ABSTRACT

With the rapid progress of the semantic web, a huge amount of structured data has become available on the web in the form of knowledge bases (KBs). Making these data accessible and useful for end-users are one of the main objectives of chatbots over linked data. Building a chatbot over linked data raises different challenges, including user queries understanding, multiple knowledge base support, and multilingual aspects. To address these challenges, we first design and develop an architecture to provide an interactive user interface. Secondly, we propose a machine learning approach based on intent classification and natural language understanding to understand user intents and generate SPARQL queries. The system can be extended with a new domain on-demand, flexible, multiple knowledge base, multilingual, and allows intuitive creation and execution of different tasks for an extensive range of topics. Furthermore, evaluation and application cases in the chatbot are provided to show how it facilitates interactive semantic data towards different real application scenarios and showcase the proposed approach for a knowledge graph and data-driven chatbot.

List of Tables

List of Figures

List of Abbreviations

1. NLU – Natural Language Understanding
2. NLP – Natural Language Processing
3. ASR – Automatic Speech Recognition
4. SVM – Support Vector Machine
5. SVC – Support Vector Classifier
6. NER – Named Entity Recognition
7. KE – Keyword Extraction
8. SPARQL – Sparkle Protocol and RDF Query Language
9. NLTK – Natural Language Toolkit
10. TF-IDF – Text Frequency–Inverse Document Frequency
11. AI – Artificial Intelligence
12. KB – Knowledge Bases
13. FAQ – Frequently Asked Questions
14. API – Application Program Interface

Chapter 1: INTRODUCTION

1.1 Background

The use of chatbots has been very popular since its inception in 1960. After two decades, research and development have seen impressive progress from Eliza 1960 to AI chatbots such as Siri 2010, Cortona, and Google assistant. Early chatbot systems, such as Eliza, Parry, and Alice, were designed based on text conversation. A chatbot is a virtual agent able to assist users by providing instant responses to the instant question provided by the user. It is not just a conversational system; they can also carry out other tasks such as ordering, booking, customer care, and many other tasks. In the past several years, giant companies have invested in artificial intelligence and developed several chatbots, among them Apple's Siri, Microsoft Cortana, Google Assistant, Facebook Messenger, and Alexa.

1.2 Problem Statement

In the context of linked data, the main purpose of chatbot systems is to retrieve useful and relevant information from one or multiple knowledge bases (KBs) by using natural language understanding (NLU) and semantic web technologies. This purpose is generally addressed by transforming natural language into a SPARQL query. Many chatbot systems have been proposed, but they require a lot of training data, which is unavailable and expensive to create. Recently, with the growth development of linked data, increasing progress on chatbots have been seen in research and industry. However, they are still facing many challenges, including user queries understanding, intent classification, multilingual aspect, multiple knowledge base, and analytical queries understanding. So, in this paper, we propose a

chatbot (K-Bot) that addresses some of the above challenges, and that can compete in terms of performance with existing linked data chatbot.

1.3 Objectives

- The main objective is to build a K-Bot for natural language understanding over linked data.
- We build a classifier for intent classification using a machine learning model (SVM) in-order map query.
- We provide analytical queries engines, including data exploration, which empowers users to explore analytical queries.
- Ensuring scalability, K-Bot is flexible by adding other knowledge bases, supporting new languages, and aiming at different tasks.

1.4 Need for present study

With the rapid progress of the semantic web, a huge amount of structured data has become available on the web in the form of knowledge bases (KBs), so to use this data with the help of machines we need a bot which can understand linked data and able to retrieve the relevant information from semantic web. So K-Bot is useful for this which works over natural language understanding and display the results in user understandable rich knowledge graphical way.

NLU

Natural language understanding is a branch of artificial intelligence that uses computer software to understand input in the form of sentences using text or speech.

NLU enables human-computer interaction. It is the comprehension of human language such as English, Spanish and French, for example, that allows computers to understand commands without the formalized syntax of computer languages. NLU also enables computers to communicate back to humans in their own languages.

The main purpose of NLU is to create chat- and voice-enabled bots that can interact with the public without supervision. Many major IT companies, such as Amazon, Apple, Google and Microsoft, and startups have NLU projects underway.

How does natural language understanding work?

NLU analyzes data to determine its meaning by using algorithms to reduce human speech into a structured ontology -- a data model consisting of semantics and pragmatics definitions. Two fundamental concepts of NLU are intent and entity recognition.

Intent recognition is the process of identifying the user's sentiment in input text and determining their objective. It is the first and most important part of NLU because it establishes the meaning of the text.

Entity recognition is a specific type of NLU that focuses on identifying the entities in a message, then extracting the most important information about those entities. There are two types of entities: named entities and numeric entities. Named entities are grouped into categories -- such as people, companies and locations. Numeric entities are recognized as numbers, currencies and percentages.

Chapter 2: LITERATURE REVIEW

J. Weizenbaum, "Eliza, a computer program for the study of natural language communication between man and machine" ELIZA is a program operating within the MAC time-sharing system at MIT which makes certain kinds of natural language conversation between man and computer possible. Input sentences are analyzed on the basis of decomposition rules which are triggered by key words appearing in the input text. Responses are generated by reassembly rules associated with selected decomposition rules. The fundamental technical problems with which ELIZA is concerned are: (1) the identification of key words, (2) the discovery of minimal context, (3) the choice of appropriate transformations, (4) generation of responses in the absence of key words, and (5) the provision of an editing capability for ELIZA "scripts". A discussion of some psychological issues relevant to the ELIZA approach as well as of future developments concludes the paper.

Jiyou Jia, "CSIEC: A computer assisted English learning chatbot based on textual knowledge and reasoning" CSIEC (Computer Simulation in Educational Communication) system with newly developed multiple functions for English instruction still focuses on supplying a virtual chatting partner (chatbot), which can chat in English with the English learners anytime anywhere. It generates communicative responses according to the user input, the dialogue context, the user's and its own personality knowledge, common sense knowledge, and inference knowledge. All these kinds of knowledge are expressed in the form of NLML, an annotation language for natural language text. These NLMLs can either be automatically obtained through parsing the text, or be easily authored with the help of GUI editors designed by us. So, the CSIEC system suggests a naïve approach of logical reasoning and inference directly through syntactic and semantic analysis of textual knowledge. This approach has

advantages over the old ELIZA-like keywords matching mechanism. The chatting log summarization of free Internet usage within six months demonstrates this advantage.

Minghu Qiu, Feng-Lin Li, Siyu Wang, Xing Gao, Yan Chen, Weipeng Zhao, Haiqing Chen, Jun Huang, Wei Chu, "AliMe chat: A Sequence to Sequence and Rerank based chatbot engine" We propose AliMe Chat, an open-domain chatbot engine that integrates the joint results of Information Retrieval (IR) and Sequence to Sequence (Seq2Seq) based generation models. AliMe Chat uses an attentive Seq2Seq based rerank model to optimize the joint results. Extensive experiments show our engine outperforms both IR and generation-based models. We launch AliMe Chat for a real-world industrial application and observe better results than another public chatbot.

Kyo-Joong Oh, DongKun Lee, ByungSoo Ko, Ho-Jin Choi, "A Chatbot for Psychiatric Counseling in Mental Healthcare Service Based on Emotional Dialogue Analysis and Sentence Generation" There are early studies to attempt users for psychiatric counseling with chatbot. They lead to changes in drinking habits based on intervention approaches via chat bot. The application does not con-sider the user's psychiatric status through the conversations, continuous user monitoring, and ethical judgment in the intervention. We contend that more accurate and continuous emotion recognition gives better satisfaction to users who need mental health care. In addition, appropriate clinical psychological responses based on ethical responses are as well. We suggest a conversational service for psychiatric counseling that is adapted methodologies to understand counseling contents based on high-level natural language understanding (NLU), and emotion recognition based on a multi-modal approach. The methodologies enable continuous observation of emotional changes sensitively. In addition, the case-based counseling response model that combines ethical judgment models provides a suitable response to clinical psychiatric counseling.

Paris (Pei-Ting) Hsu, Jingshu Zhao, Kehan Liao, Tianyi Liu, Chen Wang, "AllergyBot: A Chatbot Technology Intervention for Young Adults with Food Allergies Dining Out" Dining out is one of the biggest challenges people with food allergies face. For young adults, especially, the fear of having an allergic reaction when dining out impairs social aspects of their life. The exhausting process of searching online and communicating with restaurants also increases their anxiety. To improve the quality of life for young adults with food allergies, we present AllergyBot, an intelligent and humane Chatbot that provides restaurants' allergy accommodation information based on users' allergens. We use established instant messaging platforms to create a form of conversation that young adults are familiar with. AllergyBot aims to reduce the users' inquiry overload, improve their overall dining out experiences, and support their social life.

Lei Cui, Furu Wei, Shaohan Huang, Chuanqi Tan, Chaoqun Duan, Ming Zhou, "SuperAgent: A Customer Service Chatbot for E-commerce Websites" Conventional customer service chatbots are usually based on human dialogue, yet significant issues in terms of data scale and privacy. In this paper, we present SuperAgent, a customer service chatbot that leverages large-scale and publicly available e-commerce data. Distinct from existing counterparts, SuperAgent takes advantage of data from in-page product descriptions as well as user-generated content from e-commerce websites, which is more practical and cost-effective when answering repetitive questions, freeing up human support staff to answer much higher value questions. We demonstrate SuperAgent as an add-on extension to mainstream web browsers and show its usefulness to user's online shopping experience.

