
3. Retrieving papers, datasets and posts upon the user's request;
4. Recommending papers and datasets to users based on their interests;

[1]The name was inspired by a fictional character, a know-it-all robot, from The Good Place comedy.

5. Chatting with users in a meaningful way;
6. Responding to certain commands that aid the user in exploring the tool and the environment.

To this aim, we develop a set of finetuned models that are capable of doing general tasks that are needed to achieve this initial set of capabilities. These models include an intent classifier, an entity extractor, an ambiguous query classifier, an offensive language classifier, a neural retriever and a set of language generators. In addition, we preprocess some publicly available datasets and we had to create others manually in order to finetune these models. and finally, we report all our findings and provide directions for the future steps that can be followed to improve our conversational agent.

This report is organized as follows: Sec. 2 describes the overall system architecture; Sec. 3 describes the implementation of Janet; Sec. 4 describes the deployment of Janet into one VRE; Finally, Sec. 5 concludes the report and proposes a set of future enhancements.

2. The Architecture

In this section, we discuss the overall system architecture of Janet describing its main components and their functionalities. Janet is designed with modularity, self-improvement and context-awareness in mind. This means that it is built to be aware of the contents of the specific VRE it is serving and is capable of correcting and improving its behavior with time. It is made up of two main components: a master and a worker.

The master is responsible for training the models used to reply to the user. It is also responsible for managing the collected feedback data that will be used to run a learning algorithm in order to improve those models.

The worker, however, runs at the level of the VRE; thus, it is responsible for managing the contents of that VRE. It has been inspired by the general architecture (Figure 1) described in [1] which consists of multiple components; mainly, it must include an interface to get the input via text or speech, a message analyzer to understand the input, a dialog manager, a knowledge layer to find the answer to the query and finally a response generator. Thus, the Janet worker contains a natural

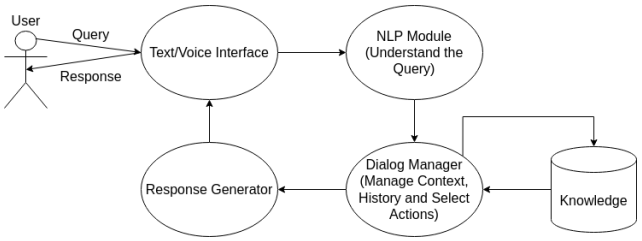

Figure 1. General Architecture of Conversational Agents

language understanding (NLU) module that is responsible for extracting the intents and entities of a given query while employing a coreference resolution mechanism as well as a query rewriter that takes into account the dialog history. It also contains an offensive language classifier as well as an ambiguous query classifier.

The dialog history is managed by the second component which is the dialog manager which keeps track of past conversations in order to maintain the conversational context. The dialog manager also decides the next action to take by selecting the appropriate response generation mechanism depending on the current state.

In addition to these two components, a response generation module makes use of the current models inside the master in order to form an answer. Mainly, it contains a retriever to fetch relevant content from the VRE, a number of language generators and a recommender to suggest content to the users based on their interests.

Finally, there is an overall worker manager which takes care of collecting feedback from the user in order to build the dataset that will be used by the master to update the models. It is also responsible for indexing the content of the VRE in a way that makes it usable by the models and collecting information about the user's interests. In other words, it is responsible for managing the knowledge of the worker.

Figure 2 summarizes the overall architecture of the described system.