Anbang Xu, Zhe Liu, Yufan Guo, Vibha Sinha, Rama Akkiraju, "A New Chatbot for Customer Service on Social Media" Users are rapidly turning to social media to request and receive customer service; however, a majority of these requests were not addressed timely or even not addressed at all. To overcome the problem, we created a new conversational system to automatically generate responses for user's requests on social media. Our system is

integrated with state-of-the-art deep learning techniques and is trained by nearly 1M Twitter conversations between users and agents from over 60 brands. The evaluation reveals that over 40% of the requests are emotional, and the system is about as good as human agents in showing empathy to help users cope with emotional situations. Results also show our system outperforms information retrieval systems based on both human judgments and an automatic evaluation metric.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean, "Distributed representations of words and phrases and their compositionality" The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of precise syntactic and semantic word relationships. In this paper we present several extensions that improve both the quality of the vectors and the training speed. By subsampling the frequent words, we obtain significant speedup and also learn more regular word representations. We also describe a simple alternative to the hierarchical softmax called negative sampling. An inherent limitation of word representations is their indifference to word order and their inability to represent idiomatic phrases. For example, the meanings of "Canada" and "Air" cannot be easily combined to obtain "Air Canada". Motivated by this example, we present a simple method for finding phrases in text, and show that learning good vector representations for millions of phrases is possible.

Ran Zhao, Oscar J. Romero, Alex Rudnicky, "SOGO: A Social Intelligent Negotiation Dialogue System" We propose a semi-automatic social intelligent negotiation dialogue system that interweaves task utterance with conversational strategies to engage human users in negotiation. Our two-phase system operates sequentially in a reasoning-and-generation loop: In the task phase, we leverage an off-the-shelf end-to-end dialogue model for negotiation to build a dialogue manager which decides the next system's task intention. Then, during the social phase, we employ a theory-driven, template-based

natural language generator to realize the task's intention as a genre of social conversational strategy. Subsequently, a set of conversational strategies are presented to a human expert who decides the final sentence to be uttered by the dialogue system. Compared to the baseline system, our proposed social intelligent dialogue system achieves a higher agreement rate and more "good deals" with humans while building interpersonal rapport.

Bernhard E. Boser, Isabelle M. Guyon, Vladimir N. Vapnik, "A training algorithm for optimal margin classifiers' ' A training algorithm that maximizes the margin between the training patterns and the decision boundary is presented. The technique is applicable to a wide variety of the classification functions, including Perceptron, polynomials, and Radial Basis Functions. The effective number of parameters is adjusted automatically to match the complexity of the problem. The solution is expressed as a linear combination of supporting patterns. These are the subset of training patterns that are closest to the decision boundary. Bounds on the generalization performance based on the leave-one-out method and the VC-dimension are given. Experimental results on optical character recognition problems demonstrate the good generalization obtained when compared with other learning algorithms.

Xuan-Son Vu, Addi Ait-Mlouk, Erik Elmroth, Lili Jiang, "Graph-based Interactive Data Federation System for Heterogeneous Data Retrieval and Analytics" Given the increasing number of heterogeneous data stored in relational databases, file systems or cloud environment, it needs to be easily accessed and semantically connected for further data analytic. The potential of data federation is largely untapped, this paper presents an interactive data federation system (https://vimeo.com/319473546) by applying large-scale techniques including heterogeneous data federation, natural language processing, association rules and semantic web to perform data retrieval and analytics on social network data. The system first creates a Virtual Database (VDB) to virtually integrate data from multiple data sources. Next, an RDF generator is built to unify data, together with SPARQL queries, to support semantic data search over the processed text data by natural language processing

(NLP). Association rule analysis is used to discover the patterns and recognize the most important co-occurrences of variables from multiple data sources. The system demonstrates how it facilitates interactive data analytics towards different application scenarios (e.g., sentiment analysis, privacy-concern analysis, community detection).

Xin Li, Dan Roth, "Learning question classifiers" In order to respond correctly to a free form factual question given a large collection of texts, one needs to understand the question to a level that allows determining some of the constraints the question imposes on a possible answer. These constraints may include a semantic classification of the sought-after answer and may even suggest using different strategies when looking for and verifying a candidate answer. This paper presents a machine learning approach to question classification. We learn a hierarchical classifier that is guided by a layered semantic hierarchy of answer types, and eventually classifies questions into fine-grained classes. We show accurate results on a large collection of free-form questions used in TREC 10.

Ram G. Athreya, Axel-Cyrille Ngonga Ngomo, Ricardo Usbeck, "Enhancing Community Interactions with Data-Driven Chatbots--The DBpedia Chatbot" In this demo, we introduce the DBpedia chatbot, a knowledge-graph-driven chatbot designed to optimize community interaction. The bot was designed for integration into community software to facilitate the answering of recurrent questions. Four main challenges were addressed when building the chatbot, namely (1) understanding user queries, (2) fetching relevant information based on the queries, (3) tailoring the responses based on the standards of each output platform (i.e., Web, Slack, Facebook) as well as (4) developing subsequent user interactions with the DBpedia chatbot. With this demo, we will showcase our solutions to these four challenges.

Chapter 3: SYSTEM ANALYSIS

3.1 Requirements Specification

1. Conversational system on linked data.
2. A web application of specified chatbot.
3. Use Intent Classification for better use of chatbot.

Requirements Model

Requirement's analysis, also called requirements engineering, is the process of determining user expectations for a new or modified product. These features, called requirements, must be quantifiable, relevant and detailed. In software engineering, such requirements are often called functional specifications.

Requirement's analysis is critical to the success or failure of a systems or software project. The requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

3.1.1 User requirements

1. Execution time should be fast
2. More Accurate
3. User Friendly

3.1.2 Software requirements

1. O/S: Windows/Linux/Mac-OS.

2. Language: Python, HTML, CSS, Javascript.
3. Backend: MySQL.
4. Browser: Chrome/FireFox/Edge/Safari.

3.1.3 Hardware requirements

1. SYSTEM: Pentium Dual Core
2. HARD DISK: 1 GB
3. MONITOR
4. RAM: 4 GB or more

3.1.4 Functional Requirements

Input:

A user question.

Output:

Relevant answer retrieved from Knowledge Bases.

3.1.5 Non-Functional Requirements

1. Execution qualities:
    a. Efficiency:

        The state or quality of being efficient, i.e., able to accomplish something with the least waste of time and effort; competency in performance.
2. Evolution qualities:
    a. Testability:

The means by which the presence, quality, or genuineness of anything is determined.

b. Extensibility:

To enlarge the scope of, or make more comprehensive, as operations, influence etc.

c. Scalability:

The ability of something, especially a computer system, to adapt to increased demands.

3.2 UML Diagrams for the project work

UML is an acronym that stands for Unified Modeling Language. Simply put, UML is a modern approach to modeling and documenting software. In fact, it's one of the most popular business process modeling techniques.

It is based on diagrammatic representations of software components. As the old proverb says: "a picture is worth a thousand words". By using visual representations, we are able to better understand possible flaws or errors in software or business processes.

The elements are like components which can be associated in different ways to make a complete UML picture, which is known as a diagram. Thus, it is very important to understand the different diagrams to implement the knowledge in real-life systems.

Any complex system is best understood by making some kind of diagrams or pictures. These diagrams have a better impact on our understanding. If we look around, we will realize that the diagrams are not a new concept but it is used widely in different forms in different industries.

We prepare UML diagrams to understand the system in a better and simple way. A single diagram is not enough to cover all the aspects of the system. UML defines various kinds of diagrams to cover most of the aspects of a system. UML was created as a result of the chaos revolving around software development and documentation. In the 1990s, there were several different ways to represent and document software systems. The need arose for a more unified

way to visually represent those systems and as a result, in 19941996, the UML was developed by three software engineers working at Rational Software.

Mainly, UML has been used as a general-purpose modeling language in the field of software engineering. However, it has now found its way into the documentation of several business processes or workflows. For example, activity diagrams, a type of UML diagram, can be used as a replacement for flowcharts. They provide both a more standardized way of modeling workflows as well as a wider range of features to improve readability and efficiency. Use cases are best discovered by examining the actors and defining what the actor will be able to do with the system. Since all the needs of a system typically cannot be covered in one use case, it is usual to have a collection of use cases. Together this use case collection specifies all the ways the system. An association provides a pathway for communication. The communication can be between use cases, actors, classes or interfaces. Associations are the most general of all relationships and consequently the most semantically weak. If two objects are usually considered independently, the relationship is an association. They provide both a more standardized way of modeling workflows as well as a wider range of features to improve readability and efficiency. Use cases are best discovered by examining the actors and defining what the actor will be able to do with the system. Since all the needs of a system typically cannot be covered in one use case, it is usual to have a collection of use cases.

By default, the association tool on the toolbox is unidirectional and drawn on a diagram with a single arrow at one end of the association. The end with the arrow indicates who or what is receiving the communication. A dependency is a relationship between two model elements in which a change to one model element will affect the other model element. Typically, on class diagrams, a dependency relationship indicates that the operations of the client invoke operations of the supplier. The workflow in this case begins from importing the dataset by the developer and then replacing missing values with mean value of corresponding column, model building, validating that model by generating a confusion matrix and finally predicting the test sample class label. Transitions are used to show the passing of the flow of control from activity.

The various UML diagrams are

1. Use-case diagram
2. Activity diagram
3. Class diagram

3.2.1 Use-case Diagram

A use case diagram is a graph of actors, a set of use cases enclosed by a system boundary, communication (participation) associations between the actors and users and generalization among use cases. The use case model defines the outside (actors) and inside (use case) of the system's behavior. Actors are not part of the system. Actors represent anyone or anything that interacts with (input to or receive output from) the system. Use-case diagrams can be used during analysis to capture the system requirements and to understand how the system should work. During the design phase, you can use use-case diagrams to specify the behavior of the system as implemented. Use-case is a sequence of transactions performed by a system that yields a measurable result of values for a particular actor. The use cases are all the ways the system may be used.

Use case is a list of actions or event steps, typically defining the interactions be- tween a role (known as an actor) and a system, to achieve a goal. In the case of the use case diagram developer and the end user are the actors. Use cases are best discovered by examining the actors and defining what the actor will be able to do with the system. Since all the needs of a system typically cannot be covered in one use case, it is usual to have a collection of use cases. Together this use case collection specifies.