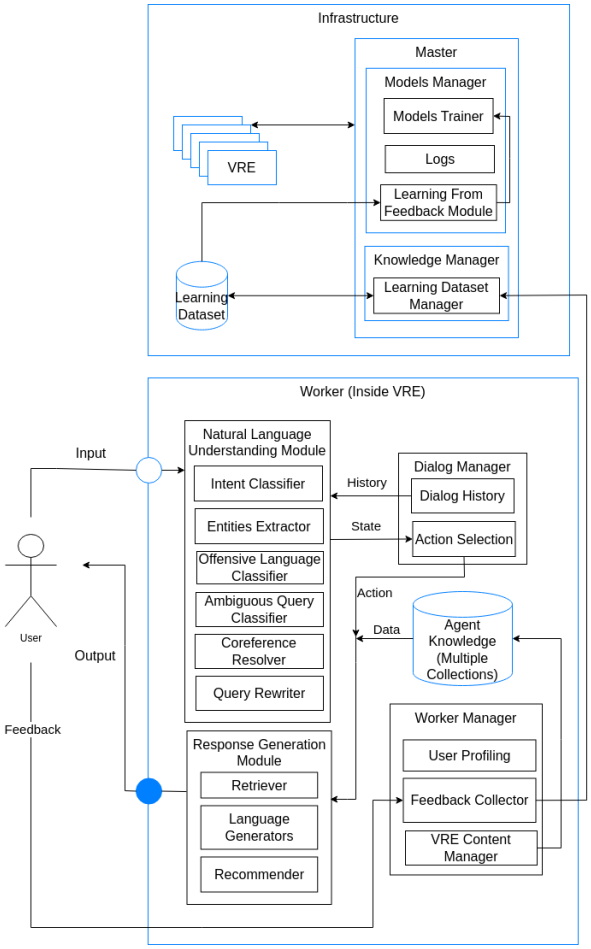

Figure 2. System Architecture

This design is chosen as it satisfies the requirements of the problem at hand, which is designing a conversational agent for virtual research environments which is capable of assisting the users with their tasks. To this aim, the agent should be open-domain since the tasks are not defined apriori within a VRE; therefore, the agent should be informative and conversational rather than task-oriented. In doing so, and following the CIR system paradigm explained by [7] and shown in Figure 3, the agent will have general purpose components that can be used to achieve a wide range of tasks, starting with the initial set described in Sec. 1.

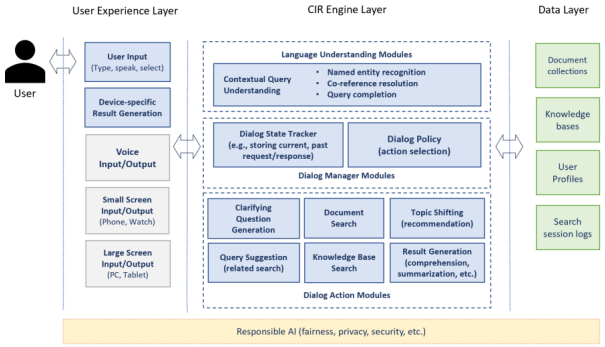

Figure 3. CIR System General Architecture by Gao et al. [7]

In addition, the response generation should be intelligent and so it should utilize advanced AI techniques for either the retrieval and/or the generation of an answer. Finally, following the state of the art, the agent should follow the general modular architecture described previously.

3. The Implementation

3.1 Natural Language Understanding (NLU) Module

This component of the Janet worker is responsible for the first stage of the pipeline processing the user's input. It consists of a number of sub-modules that are used to understand the input of the user.

The first step is to resolve the coreferences in the user's input using the conversation history. This is achieved by means of a pre-trained neural coreference resolver[2]. It works by using deep learning to cluster tokens that refer to the same entity together in one cluster which can then be used to replace the coreferences with the original mention.

After the coreference resolution, it extracts the intent of the modified query and checks the confidence score.

If the score is above a certain threshold, which was set by us to be 50%, it extracts the entities and checks if the query is offensive in order to flag it as offensive if this is the case. If the entities set is empty and the intent requires the presence of entities, the query is flagged as unclear. Otherwise, the query is considered clear and is passed to the dialog manager.

[2]https://github.com/explosion/projects/tree/v3/experimental/coref

If the confidence score of the intent is below the threshold, however, the query is passed to the ambiguous query classifier in order to decide if it is ambiguous.

If it was found ambiguous, an attempt to rewrite the query by a pre-trained neural rewriter is performed. Then, the intent, entities and offensive flag are computed. If the intent score is still below the threshold or if the ambiguous classifier still believes the query is ambiguous, or if the entities set of certain intents is empty, the query is considered ambiguous and is flagged as such. Otherwise, the query is clear and gets passed to the dialog manager.

If the ambiguous query classifier flags the query as unambiguous, however, then the entities set is extracted and if it's empty for certain intents, the query is flagged as ambiguous. If the query passes this final test, it is considered clear. The flow chart in Figure 4 summarizes this process, where the solid line represents the flow if the query does not go through the query rewriting step whereas the dashed line shows the flow when the query is rewritten.

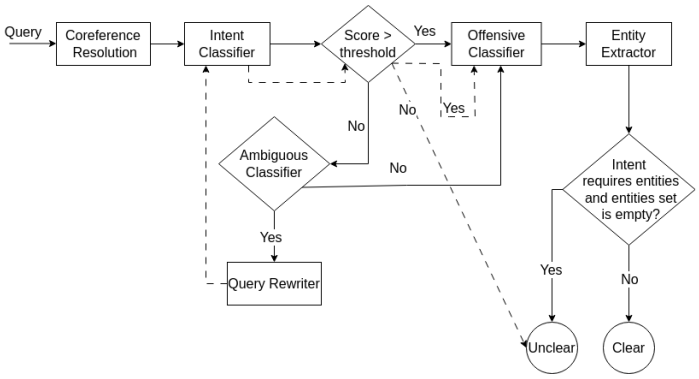

Figure 4. Natural Language Processing Module Flowchart

In conclusion, the NLU module contains 4 classifiers (*i*) the intent classifier, (*ii*) the entities extractor, (*iii*) the ambiguous query classifier, and, (*iv*) the offensive language classifier. It also includes a one sequence to sequence model that is responsible for rewriting the query, when needed, considering the conversation history in an effort to disambiguate an ambiguous query.

Regarding the intent classifier, the term intent describes the purpose or the aim of a given query. That is why it is an essential component in the natural language understanding pipeline. In the initial version of the conversational agent, we support a set of general purpose intents that will be used by the dialog manager to choose the appropriate response generation mechanism. It is, however, important to clarify that the term intent does not refer to tasks like in traditional conversational agents. Rather, it is just a way to let the dialog manager decide which components to use for generating the response.

The intents supported for the initial version are an expansion of the goals described in Section 1. They include: (*a*) question-answering (QA), (*b*) chitchatting, (*c*) retrieving a cat-

alogue item (papers or datasets) and posts, (*d*) summarizing a paper, (*e*) affirmation, (*f*) negation, (*g*) listing catalogue items (papers and datasets) and VRE topics, (*h*) asking for help

To achieve the highest accuracy, and to allow the component to be maintain its performance when it is retrained after adding new intents in the future, a BERT-based model was chosen to implement the intent classifier. In particular, the distilled version of BERT was chosen as it has computational benefits while maintaining a high accuracy for classification tasks [16]. The dataset used to train this model was curated manually. It contains examples for each of the mentioned intents. Those examples are in the form of <text, intent> which were used in order to fine-tune the distilled BERT with a classification head attached to it. In total, the dataset contains 275 sentences that are classified into one of 14 intents that more or less cover the goals described above. For instance, we have sentences like "what can you do?", "can you help me?" labeled with the intent corresponding to asking for help. Another example is "get me a paper about neural networks" which is labeled as question-answering. A final example is "what's your name?" which is labeled as chitchatting as it corresponds to small talk with the agent.

The entities extractor, however, is a supplementary component to the intent classifier because some of the intents cannot be resolved directly by the response generator without having a deeper understanding about what they are about. More specifically, for the intents related to finding a resource or summarizing a paper, it is probably more helpful to extract more information from the query. In particular, the information that was considered for this initial version include the topic of interest, the type of the resource, the title of the resource, the author of the resource and the exact date of publication. These are important in the process of identifying those resources in order to execute the user's queries on those resources. Therefore, we manually curated a dataset whose examples are on the form of <text, entities> where entities is a list of tuples containing an entity label and the start and end indices of the character span of that entity in the text. This dataset contains 97 tuples where each tuple is a sentence plus the entities within it. For instance, a sentence like "get me a paper about reinforcement learning", will have the index span corresponding to "reinforcement learning" labeled as topic. Whereas the sentence "summarize a paper about neural networks authored by OpenAI" will have the index span corresponding to "neural networks" as topic and the index span corresponding to "OpenAI" as author.

Similar to the intent classifier, the entity extractor is also a transformer model, RoBERTa, which is based on BERT and is the base of the state of the art token classifiers [11].

As for the ambiguous query classifier, we made use of the public dataset ConvAI3 [2] which contains queries that are labeled based on their degree of ambiguity from 1 to 4, where 1 representing a clear query and 4 representing an ambiguous one. So, the dataset is composed of examples on the form <query, ambiguity degree>. A suitable model for this task was again the distilled BERT similar to the intent classifier.

Regarding the offensive language classifier, it is a binary classifier that would determine if a query is inappropriate. This is important in our context because our conversational agent is designed with self-improvement in mind and as it will improve itself in the future using the user's queries and user's feedback. Thus, it is important to have this component in order to keep the responses appropriate when we are using the user's feedback in the future. We used the Hate Speech and Offensive Language Dataset [5] which is publicly available on kaggle.com. It contains tweets in English that have been labeled via crowd-sourcing as neutral, offensive or hateful. The labeling was done after getting the judgement of a number of labelers and then performing a majority voting to determine the final label of each tweet. Hence, the dataset is made up of examples on the form <text, label>. Finally, just like the intent classifier, the distilled BERT model was used to develop this component.