An include relationship is a stereotyped relationship that connects a base use case to an inclusion use case. An include relationship specifies how behavior in the inclusion use case is used by the base use case. An extended relationship is a stereotyped relationship that specifies how the functionality of one-use case can be inserted into the functionality of another use case. Extend relationships between use cases are modeled as dependencies by using the Extend stereotype. The different use cases are import dataset for importing datasets, preprocessing, model building, validation and prediction.

Extend is a directed relationship that specifies how and when the behavior defined in the usually supplementary (optional) extending use case can be inserted into the behavior defined in the extended use case. Extended use case is meaningful on its own, it is independent of the extending use case. Extending use cases typically defines optional behavior that is not necessarily meaningful by itself. The extended relationship is owned by the extending use case. The same extending use case can extend more than one use case, and extending use case may itself be extended. The extension takes place at one or more extension points defined in the extended use case. Extend relationship is shown as a dashed line with an open arrowhead directed from the extending use case to the extended (base) use case. The arrow is labeled with the keyword «extend».

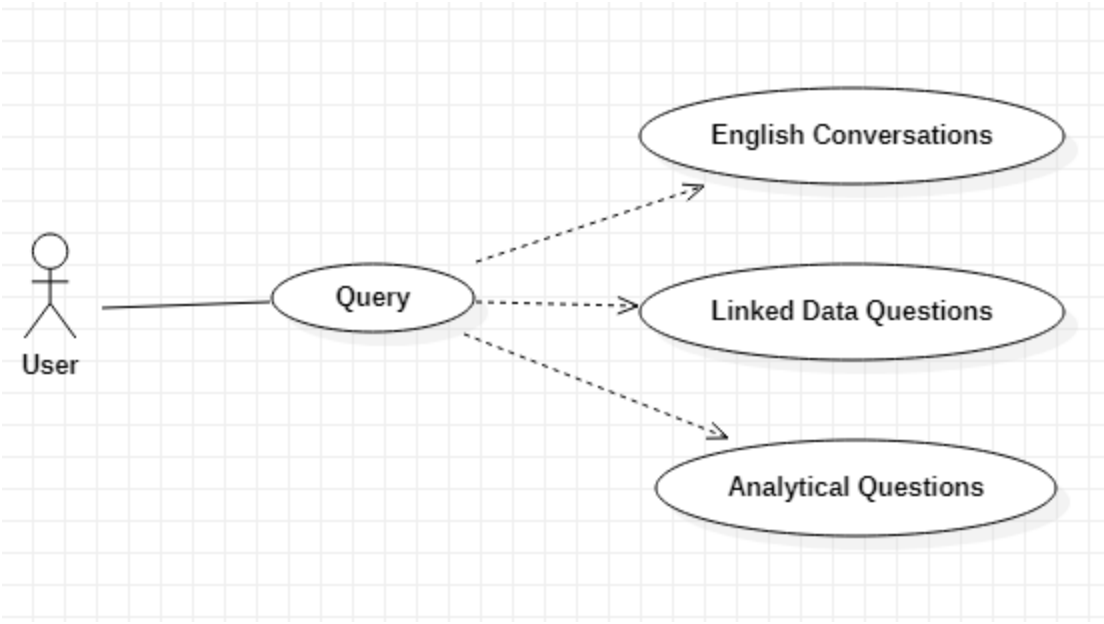


FIGURE 1. Use Case Diagram for User

3.2.2 Activity Diagram

An Activity diagram is a variation of a special case of a state machine, in which the states are activities representing the performance of operations and the transitions are triggered by the completion of the operations. The purpose of the Activity diagram is to provide a view

of flows and what is going on inside a use case or among several classes. Activity diagrams contain activities, transitions between the activities, decision points, and synchronization bars. An activity represents the performance of some behavior in the workflow. In the UML, activities are represented as rectangles with rounded edges, transitions are drawn as directed arrows, decision points are shown as diamonds, and synchronization bars are drawn as thick horizontal or vertical bars as shown in the following. The activity icon appears as a rectangle with rounded ends with a name and a component for actions.

The workflow in this case begins from importing the configuration files and database and then preprocessing the sentence to parse. After parsing the sentence is split into sub queries and then estimating the attributes. Finally, a query is formed. Transitions are used to show the passing of the flow of control from activity to activity. They are typically triggered by the completion of the behavior in the originating activity.

Transition connects activities with other model elements and object flows connect activities with objects. They are typically triggered by the completion of the behavior in the originating activity.

Swim lanes may be used to partition an activity diagram. This typically is done to show what person or organization is responsible for the activities contained in the swim lane. Swim lanes are helpful when modeling a business workflow because they can represent organizational units or roles within a business model. Swim lanes are very similar to an object because they provide a way to tell who is performing a certain role. Swim lanes only appear on activity diagrams. When a swim lane is

dragged onto an activity diagram, it becomes a swimlane view. Swim lanes ap- pear as small icons in the browser while swim lane views appear between the thin, vertical lines with a header that can be renamed and relocated. An activity represents the performance of some behavior in the workflow. In the UML, activities are rep- resented as rectangles with rounded edges, transitions are drawn as directed arrows, decision 17 points are shown as diamonds, and synchronization bars are drawn as thick horizontal or vertical bars as shown in the following.

The activity icon appears as a rectangle with rounded ends with a name and a component for actions.

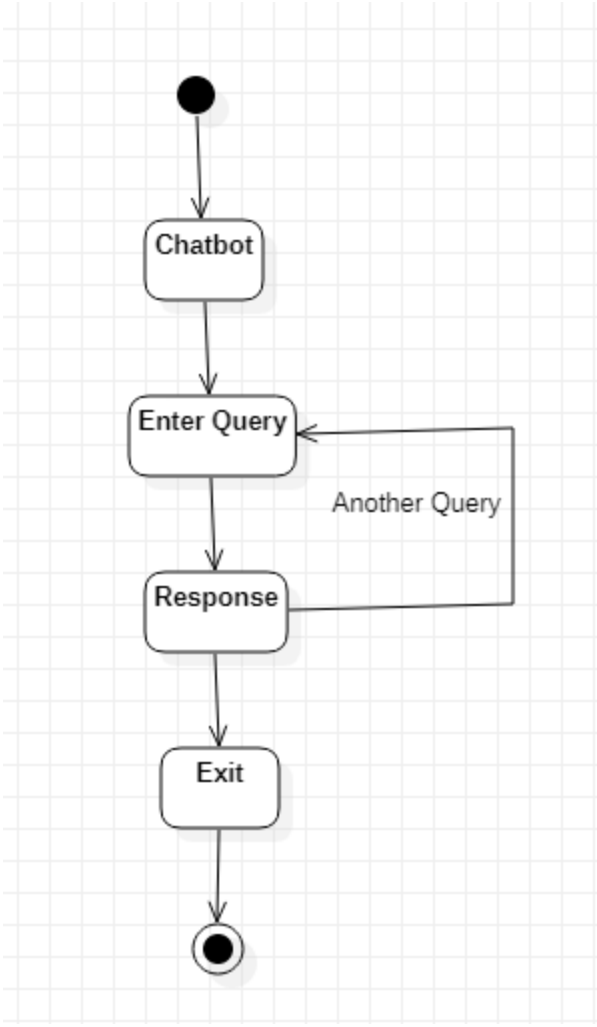


FIGURE 2. Activity Diagram for User

3.2.3 Class Diagram

Class diagrams contain icons representing classes, interfaces, and the relationships. You can create one or more class diagrams to represent the classes at the top level of the current model; such class diagrams are themselves contained by the top level of the current model. You can also create one or more class diagrams to represent classes contained by each package in

your model, such class diagrams are themselves contained by the package enclosing the classes they represent, the icons representing logical packages and classes in class diagrams.

1. Class diagrams are created to provide a picture or view of some or all of the classes in the model.

2. The main class diagram in the logical view of the model is typically a picture of the packages in the system. Each package also has its own main class diagram, which typically displays the public classes of the package.

A class diagram is a picture for describing generic descriptions of possible systems. Class diagrams and collaboration diagrams are alternate representations of object models.

A Class is a description of a group of objects with common properties (attributes) , common behavior(operations), common relationships to their objects, and com- mon semantics. Thus, a class is a template to create objects. Each object is

an instance of some class and objects cannot be instances of more than one class. In the UML, classes are represented as compartmentalized rectangles.

1. The top compartment contains the name of the class.
2. The middle compartment contains the structure of the class(attributes).
3. The bottom compartment contains the behavior of the class(operations).

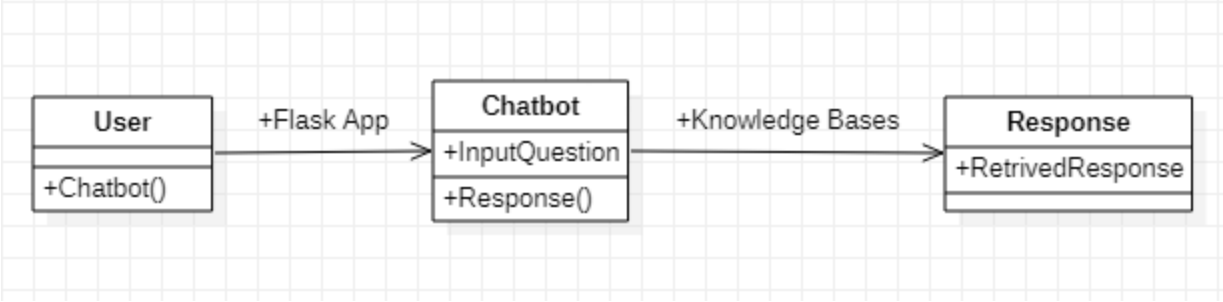


FIGURE 2. Class Diagram for User

3.2.4 Sequence Diagram

A sequence diagram is an interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence

diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams.