The query rewriter model is a generative sequence-to-sequence model whose purpose is to take as input an ambiguous query with possible references to past utterances and to output a history-independent query that can be used as a stand-alone query. In other words, its main job is to remove the dependence of the query on the history of the conversation. In order to achieve this goal, we made use of an existing T5 [15] model[3] that was finetuned on the publicly available CANARD dataset [6] which is a dataset containing examples on the form <query, history, modified query> where history is an ordered list of previous utterances in the conversation. The creators of the model modified the dataset in order to change the history into a string where a special token is used as a separator between each utterance and the next. Then the query was appended in the end along with the separator in order to have the query and the history in one text sequence. This is because they made use of the transformer model, T5 , which is a text2text model that takes as input a text sequence and outputs a target text sequence. In the end, they used the pre-processed dataset whose examples are now on the form of <text, target text>, where text is the history along with the potentially ambiguous history-dependent query and target text is the modified history-independent query, in order to finetune the T5 model. We chose to make use of this publicly available model as training generative models proved to be very time consuming and also because it did not make sense to repeat the training process they did on the same dataset, which is the only existing benchmark for query rewriting as far as we know.

3.2 Dialog Manager Module

The dialog manager is simply the working memory of the whole system. It contains two data structures; the current state and the chat history.

The current state is simply the output of the NLU component in addition to two more flags; one to determine if it is

[3] https://huggingface.co/castorini/t5-base-canard

the start of the conversation and another to determine if the user is inactive. The output of the NLU is nothing but the modified query, the intent of the query, the extracted entities, the offensive flag and the ambiguous flag.

The chat history, however, is nothing but a list of the most recent n exchanges, where one exchange includes the user's prompt and the agent's response to the prompt. In order to make it easier to manipulate by the NLU, it is stored in two modes; a concatenated string mode with no separators and another one with a separator. The former is useful for the coreference resolution while the latter is useful for the query rewriter.

For the purpose of this project, and since the task space is not huge, it was sufficient to implement it as a finite state machine where it takes actions based on the current state.

The following finite-state machine in Figure 5 describes the way the dialog manager selects its next action. The states are represented by circles while the actions are represented by squares.

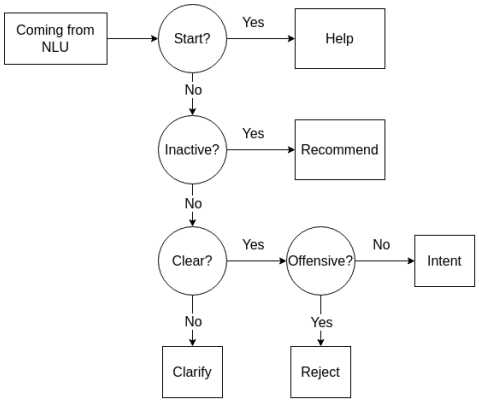

Figure 5. Dialog Manager Finite-State Machine

3.3 Response Generation Module

This module is the one responsible for providing an answer to the user's query. In order to achieve this, it has access to all the knowledge contained within the VRE which is provided by the worker manager module which is explained in section 3.4. Moreover, it contains three vital models which together compose the core of this component; i.e., the content retriever, the language generator and the recommender.

The retriever represents the knowledge of the agent. This is because it is able to retrieve the most relevant content from the content index which can be further exploited by other components in order to generate the answer. It is based on dense information retrieval as it basically transforms the content into dense vectors which can later be indexed for fast retrieval via similarity metrics like distance or cosine similarity. In order to have the best performance, a sentence transformer was used to model this component. The one we used was the mpnet-base sentence transformer which maps paragraphs into dense 768-dimensional vectors as shown in figure 6: In

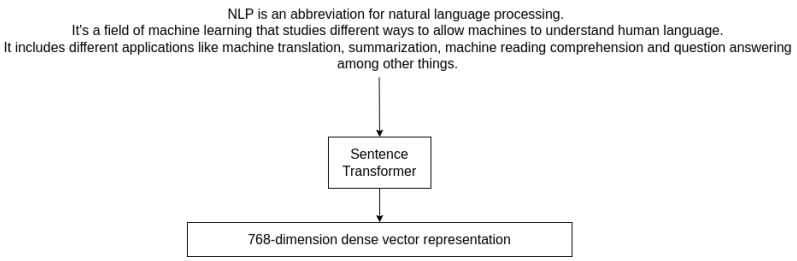

Figure 6. Sentence Transformer

order to train this model, we used different sources to create a general purpose dataset. We manually came up with questions about articles in the user manual of D4Science. Then, we augmented them with question-context pairs extracted from the publicly available MS-Marco v2.1 dataset [14], PUBMED QA dataset [9], QASPER dataset [4], and ScienceQA dataset [12]. The sentence transformer was finetuned used this large combined dataset whose examples are on the form of <query, context>.

In general, this retriever is used to compute vector representations of the content inside the VRE. These vectors are then indexed via FAISS [10] for fast retrieval as shown in figure 7. Whenever a query is given to the retriever, it would

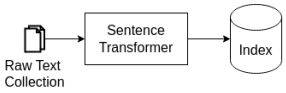

Figure 7. Indexing Documents

encode it; i.e., transform it into its vector representation and then, using the index, it would quickly find the most relevant content that could possibly answer it.

The language generator, however, is responsible for any sequence-to-sequence operations that the agent can perform. This includes QA, summarization, chitchat and clarifying question generation. The way it works is that it takes a query, appends some information to it that would help answer it in a good way then generate an answer. The current state of the art sequence-to-sequence models are again transformers and we chose to use the T5 model [15] for experimental purposes. This is because it implements an encoder-decoder architecture which is a desirable model architecture in order to implement the retrieval-augmented language generation when needed. Furthermore, we wanted to see if we can have just one generator for all the sequence-to-sequence operations we support or if we need to have one for each operation. T5 provides the possibility of performing multi-task training by simply appending a task prefix to each training example. For example, a QA training example can be on the form <question: text context: text, answer> where question: and context: are QA-specific prefixes. So, we wanted to conduct experiments to see if we can have one model for all our operations or if it is better to have one for each. To this end, we collected different training data for each of our operations and appended them

with appropriate task prefixes.

For the QA, we had the same dataset used for the retriever except that we appended the questions to the contexts in one string and we had the answer as our target.

For the summarization operation, we made use of the OpenAI Summarize From Feedback dataset [17] and the XSum dataset [13].

Finally, for the chitchat operation, we used the PersonaChat dataset [18] which includes conversations between humans where one of them has to maintain a certain personality that is represented by a sequence of sentences.

In the end, we ended up with 3 specific datasets that we then concatenated to generate a bigger multi-purpose dataset where each example is on the form <text, target text> and text is simply the query augmented with useful data and task specific prefixes.

In addition, we used a separate dataset for the clarifying question generator. Namely, we used ConvAI3 [2] which contained a number of ambiguous queries along side the appropriate clarifying question. We removed the queries that had an ambiguity score of 1 and 2 and we only considered those with scores 3 and 4. So, we had a dataset whose training examples are on the form <ambiguous query, clarifying question>.

The recommender component is made up of a list of candidate recommendations that can be given to the user. This list is updated whenever new material is added to the VRE or whenever new user interests are detected. It basically makes use of the retriever to encode the user's interests and then it encodes the tags associated with the VRE material also using the retriever. Then, it computes the cosine similarity between these two representations and if it is above a certain threshold, then the material gets added to the list of current recommendations. Then, when making a recommendation, it picks at random one of the generated recommendations and marks it as recommended, to avoid recommending it again, and it forms a recommendation sentence based on the metadata of the recommended material.

As explained previously, depending on the current dialog state, the dialog manager selects a certain mode to let the response generator know how to generate the response. We support 9 different modes of operation: help upon start, recommendation, rejecting offensive queries, question-answering, chitchat, retrieving a resource (paper or dataset) or a post, summarization, asking for help, listing resources, and clarification. The help upon start, the listing of resources, the asking for help and the rejection of offensive queries are trivial modes as they return a fixed response depending on the case. The recommendation mode is also trivial as it just emits the recommendation generated by the recommender which was explained previously.

The question-answering response generation makes use of both the retriever and the generator. First, the processed query of the user gets encoded by the retriever in order to get its vector representation. Then, the indexed document collection is searched to find the most relevant text which then gets appended to the query. Finally, the concatenated sequence containing the question and the context is given to the generator which then generates the answer.

Regarding the pipeline of chitchatting, it is pretty simple. It makes use of a generator; however, the query gets a little bit modified. Basically, in order to add personality to the replies, and to enable the agent to respond to questions about itself, we append a couple of sentences that describe the agent, preceded by the special token <persona>, to the query. Then, we return the generator's response.