3.2.5 Collaboration Diagram

A collaboration diagram shows the order of messages that implement an operation or a transaction. Collaboration diagrams show objects, their links, and their messages. They can also contain simple class instances and class utility instances. Each collaboration diagram provides a view of the interactions or structural relationships that occur between objects and object-like entities in the current model. Collaboration diagrams and sequence diagrams are called interaction diagrams. A collaboration diagram shows the order of messages that implement an operation or a transaction. Collaboration diagrams show objects, their links, and their messages. They can also contain simple class instances and class utility instances. Each collaboration diagram provides a view of the interactions or structural relation-
ships that occur between objects and object-like entities in the current model.

The second interaction diagram is the collaboration diagram. It shows the object organization as seen in the following diagram. In the collaboration diagram, the method call sequence is indicated by some numbering technique. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram. Method calls are similar to that of a sequence diagram. However, the difference being the sequence diagram does not describe the object organization, whereas the collaboration diagram shows the object organization. To choose between these two diagrams, emphasis is placed on the type of requirement. If the time sequence is important, then the sequence diagram is used. If organization is required, then a collaboration diagram is used. interaction diagrams are used to describe the dynamic nature of a system. Now, we will look into the practical scenarios where these diagrams are used. To understand the practical application, we need to understand the basic nature of sequence and collaboration diagrams.

The main purpose of both the diagrams are similar as they are used to capture the dynamic behavior of a system. However, the specific purpose is more important to clarify and understand.

Sequence diagrams are used to capture the order of messages flowing from one object to another. Collaboration diagrams are used to describe the structural organization of the objects taking part in the interaction. A single diagram is not sufficient to describe the dynamic aspect of an entire system, so a set of diagrams are used to capture it as a whole.

Interaction diagrams are used when we want to understand the message flow and the structural organization. Message flow means the sequence of control flow from one object to another. Structural organization means the visual organization of the elements in a system.

3.2.6 State Chart Diagram

Use cases and scenarios provide a way to describe system behavior; in the form of interaction between objects in the system. Sometimes it is necessary to consider the inside behavior of an object. A state chart diagram shows the states of a single object, the events or messages that cause a transition from one state to another and the actions that result from a state change. As an activity diagram, the state chart diagram also contains special symbols for start state and stop state.

State chart diagrams cannot be created for every class in the system, it is only for those class objects with significant behavior. State transition: A state transition indicates that an object in the source state will perform certain specified actions and enter the destination state when a specified event occurs or when certain conditions are satisfied.A state transition is a relationship between two states, two activities, or between an activity and a state.We can show one or more state transitions from a state as long as each transition is unique. Transitions originating from a state cannot have the same event, unless there are conditions on the event.

Provide a label for each state transition with the name of at least one event that causes the state transition. You do not have to use unique labels for state transitions because the same event can cause a transition to many different states or activities.

3.2.7 Component Diagram

Component Diagrams show the dependencies between software components in the system. The nature of these dependencies will depend on the language or languages used for the development and may exist at compile-time or at runtime.

In a large project there will be many files that makeup the system. These files will have dependencies on one another. The nature of these dependencies will depend on the language or languages used for the development and may exist at compile-time, at link-time or at run-time. There are also dependencies between source code

files and the executable files or bytecode files that are derived from them by compilation. Component diagrams are one of the two types of implementation diagrams in UML. Component diagrams show these dependencies between software components in the system. Stereotypes can be used to show dependencies that are specific to particular languages also.

A component diagram shows the allocation of classes and objects to components in the physical design of a system. A component diagram may represent all or part of the component architecture of a system along with dependency relationships. The dependency relationship indicates that one entity in a component diagram uses the services or facilities of another.

3.2.8 Deployment Diagram

The second type of implementation diagram provided by UML is the deployment diagram. Deployment diagrams are used to show the configuration of run-time processing elements and the software components and processes that are located on them. Deployment diagrams are made up of nodes and communication associations.Nodes are typically used to show computers and the communication associations show the network and protocols that are used to communicate between nodes. Nodes can be used to show other processing resources such as people or mechanical resources. Nodes are drawn as 3D views of cubes or rectangular prisms, and the following figure shows a simplest deployment diagram where the nodes are connected by communication associations.

Chapter 4: SYSTEM DESIGN

4.1 Architecture

Our proposed approach is based on a modular approach, as shown in Figure 1, and takes advantage of semantic web techniques, knowledge graph, and machine learning. The proposed chatbot can handle different tasks (e.g., analytical queries, FAQs, etc.), gathering information from multiple sources (KBs, web services), and presenting them in the form of knowledge card (info-boxes that shows users' responses and displays only important attributes about user query). To enhance usability, users can interact with the system by using a chit-chat system or voice-based messages. The proposed chatbot is developed using Flask framework and can run on standalone or distributed mode to improve response time of information retrieval. The queries and user feedback are stored in a database in an anonymous way for continuous learning and future improvement.

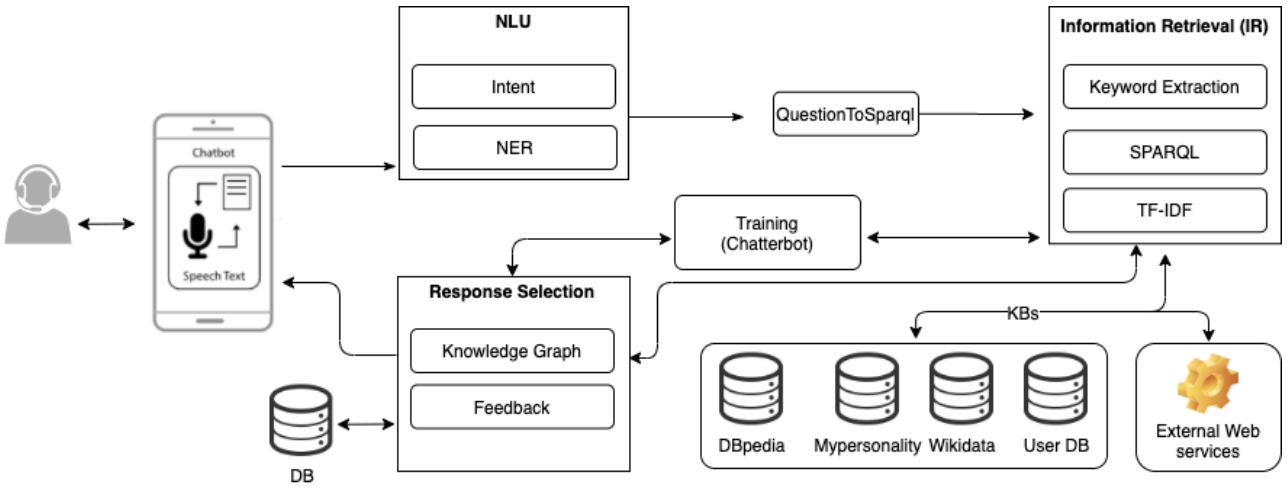


FIGURE 4. Architecture overview of K-Bot

4.2 Modules to be implemented

1. NLU
2. Langdetect
3. ASR
4. SVC
5. NER
6. KE
7. SPARQL
8. NLTK
9. TF-IDF
10. FLASK

4.3 Modules Description

NLU Module:

The interaction begins with a text/speech user query, which will be processed by the 'NLU' module. Speech Known as automatic speech recognition (ASR) is one of the core techniques of computational linguistics that develops methodologies that enable the recognition of spoken language into text by computers. This technique can help disabled people who cannot use other devices to interact with chatbots, and it also improves typing speed. 'NLU' module firstly detects the language of the user query. Afterward, it classifies user intents using Support Vector Machine (SVM) and parses the user query's structure by using a specific parser in combination with a regular expression (Regex). After processing the user query, the 'NER' sub-module is used to extract the named entities mentioned in the query. This sub-module is considered as the key to understanding natural language questions. If the user query belongs to factoid questions class, the sub-module 'QuestionTosparql' converts the user question to SPARQL query and retrieves the relevant response from KBs (i.e., DBpedia, Wikidata) and third-party services (OpenStreetMap, API, etc.). Once the response is retrieved, the 'Response Selection' module selects a relevant answer and presents it to the user in a rich knowledge panel.

Langdetect:

Most existing chatbots and conversational systems have been developed for English users, given a large number of English resources available on the web. There is always a need to handle multilingual queries on conversational systems and chatbots to serve a wide range of users. In our proposed K-Bot, we used a langdetect library to detect the language from the user question automatically. This module can analyze user questions and detect the language

automatically. Afterward, the detected language (e.g., Swedish, Spanish, Arabic, etc.) will be used by ‘Response Retrieval’ to retrieve the relevant answer; otherwise, the answer will be in English since the most covered content in KBs is in English. For instance, if a user asks a question in French ‘‘C’est qui Albert Einstein?’’, ‘NLU’ module will detect that the user query is posted in French, and ‘QuestionToSparql’ will retrieve and filter only the answer in French.

ASR Module:

ASR is a python module which is used to detect language of user query. Speech Known as automatic speech recognition (ASR) is one of the core techniques of computational linguistics that develops methodologies that enable the recognition of spoken language into text by computers. Automatic Speech Recognition or ASR, as it’s known in short, is the technology that allows human beings to use their voices to speak with a computer interface in a way that, in its most sophisticated variations, resembles normal human conversation.

SVC:

Intent classification is usually the first stage in conversational systems. It is the process of mapping queries to a predefined class, and it aims to facilitate user query understanding. Query classes can raise different extraction strategies. The strategy used to search for the answer for a query like ‘‘Who is Alan Turin’’ is probably different from the strategy used to search for a question like ‘‘Who invented the Turing machine’’. The first query is about description more specifically about the definition of ‘‘Alan Turing’’, while the second one expects the name of a person, which is ‘‘Alan Turing’’. To address this critical issue of distinguishing query types, we used a Support Vector Machine (SVM) based query classification aiming to be language and domain-independent. We specifically focused on the user query classification part, and the purpose is to classify a given user query into predefined categories. This classification will help to identify user intention before generating SPARQL queries. In this context, we use a supervised machine learning model called support vector

classifier (SVC) that uses classification algorithms for two-group classification. SVC takes training data points and outputs the hyperplane (decision boundary) that best separates two classes of samples. SVC has demonstrated to perform well with high dimensional data used in many NLP tasks, including text classification.