In order to support the functionality of finding a resource and potentially operating on it, we had to keep a state variable to hold the metadata of the resource of interest. Then, using the extracted information from the query; i.e., the entities regarding mainly titles and topics, we performed a similarity search. This similarity search was done on the titles versus the title extracted from the query, and the resource with the title that matches the query title the most is returned if the cosine similarity score is above a certain threshold. In order to compute this cosine similarity, the retriever was used to encode the titles and the query title into vectors and then the scoring was done normally. The same pipeline was applied in the case of topics as well. If no resource was found after performing these steps, then we perform an index search using the entire query by encoding it into a vector using the retriever and comparing it to the vector encodings of the descriptions of all the resources, or the text of all posts in the case of post retrieval. The most recently obtained paper, dataset or post are kept in a data structure in order to allow the user to reference them in the future without having to specify them in detail.

The clarification mode, however, consists of two different modes of reply depending on the intents and entities. If the intent is related to a resource i.e., finding a paper, a dataset or summarizing a paper, and the entities set is empty and the data structure keeping the most recently obtained resource is empty, then the user is prompted to repose their query specifying some attribute about their resource of interest, like a title or a topic. If that is not the case, then a clarifying question is generated using the clarifying question generator and is sent to the user in order to try to understand their needs.

Finally, the pipeline of summarization is a mix of the preceding two tasks. Basically, in order to summarize a paper, it has to be found. This can be done in both ways: either the paper has been previously mentioned by the user and is stored in the state variable that we described previously, or it is mentioned for the first time. In the second case, we perform the pipeline of finding a resource in order to get the paper, giving priority to the title over the topic, then we extract the text and then pass it to a language generator to generate the summary and return it to the user. In the first case, we just extract the text from the already obtained paper and summarize it using the language generator and return the summary.

3.4 Worker Manager

This component is responsible for three subtasks; modelling the user, fetching the VRE content and collecting feedback.

The user subcomponent is responsible for keeping track of the user's interests over time. It basically populates a database with topics extracted from user queries and assigns an interest value to them. Initially, when a new interest is added, it gets a high value. However, these interest values decay over time if they do not occur often in the user's queries. In other words, the user's queries provide insight about their interests which are then added with high interest value to the database and this value increases with each occurrence of that specific interest or decreases gradually over time when they do not appear. These interests are used in order to provide recommendations to the user based on their interests. In order to make it practical, when making recommendations, the system only considers n interests which have the highest interest value.

The VRE content fetcher is responsible for organizing the contents; i.e., research papers and datasets, along with their metadata from the VRE. It then populates different databases and index structures with this fetched content. Basically, it creates and maintains and a database of papers metadata, a database of datasets metadata and a database of raw text content from the VRE papers and other sources like the social network. It also creates and maintains different index structures. This includes an index for the titles of the datasets, an index for the titles of the papers, an index for the description of the datasets, and index for the description of the papers and an index for the raw content of the VRE. These different indices are populated using the dense representation computed using the retriever described previously in order to make it faster to search over the contents of the VRE. So, in other words, the index contains the vector representation of the content while the database contains the actual content. In order to stay up-to-date, this component periodically performs a query to the VRE in order to fetch newly added content, if any, and then add it to the current index structure. If the amount of new content will increase the number of indexed documents beyond a certain threshold, the index will be trained from scratch in order to have a more efficient storage to allow for fast retrieval.

Finally, the feedback of the user is collected by a feedback collector which then populates a database of feedback that can be used to enhance the models. Basically, after the agent provides an answer, the system would ask the user a set of questions whose answers would build up a dataset that can be used to retrain and enhance the retriever and the generator. It can help in developing a reinforcement learning from human feedback pipeline which is one of the future works that we recommend in the end of this work.

4. The Deployment

In order to deploy Janet on one of the VREs, it had to be containerized. Therefore, we made use of Docker in order to achieve this goal. We created two main containers; one for the front-end and one for the back-end. Furthermore, there was an additional container for a POSTGRESQL database that is used to collect the feedback. It is worth noting that the deployment of the three containers was orchestrated via docker-compose.

The back-end service includes all the modules that compose the Janet worker which were all developed in python. Furthermore, we developed it as a REST API using flask and python. It has two main post methods; one to post a user query to agent and the other to post the user feedback. In addition to the main methods, there is a get method to check the health of the service, and another post method to establish the connection and the user credentials with the frontend. In the beginning, the frontend communicates the user authentication token to the backend, which then sends a signal to the frontend that it is ready to receive user queries. Upon receiving a query, it gets passed through the components in order to generate a response and send it back. It gets analyzed by the NLU module, then some user interests are extracted from it depending on the entities of the query. Then, the dialog manager updates the history and chooses the next action which is then passed to the response generator which in turn generates the response and returns it to be sent to the user. Upon sending the response, the user is supposed to evaluate the response and that evaluation is in turn sent back to the backend which communicates with the database service to store that evaluation.