NER Module:

NER is a python module which is used to perform subtask of information extraction that seeks to locate and classify named entities mentioned in unstructured text into predefined categories such as person names, organizations, locations, medical codes, time expressions, quantities, monetary values, percentages, etc. NER is one of the main components of the proposed KBot, which is employed to extract information about different types of entities (PERSON, GPE, ORG, etc.), relationships, or events by using tokenization and part-of-speech tagging (POS) approaches. For a given user query, we extract entities based on user intent then generate SPARQL queries to facilitate the exploitation of linked data (e.g., DBpedia, Wikidata, etc.) and use their strength to retrieve relevant answers.

For instance, let Q = ‘‘Who is Alan Turing?’’ denote a user query; the K-Bot will identify the class of question as ‘‘PERSON’’ and extract ‘‘Alan Turing’’ as the main entity. Afterwards, it generates the SPARQL query that gathers information about this entity from knowledge bases and presents it in a knowledge panel. Figure 2 presents a knowledge graph for the extracted entity from the user query.

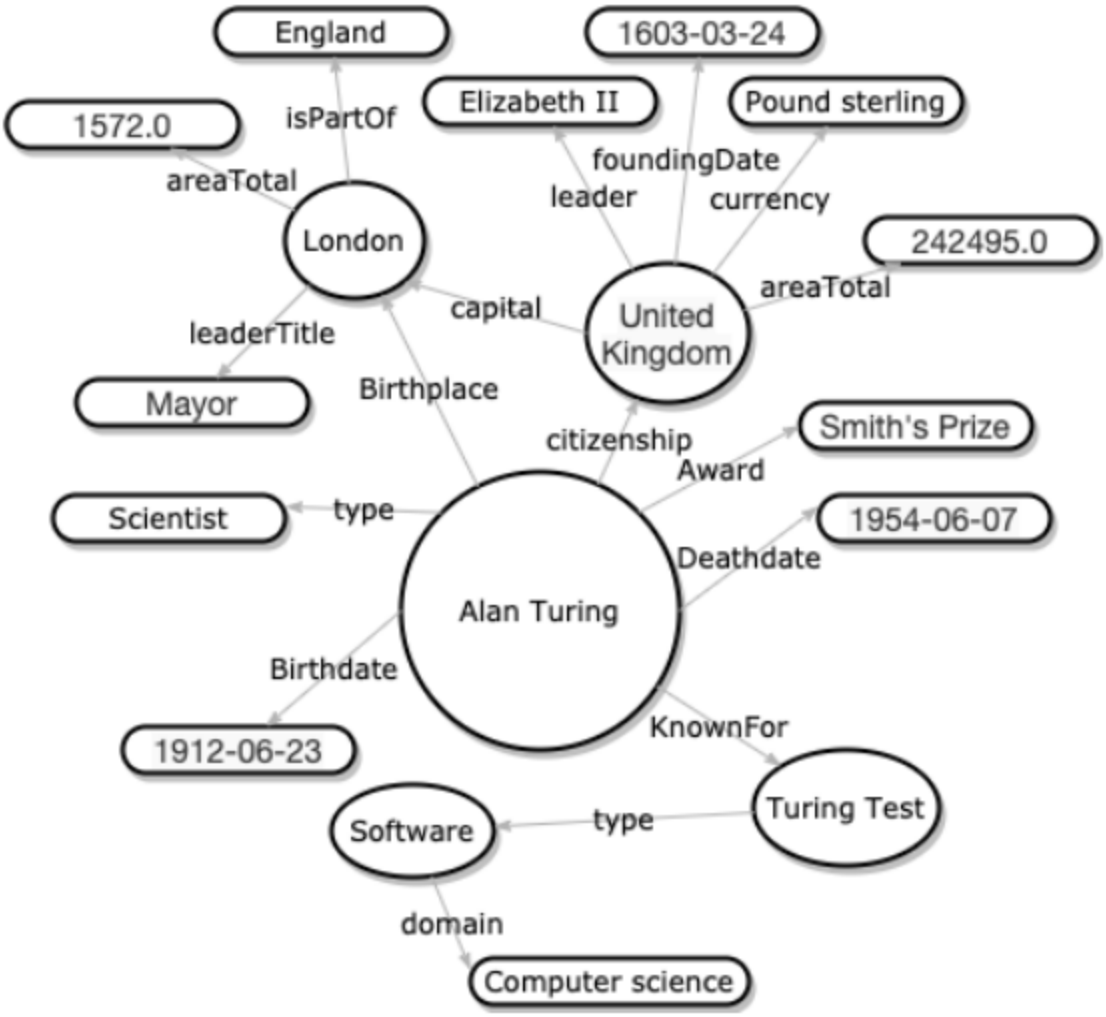


FIGURE 5. Knowledge graph example

KE:

Keyword extraction is a text analysis technique with the automatic identification of terms that best describe a text document's subject. It helps summarize the content of a text and recognize the main topics which are being discussed. In this step, we use a specific parser with stop words and a regular expression to allow the chatbot to lift the most important words from user queries.

SPARQL:

SPARQL (pronounced "sparkle", a recursive acronym for SPARQL Protocol and RDF Query Language) is an RDF query language—that is, a semantic query language for databases—able to retrieve and manipulate data stored in Resource Description Framework (RDF) format. In this step, we construct a set of SPARQL queries after processing and understanding the user query. These SPARQL queries represent a possible interpretation of user queries within the given KBs (DBpedia, Wikidata). The main objective is to generate possible queries containing information about user queries. The main challenge is to construct a SPARQL query from user questions efficiently and query multiple knowledge bases according to user intent to retrieve a result-set.

NLTK Module:

NLTK is a python module which is one of the leading platforms for working with human language data and Python, the module NLTK is used for natural language processing. NLTK is literally an acronym for Natural Language Toolkit. We used the NLTK library to summarize and improve the result using TF-IDF; this can extract useful and relevant information.

TF-IDF:

In information retrieval, tf–idf, TF*IDF, or TF-IDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The TF-IDF value increases proportionally to the number of times a word appears in the document and is offset by the

number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

Query	Where is Sweden located geographically?
Answer	Approximately 85 of the population lives in urban areas. Sweden is part of the geographical area of Fennoscandia. The capital city is Stockholm, which is also the most populous city in the country. Legislative power is vested in the 349-member unicameral Riksdag. Executive power is exercised by the government chaired by the prime minister.

TABLE 1. Example of answer to the user query using TF-IDF

With the huge amount of semantic data in knowledge bases, which is mostly textual data, there is a need to develop automatic text summarization techniques that easily allow users to get insights. Most of the information is redundant, insignificant, and may not convey the intended meaning. For instance, if the user searches for specific information about entities (Google, Alan Turing, etc.) in linked data abstract and page description (DBpedia, Wikidata, etc.), he may have to dig through its content and spend unnecessary time before getting the intended information. In this context, we used the NLTK10 library to summarize and improve the result using TF-IDF; this can extract useful and relevant information. Text summarization using TF-IDF refers to the process of shortening the long text to only the main points outlined in the document. TF-IDF is an information retrieval technique that weighs a term's frequency (TF) and its inverse document frequency (IDF). TF is the number of times a word appears in a document divided by the number of words in the document, while IDF is the log of the number of documents divided by the number of documents containing the word w. An example is given in Table 1.

FLASK:

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

Chapter 5: ALGORITHM ANALYSIS

Input : Question, Endpoint
Output: Response

$Entities \leftarrow \emptyset$
$Keywords \leftarrow \emptyset$

$Input \leftarrow Question$
$Query \leftarrow Process\_Input(Input)$
$Intent \leftarrow Get\_Intent(Query)$
foreach *Word* $W_i \in$ *Question* **do**
    $Entity \leftarrow Ner\_Tagger(W_i)$
    $Entities \leftarrow Entities + Entity$
end foreach
if *Entities equal 0* **then**
    $Keywords \leftarrow Parser(Query)$
    $Response \leftarrow Get\_Response(Keywords)$
else
    $Query \leftarrow$
     $QuestionToSparql(Entities, Intent)$
    $Response \leftarrow$
     $Get\_Response(Endpoint, Query)$
end if
return Response

As shown we take an input question and we process the input question through three stages - Intent Classification to get intents, Named Entity Recognition to get entities and Keyword Extraction to get keywords. Intents are used to well known about what user is asking i.e., it tells about user perspective in the input question. We use derived intents in Sparql Query Generation. Entities are used to extract the main point of question. With combination of both entities and intents we generate either DBPedia sparql query or WikiData sparql query with the help of Sparql Query Generation Module. Here keywords are used when there are no named entities obtained from Named Entity Recognition. In such a case we generally give keywords derived instead of entities. Finally the query generated using either entities and intents or keywords and intents extracts the required solution from Knowledge Bases which is stored in the response attribute and it is returned.

Attributes:

input question, intents, entities, keywords, query, response.

Chapter 6: IMPLEMENTATION

6.1 Intent Classification

```
class IntentClassifier:
    classifier = None

    @classmethod
    def get_svm_config(self):
        config = {
            "kernel": "linear",
            "C": [1,2,5,10,20,100]
        }
        return config

    @classmethod
    def train(self):
        X, labels = DataHelper.get_data_and_labels()
        X, labels = shuffle(X,labels)
        le = LabelEncoder()
        y = le.fit_transform(labels)

        DataHelper.save_index_to_label_mapping(y,le)

        sklearn_config = IntentClassifier.get_svm_config()
        C = sklearn_config.get("C")
        kernel = sklearn_config.get("kernel")

        tuned_parameters = [{"C": C, "kernel": [str(kernel)]}]

        clf = GridSearchCV(SVC(probability=True, class_weight='balanced',decision_function_shape='ovr'),
                           param_grid=tuned_parameters,
                           cv=5, scoring='f1_weighted', verbose=1)
        clf.fit(X, y)
        print("best params : {} , best F1: {}".format(clf.best_params_,clf.best_score_))
        f = open(MODEL_PATH, 'wb')
        pickle.dump(clf, f)
        f.close()
        return clf
```

```
    @classmethod
    def get_classifier(self):
        if(self.classifier is None):
            print("Loading classifier")
            try:
                f = open(MODEL_PATH, 'rb')   # 'rb' for reading binary file
                self.classifier = pickle.load(f)
                print("Loaded classifier")
                f.close()
            except:
                print("Exception in loading classifier")
        return self.classifier

    @classmethod
    def predict(self,query):
        model = self.get_classifier()
        if(model is not None):
            query_lemmatized = DataHelper.lemmatize(str(query))
            prediction = model.predict_proba(np.array(spacy_model(query_lemmatized).vector).reshape(1,96))
            index_to_label_dict = DataHelper.get_index_label_dict()
            label_index = np.argmax(prediction)
            label = index_to_label_dict[str(label_index)]
            class_score = prediction[0][label_index]
            return {"label":label,
                    "probability":class_score}
        else:
            print("Unable to load classifier")
```

Output:

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
best params : {'C': 1, 'kernel': 'linear'} , best F1: 0.8655177800304864
GridSearchCV(cv=5, estimator=SVC(class_weight='balanced', probability=True),
             param_grid=[{'C': [1, 2, 5, 10, 20, 100], 'kernel': ['linear']}],
             scoring='f1_weighted', verbose=1)
```

```
[8] # Example for intent classification
    print(IntentClassifier.predict("Who is alan turing")['label'])
```

```
Loading classifier
Loaded classifier
who
```

6.2 Named Entity Recognition

```
import nltk
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')
nltk.download('punkt')

class NER:
    @classmethod
    def parts_of_speech(self,message,query_properties):
        sentences = nltk.sent_tokenize(message)
        tokenized = [nltk.word_tokenize(sentence) for sentence in sentences]
        pos_tags = [nltk.pos_tag(sentence) for sentence in tokenized]
        print('pos tags:',pos_tags[0])

        for name, tag in pos_tags[0]:
            if tag == 'NN':
                name = name.split("-")
                name = " ".join(name)
                query_properties.append(name)
        print('query_proprties:',query_properties)
        chunked_sents = nltk.ne_chunk_sents(pos_tags, binary=True)

        return chunked_sents

    @classmethod
    def traverse(self,tree):
        entity_names=[]
        if hasattr(tree, 'label') and tree.label:
            if tree.label()=='NE':
                entity_names.append(' '.join([child[0] for child in tree]))
            else:
                for child in tree:
                    entity_names.extend(self.traverse(child))

        return entity_names
```

```
    @classmethod
    def named_entities(self,message):
        # Identifying Parts of Speech
        query_properties=[]
        chunks=self.parts_of_speech(message,query_properties)

        entities=[]
        for chunk in chunks:
            temp=sorted(list(set([word for tree in chunk for word in self.traverse(tree)])))
            for e in temp:
                if e not in entities:
                    entities.append(e)

        return entities,query_properties
```

Output:

```
# Example for named entity recognition
NER.named_entities('Who invented Turing machine')
```

```
pos tags: [('Who', 'WP'), ('invented', 'VBD'), ('Turing', 'NNP'), ('machine', 'NN')]
query_proprties: ['machine']
(['Turing'], ['machine'])
```

6.3 Keyword Extraction

```
import re
import math
import operator
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import sent_tokenize, word_tokenize
nltk.download('stopwords')
nltk.download('wordnet')

Stopwords = set(stopwords.words('english'))
wordlemmatizer = WordNetLemmatizer()

class KE:
    @classmethod
    def lemmatize_words(self,words):
        lemmatized_words = []
        for word in words:
            lemmatized_words.append(wordlemmatizer.lemmatize(word))

        return lemmatized_words

    @classmethod
    def keywords(self,text):
        tokenized_sentence = sent_tokenize(text)
        tokenized_words_with_stopwords = word_tokenize(text)
        tokenized_words = [word for word in tokenized_words_with_stopwords if word not in Stopwords]
        tokenized_words = [word for word in tokenized_words if len(word) > 1]
        tokenized_words = [word.lower() for word in tokenized_words]
        tokenized_words = self.lemmatize_words(tokenized_words)
        return tokenized_words
```

Output:

```
# Example for keyword extraction
KE.keywords("Who is Alan Turing")
```

```
['who', 'alan', 'turing']
```

6.4 Sparql Query Generation

```
[ ]  from SPARQLWrapper import SPARQLWrapper as sq, JSON
     from vocabulary.vocabulary import Vocabulary as vb
```

```
[ ]  class SPARQL:
         @classmethod
         def question_to_sparql(self,intents,named_entity,keywords,query_properties):
             property_code = []
             if len(query_properties) != 0:
                 properties = open('/content/drive/MyDrive/Colab Notebooks/data/query.json', 'r')
                 properties = json.load(properties)
                 noun = query_properties[0]
                 noun_synonyms = vb.synonym(noun, format="dict")

                 for p, prop in list(properties.items()):
                     if prop == noun:
                         property_code.append(p)
                         break

                 if len(property_code) == 0:
                     for p, prop in list(properties.items()):
                         if type(noun_synonyms) != bool:
                             for synonym in noun_synonyms.values():
                                 if prop == synonym:
                                     property_code.append(p)
                                     break
                 print('property_code:',property_code)
```

```
if len(named_entity) != 0:
    if len(property_code) != 0:
        flag = 0
        sparql = sq("https://query.wikidata.org/sparql")
        query = """SELECT ?label ?property ?thumbnail WHERE
                    {
                    ?entity rdfs:label ?label .
                    ?entity wdt:""" + property_code[0] + """ ?property_id .
                    ?property_id rdfs:label ?property .
                    OPTIONAL { ?property_id wdt:P18 ?thumbnail .}
                    FILTER (STR(?label) = '""" + named_entity[0] + """') .
                    FILTER (LANG(?property) = "en")
                    }"""
        print('query:',query)
    else:
        flag = 1
        sparql = sq("https://dbpedia.org/sparql")
        query = """SELECT ?label ?description ?thumbnail WHERE
            {
            ?entity rdfs:label ?label .
            ?entity dbo:abstract ?description .
            ?entity dbo:thumbnail ?thumbnail .
            FILTER (STR(?label) = '""" + named_entity[0] + """' && LANG(?description) = "en") .
            }
            LIMIT 1"""
        print('query:',query)
```

```
            sparql.setQuery(query)
            sparql.setReturnFormat(JSON)
            global result
            result = list()
            temp = str()
            try:
                results = sparql.query().convert()
                print('results:',results)
                if flag == 0:
                    for data in results["results"]["bindings"]:
                        result.append(data["property"]["value"])
                        if "thumbnail" in data and data["thumbnail"]["value"] != "":
                            temp = data["thumbnail"]["value"]
                else:
                    for data in results["results"]["bindings"]:
                        result.append(data["description"]["value"])
                        if "thumbnail" in data and data["thumbnail"]["value"] != "":
                            temp = data["thumbnail"]["value"]

                result = list(set(result))
                response = []
                for i in result:
                    print(i)
                    response.append(''.join(i).encode('utf-8'))
            except:
                response = ["Unable to retrieve data"]
        else:
            response = ["Unable to retrieve data"]

        print('response:',response)
        if response!=["Unable to retrieve data"] and response!=[]:
          response[0]=response[0].decode('utf-8')
        return response
```

6.5 Text Summarization

```
class TFIDF:
    @classmethod
    def remove_special_characters(self,text):
        regex = r'[^a-zA-Z0-9\s]'
        text = re.sub(regex,'',text)

        return text
```

```
@classmethod
def freq(self,words):
    words = [word.lower() for word in words]
    dict_freq = {}
    words_unique = []
    for word in words:
        if word not in words_unique:
            words_unique.append(word)
    for word in words_unique:
        dict_freq[word] = words.count(word)

    return dict_freq

@classmethod
def lemmatize_words(self,words):
    lemmatized_words = []
    for word in words:
        lemmatized_words.append(wordlemmatizer.lemmatize(word))

    return lemmatized_words

@classmethod
def pos_tagging(self,text):
    pos_tag = nltk.pos_tag(text.split())
    pos_tagged_noun_verb = []
    for word,tag in pos_tag:
        if tag == "NN" or tag == "NNP" or tag == "NNS" or tag == "VB" or tag == "VBD" or tag == "VBG" or tag == "VBN" or tag == "VBP" or tag == "VBZ":
            pos_tagged_noun_verb.append(word)

    return pos_tagged_noun_verb
```

```
    @classmethod
    def tf_score(self,word,sentence):
        word_frequency_in_sentence = 0
        len_sentence = len(sentence)
        for word_in_sentence in sentence.split():
            if word == word_in_sentence:
                word_frequency_in_sentence = word_frequency_in_sentence + 1
        tf =  word_frequency_in_sentence/ len_sentence

        return tf

    @classmethod
    def idf_score(self,no_of_sentences,word,sentences):
        no_of_sentence_containing_word = 0
        for sentence in sentences:
            sentence = self.remove_special_characters(str(sentence))
            sentence = re.sub(r'\d+', '', sentence)
            sentence = sentence.split()
            sentence = [word for word in sentence if word.lower() not in Stopwords and len(word)>1]
            sentence = [word.lower() for word in sentence]
            sentence = [wordlemmatizer.lemmatize(word) for word in sentence]
            if word in sentence:
                no_of_sentence_containing_word = no_of_sentence_containing_word + 1
        idf = math.log10(no_of_sentences/no_of_sentence_containing_word)

        return idf

    @classmethod
    def tf_idf_score(self,tf,idf):
        return tf*idf
```

```
    @classmethod
    def word_tfidf(self,word,sentences,sentence):
        tf = self.tf_score(word,sentence)
        idf = self.idf_score(len(sentences),word,sentences)
        tf_idf = self.tf_idf_score(tf,idf)

        return tf_idf

    @classmethod
    def sentence_importance(self,sentence,dict_freq,sentences):
        sentence_score = 0
        sentence = self.remove_special_characters(str(sentence))
        sentence = re.sub(r'\d+', '', sentence)
        pos_tagged_sentence = self.pos_tagging(sentence)
        for word in pos_tagged_sentence:
            if word.lower() not in Stopwords and word not in Stopwords and len(word)>1:
                    word = word.lower()
                    word = wordlemmatizer.lemmatize(word)
                    sentence_score = sentence_score + self.word_tfidf(word,sentences,sentence)

        return sentence_score

    @classmethod
    def tf_idf(self,text):
        print('text:',text)
        tokenized_sentence = sent_tokenize(text)
        if len(tokenized_sentence)<=4:
            return text
```