As for the frontend service, it was developed using React. Basically, it implements a chat icon that appears at the bottom of the page which, upon clicking it, renders a chat box where the conversational interaction between the user and the agent is displayed. Once the user enters the web page hosting this service, and after the token authentication step with the backend is complete, the frontend sends an artificial query to the backend to display a help message to the user which would inform them more about the tool and its functionalities. The user can then submit queries and get replies through the frontend from the backend. When the user receives back a reply, a feedback form is displayed to the user where they can evaluate their satisfaction with the response they got. This feedback form is also used to collect data which can be used in the future to enhance the models. Upon submitting the form, it disappears from the page and the answers get sent to the backend which in turn stores them into a database.

Janet, was hosted on a dedicated web page that was used as an iFrame inside one of D4Science VREs; namely, a VRE titled AssistedLab.

Figures 8 and 9 show the integration of Janet into the AssistedLab VRE.

After deploying the system, we let users interact with it to test if the overall system was working; however, the testing period was short due to time constraints; therefore, the system was not thoroughly tested. Nonetheless, the testing allowed us to highlight some of the strengths and weakness points of Janet. Namely, our attention was drawn to the poor perfor-

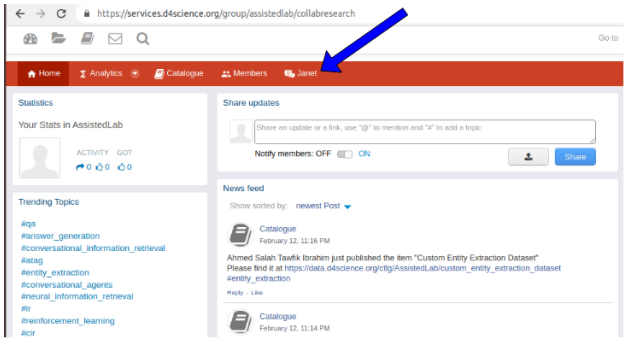

Figure 8. Janet IFrame in AssistedLab VRE

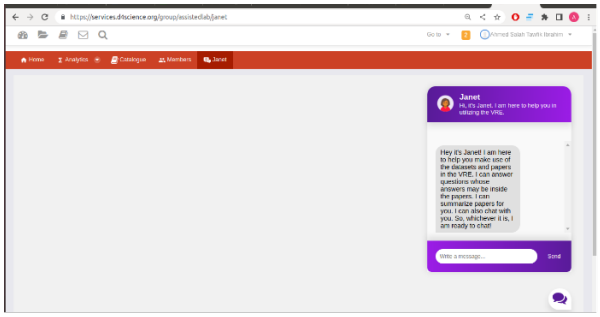

Figure 9. Janet in Operation inside the VRE

mance of the generative models and the strong performance of the retriever when the VRE collection contains related content to the query as shown in Fig. 11 and Fig. 10.

Figure 10. Response Usefulness Evaluation

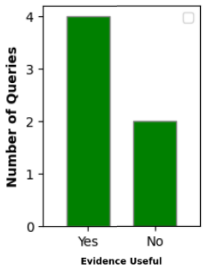

Figure 11. Usefulness of Retrieved Evidence for QA

5. Conclusion and Future Works

In conclusion, we have shown in this report the main design and architectural decisions that we took in implementing Janet as the main activity of the thesis project of the author [8]. We have highlighted the main results we obtained after deploying it and we acknowledge the fact that more testing is needed to have more solid results. Nonetheless, from the preliminary testing, we obtained some insight as to how to proceed in the near future in order to enhance Janet.

In particular, we noted that due to the limited size of the training set of the intent classifier (275 examples), misclassifications may happen if the query is very different from the dataset. We also noted that the retriever would perform accurately in most of the cases when there is relevant content to the query that can be retrieved from the collection of content. A possible enhancement is to equip the retriever with the capability of looking outside the VRE for relevant content in order to not limit itself to the knowledge contained within the VRE. We also observed how poorly the generators performed due to the fact that their training was very time-consuming and we had a limited time window to complete the project. In fact, we hypothesize that another reason why we were unable to build satisfactory generators is that we had to rely on publicly available general purpose datasets which could be a limitation as the agent is deployed into a scientific environment. Therefore, we recommend as future enhancement, to make use of the facilities already implemented for collecting feedback in order to utilize reinforcement learning from feedback which has been proven to dramatically enhance the performance of generative models.