```
        tokenized_words_with_stopwords = word_tokenize(text)
        tokenized_words = [word for word in tokenized_words_with_stopwords if word not in Stopwords]
        tokenized_words = [word for word in tokenized_words if len(word) > 1]
        tokenized_words = [word.lower() for word in tokenized_words]
        tokenized_words = self.lemmatize_words(tokenized_words)
        word_freq = self.freq(tokenized_words)
        no_of_sentences = 2
        c = 1
        sentence_with_importance = {}
        for sent in tokenized_sentence:
            sentenceimp = self.sentence_importance(sent,word_freq,tokenized_sentence)
            sentence_with_importance[c] = sentenceimp
            c = c+1
        sentence_with_importance = sorted(sentence_with_importance.items(), key=operator.itemgetter(1),reverse=True)
        cnt = 0
        summary = []
        sentence_no = []
        for word_prob in sentence_with_importance:
            if cnt < no_of_sentences:
                sentence_no.append(word_prob[0])
                cnt = cnt+1
            else:
                break
        sentence_no.sort()
        cnt = 1
        for i in range(2):
            summary.append(tokenized_sentence[i])
        for sentence in tokenized_sentence:
            if cnt in sentence_no and sentence not in summary:
                summary.append(sentence)
            cnt = cnt+1
        summary = " ".join(summary)

        return str(summary)
```

Output:

```
text = "Alan Mathison Turing OBE FRS (/ˈtjʊərɪŋ/; 23 June 1912 – 7 Ju

# Example for text summerization
TFIDF.tf_idf(text)

text: Alan Mathison Turing OBE FRS (/ˈtjʊərɪŋ/; 23 June 1912 – 7 June
'Alan Mathison Turing OBE FRS (/ˈtjʊərɪŋ/; 23 June 1912 – 7 June 1954
philosopher, and theoretical biologist. Turing was highly influential
on of the concepts of algorithm and computation with the Turing machi
was a fellow at Cambridge, he published a proof demonstrating that so
and defined a Turing machine, and went on to prove the halting proble
hat was responsible for German naval cryptanalysis.'
```

6.6 Chatterbot

```python
# Pip imports
from chatterbot import ChatBot
from chatterbot.trainers import ChatterBotCorpusTrainer

class Chatbot:
    # Creation of Bot
    bot = ChatBot(
        'Lucy',
        preprocessors=[
            'chatterbot.preprocessors.clean_whitespace'
        ],
        storage_adapter='chatterbot.storage.SQLStorageAdapter',
        logic_adapters=[
            {
                'import_path': 'chatterbot.logic.BestMatch',
                'default_response': 'I am sorry, but I do not understand.',
                'maximum_similarity_threshold': 1.0
            }
        ],
        database_uri='sqlite:///database.sqlite3',
    )

    # Training
    trainer = ChatterBotCorpusTrainer(bot)
    trainer.train(
        "chatterbot.corpus.english"
    )

    @classmethod
    def chatbot_response(self,message):
        response = self.bot.get_response(message)
        return str(response)
```

Output:

```
Training ai.yml: [####################] 100%
Training botprofile.yml: [####################] 100%
Training computers.yml: [####################] 100%
Training conversations.yml: [####################] 100%
Training emotion.yml: [####################] 100%
Training food.yml: [####################] 100%
Training gossip.yml: [####################] 100%
Training greetings.yml: [####################] 100%
Training health.yml: [####################] 100%
Training history.yml: [####################] 100%
Training humor.yml: [####################] 100%
Training literature.yml: [####################] 100%
Training money.yml: [####################] 100%
Training movies.yml: [####################] 100%
Training politics.yml: [####################] 100%
Training psychology.yml: [####################] 100%
Training science.yml: [####################] 100%
Training sports.yml: [####################] 100%
Training trivia.yml: [####################] 100%
```

6.7 KBot

```
class KBOT:
    @classmethod
    def kbot_response(self,message):
        # Intent Classification
        intents=IntentClassifier.predict(message)['label']
        print('Intents:',intents)

        # Named Entity Recognition
        entities,query_properties=NER.named_entities(message)
        print('Entities:',entities)


        # Keyword Extraction
        keywords=KE.keywords(message)
        print('Keywords:',keywords)

        # SPARQL Query Generation
        response=SPARQL.question_to_sparql(intents,entities,keywords,query_properties)
        print('Response after SPARQL execution:',response)

        # Text Summerization
        if response!=['Unable to retrieve data'] and response!=[]:
            response=TFIDF.tf_idf(response[0])
        else:
            response='Unable to retrieve data'
        print('Response after TF-IDF:',response)

        return response
```

Output:

```
KBOT.kbot_response("who is Adolf Hitler")
```

```
Intents: who
pos tags: [('who', 'WP'), ('is', 'VBZ'), ('Adolf', 'NNP'), ('Hitler', 'NNP')]
query_proprties: []
Entities: ['Adolf Hitler']
Keywords: ['adolf', 'hitler']
query: SELECT ?label ?description ?thumbnail WHERE
                    {
                    ?entity rdfs:label ?label .
                    ?entity dbo:abstract ?description .
                    ?entity dbo:thumbnail ?thumbnail .
                    FILTER (STR(?label) = 'Adolf Hitler' && LANG(?description) = "en") .
                    }
                    LIMIT 1
results: {'head': {'link': [], 'vars': ['label', 'description', 'thumbnail']}, 'results': {'dis
Adolf Hitler (German: ['adɔlf 'hɪtlɐ]; 20 April 1889 – 30 April 1945) was an Austrian-born Germ
response: [b'Adolf Hitler (German: [\xcb\x88ad\xc9\x94lf \xcb\x88h\xc9\xaatl\xc9\x90]; 20 April
Response after SPARQL execution: ['Adolf Hitler (German: ['adɔlf 'hɪtlɐ]; 20 April 1889 – 30 Ap
text: Adolf Hitler (German: ['adɔlf 'hɪtlɐ]; 20 April 1889 – 30 April 1945) was an Austrian-bor
Response after TF-IDF: Adolf Hitler (German: ['adɔlf 'hɪtlɐ]; 20 April 1889 – 30 April 1945) wa
'Adolf Hitler (German: ['adɔlf 'hɪtlɐ]; 20 April 1889 – 30 April 1945) was an Austrian-born Ger
m 1933 until his death in 1945. He rose to power as the leader of the Nazi Party, becoming the
Führer und Reichskanzler in 1934. Former chancellor Franz von Papen and other conservative lead
oint Hitler as chancellor on 30 January 1933. Hitler sought Lebensraum (lit.'
```

```
KBOT.kbot_response("Who is Elon Musk")
```

```
Intents: who
pos tags: [('Who', 'WP'), ('is', 'VBZ'), ('Elon', 'NNP'), ('Musk', 'NNP')]
query_proprties: []
Entities: ['Elon Musk']
Keywords: ['who', 'elon', 'musk']
query: SELECT ?label ?description ?thumbnail WHERE
                    {
                    ?entity rdfs:label ?label .
                    ?entity dbo:abstract ?description .
                    ?entity dbo:thumbnail ?thumbnail .
                    FILTER (STR(?label) = 'Elon Musk' && LANG(?description) = "en") .
                    }
                    LIMIT 1
results: {'head': {'link': [], 'vars': ['label', 'description', 'thumbnail']}, 'results':
Elon Reeve Musk FRS (/'i:lɒn/; born June 28, 1971) is an entrepreneur and business magnat
response: [b"Elon Reeve Musk FRS (/\xcb\x88i\xcb\x901\xc9\x92n/; born June 28, 1971) is a
Response after SPARQL execution: ["Elon Reeve Musk FRS (/'i:lɒn/; born June 28, 1971) is
text: Elon Reeve Musk FRS (/'i:lɒn/; born June 28, 1971) is an entrepreneur and business
Response after TF-IDF: Elon Reeve Musk FRS (/'i:lɒn/; born June 28, 1971) is an entrepren
'Elon Reeve Musk FRS (/'i:lɒn/; born June 28, 1971) is an entrepreneur and business magna
early-stage investor, CEO and Product Architect of Tesla, Inc.; founder of The Boring Com
roposed the Hyperloop, a high-speed vactrain transportation system. In 2018, he was sued
m Luang cave rescue; a California jury ruled in favor of Musk.'
```

6.8 Flask App Creation

```
[ ]  from flask import Flask, render_template, request
     from flask_ngrok import run_with_ngrok

     # Flask app creation
     app = Flask(__name__)
     run_with_ngrok(app)
```

```
# Routes
@app.route("/")
def index():
    return render_template("index.html")

@app.route('/get')
def get_bot_response():
    # Getting user's question
    userText = request.args.get('msg')
    if userText=="":
        return "Please ask a question."
    if userText=='What can you do.' or userText=='What can you do?' or userText=='what can you do.'
        return "You can chat with me, ask questions and I can bring you answers from Linked Data."