Furthermore, enhancements should be made to the natural language understanding module by retraining the intent classifier with new data and equipping it with more intents. A particularly interesting enhancement is to allow the intent classifier to detect multiple intents in the query in order to enhance the understanding capabilities of the model. Moreover, the entity extractor can be developed further to support useful intents like dates (absolute and relative) and numbers. Finally, the query rewriter should be also enhanced using the feedback collected from the users when interacting with the system.

Software availability

The software described in this report is available at Janet-Backend[4] and Janet-Frontend[5]

Acknowledgments

This work received funding from the European Union's Horizon 2020 research and innovation programme under: Blue Cloud project (grant agreement No. 862409).

[4] https://code-repo.d4science.org/ahmed.ibrahim39699/JanetBackEnd
[5] https://code-repo.d4science.org/ahmed.ibrahim39699/JanetFrontEnd

Author contributions

According to CRediT taxonomy, authors contributed as follows. Conceptualization, Writing – original draft, Writing – review & editing: ASTI and LC; Data curation, Software: ASTI. Funding acquisition, Supervision: LC.

References

[1] E. Adamopoulou and L. Moussiades. Chatbots: History, technology, and applications. *Machine Learning with Applications*, 2:100006, 2020. ISSN 2666-8270. doi: https://doi.org/10.1016/j.mlwa.2020.100006. URL https://www.sciencedirect.com/science/article/pii/S2666827020300062.

[2] M. Aliannejadi, J. Kiseleva, A. Chuklin, J. Dalton, and M. Burtsev. Convai3: Generating clarifying questions for open-domain dialogue systems (clariq), 2020.

[3] M. Assante, L. Candela, D. Castelli, R. Cirillo, G. Coro, L. Frosini, L. Lelii, F. Mangiacrapa, P. Pagano, G. Panichi, and F. Sinibaldi. Enacting open science by d4science. *Future Generation Computer Systems*, 101:555–563, 2019. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2019.05.063. URL https://www.sciencedirect.com/science/article/pii/S0167739X1831464X.

[4] P. Dasigi, K. Lo, I. Beltagy, A. Cohan, N. A. Smith, and M. Gardner. A dataset of information-seeking questions and answers anchored in research papers. 2021.

[5] T. Davidson, D. Warmsley, M. Macy, and I. Weber. Automated hate speech detection and the problem of offensive language, 2017.

[6] A. Elgohary, D. Peskov, and J. Boyd-Graber. Can you unpack that? learning to rewrite questions-in-context. In *Empirical Methods in Natural Language Processing*, 2019. URL http://umiacs.umd.edu/~jbg//docs/2019_emnlp_sequentialqa.pdf.

[7] J. Gao, C. Xiong, P. Bennett, and N. Craswell. Neural approaches to conversational information retrieval, 2022. URL https://arxiv.org/abs/2201.05176.

[8] A. S. T. Ibrahim, L. Candela, and M. G. C. A. Cimino. Development of a conversational software agent for a virtual research environment. Master's thesis, Università di Pisa, 2023. unpublished thesis.

[9] Q. Jin, B. Dhingra, Z. Liu, W. W. Cohen, and X. Lu. Pubmedqa: A dataset for biomedical research question answering, 2019.

[10] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[11] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. URL http://arxiv.org/abs/1907.11692.

[12] P. Lu, S. Mishra, T. Xia, L. Qiu, K.-W. Chang, S.-C. Zhu, O. Tafjord, P. Clark, and A. Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. In *The 36th Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[13] S. Narayan, S. B. Cohen, and M. Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *ArXiv*, abs/1808.08745, 2018.

[14] T. Nguyen, M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, and L. Deng. MS MARCO: A human generated machine reading comprehension dataset. *CoRR*, abs/1611.09268, 2016. URL http://arxiv.org/abs/1611.09268.

[15] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

[16] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108, 2019.

[17] N. Stiennon, L. Ouyang, J. Wu, D. M. Ziegler, R. Lowe, C. Voss, A. Radford, D. Amodei, and P. Christiano. Learning to summarize from human feedback. In *NeurIPS*, 2020.

[18] S. Zhang, E. Dinan, J. Urbanek, A. Szlam, D. Kiela, and J. Weston. Personalizing dialogue agents: I have a dog, do you have pets too?, 2018.