    # Getting response from chatterbot
    response=Chatbot.chatbot_response(str(userText))
    if response!='I am sorry, but I do not understand.':
        return response

    # Getting response from knowledge bases
    response=KBOT.kbot_response(str(userText))
    if response=='Unable to retrieve data':
        return 'I am sorry, but I do not understand.'

    return response
```

```
[ ]  if __name__ == "__main__":
         app.run()
```

Output:

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Running on http://c046-35-245-135-254.ngrok.io
 * Traffic stats available on http://127.0.0.1:4040
```

Chapter 7: EVALUATION

7.1 Intent Classification Task

To correctly interact with the user in chatbot systems, we need to understand what the user asks for. Intent classification, (i.e., classifying the query into several categories) can suggest plausible constraints to get an answer. For example, if the chatbot understands that the query “Hello, who is Alan Turing” is about a person’s name, the confusion will be significantly reduced, and the answer will be relevant. The accuracy of the intent classification is crucial to the overall performance of the chatbot. To train the model using SVM, we follow the question taxonomy, which contains six coarse categories (i.e., entities, description, abbreviation, human, numerical, location) and 50 fine-grained question categories, as shown in Table 1. Usually, the number of question categories is less than 20. However, a fine-grained category definition is more beneficial in verifying the relevant answers. To simplify the experiments, we assume that one

Category	Fine category
ENTY	animal, body, color, creation, currency, disease/medical, event, food, instrument, language, letter, other, plant, product, religion, sport, substance, symbol, technique, term, vehicle, word.
DESC	definition, description, manner, reason
ABBR	abbreviation, expansion.
HUM	description, group, individual, title.
NUM	code, count, date, distance, money, order, other, percent, period, speed, temperature, size, weight.
LOC	city, country, mountain, other, state.

TABLE 2. The coarse and fine-grained question categories

question belongs only to one category, and the query is labeled with its most probable class.

UIUC has manually labeled the publicly available training dataset used in this paper according to the coarse- and fine-grained categories. There are about 5500 labeled questions. Our testing dataset contains 100 queries and manually labeled
according to our question hierarchy. We used Precision, Recall, and F-measure metrics for evaluation, the predicted intent was compared the labels, truth, and the evaluation result is given in Table 2. This table presents the classifier performance (precision, recall, and f-measure) using SVM; the result shows that SVM correctly identified the question classes. More specifically, SVM had 100% recall for class
type LOC and similar recall for classes like NUM, HUM, and DESC. Weighted avg computes F1 for each label, and returns the average considering the proportion for each label in the dataset. The intent classification task is sensitive to the training data related to some classes and the language model used during the training.

Classes	Precision	Recall	F1-score	Support
ABBR	1.00	0.33	0.50	3
DESC	0.54	0.95	0.69	21
ENTY	0.12	0.12	0.12	8
HUM	0.88	0.93	0.90	30
LOC	1.00	0.35	0.51	26
NUM	0.85	0.92	0.88	12
W eighted avg	0.78	0.70	0.68	100

TABLE 3. Evaluation metrics for coarse classes

7.2 Performance and Effectiveness of K-Bot

As shown in Table 3, we present the characteristics of K-Bot and some of the online available state-of-the-art chatbots (DBpedia chatbot, Open Data Chatbot, Open Data Assistant); all these chatbots make use of linked data. The first column specifies the chatbot name together with the language it processed in case of multilingual queries, Knowledge Base states for the knowledge base used to provide an answer.

Chatbot	Language	Knowledge Base
DBpedia [28]	En	DBpedia
ODC [29]	En	DBpedia
ODA [30]	En	DBpedia
KBot	En/ Fr	DBpedia, Wikidata, myPersonality

TABLE 4. Characteristics of the online available chatbots

To evaluate the performance and effectiveness of K-Bot in terms of response to user queries, we used precision, recall, and F-measure. In this case, the precision is the ratio of the number of pertinent answers found over the total number of answers found. While the recall is the ratio of the number of pertinent answers found over the total number of relevant answers.

Table 4 reports the results obtained by the participating systems on the multilingual and hybrid queries, respectively. The first column specifies the chatbot name, Total states for the number of the queries; Right specifies the number of these questions answered correctly, Wrong states for the number of the queries the system provides the wrong answer. Recall, Precision, and F-measure report the metrics concerning the evaluation of the overall system. For example, with a total of fifty-six queries, K-Bot provides nineteen relevant answers, while other state-of-the-art provide less than five relevant answers. Meanwhile, K-Bot had (0.3)

recall and outperformed other systems. These results indicate that in terms of precision, recall, and F-measure, K-Bot had better performance. The results also validate that combining NLP with linked data, especially my Personality for analytical queries, improved the performances and enabled the machine learning algorithms (SVM) to understand natural language queries better and retrieve relevant answers.

Chatbot	Total	Right	Wrong	P	R	F-measure
DBpedia	56	4	6	0.4	0.06	0.1
ODC	56	3	2	0.6	0.05	0.09
ODA	56	2	3	0.4	0.03	0.05
KBot	**56**	**19**	**3**	**0.8**	**0.3**	**0.44**

TABLE 5. Results for each of the participating systems

The evaluation results show that our proposed approach achieves much better overall performance and outperforms the state-of-the-art systems over linked data. Based on that, the integration of NLP and intent classification within the queries understanding process over linked data performs well and produces a relevant response to user queries. After eliminating the non-interesting tokens from user query using a specific parser. ‘QuestionTosparql’ module generates SPARQL queries to retrieve relevant information from available knowledge bases (DBpedia, Wikidata, etc.)

7.3 Discussion

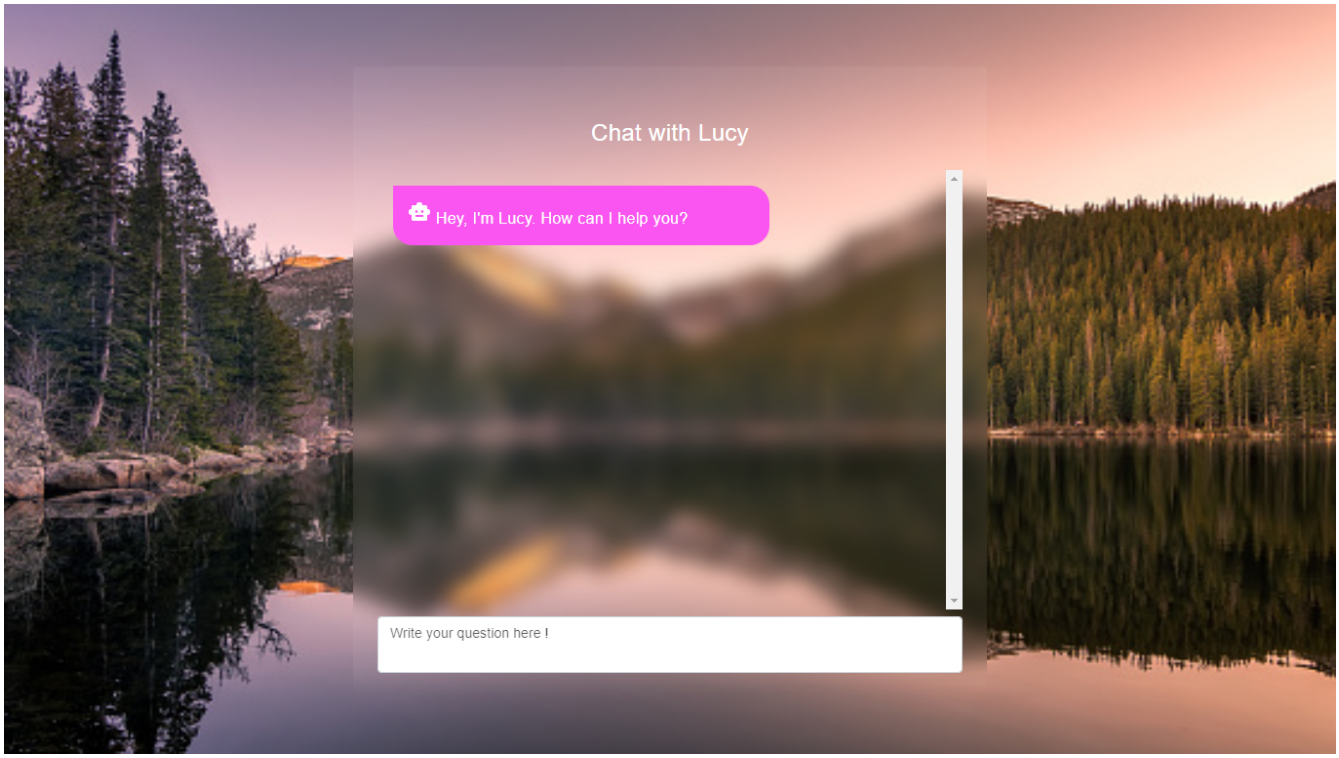
In literature, one of the main issues of linked data is insufficiently integrated chatbot and similar systems built on top of semantic web technologies; most solutions are focused on a rule and AI-based chatbots. However, it ignored personalized knowledge bases and the strength of linked data and the semantic web technologies. To overcome some of these issues, we proposed K-Bot, a comprehensive open-access chatbot by exploiting the potential of semantic web technologies (RDF, SPARQL, etc.), federated database, and natural language understanding. For instance, it supports many tasks, including dialogue management, intent classification, FAQs, question answering, analytical queries, and data exploration. K-Bot can answer some research queries and address particular use cases such as personality and sentiment analysis. In summary, K-Bot contributes to a better understanding of user queries in the context of linked data (DBpedia, Wikidata) by answering different user queries. The proposed K-Bot has the following three significant strengths:

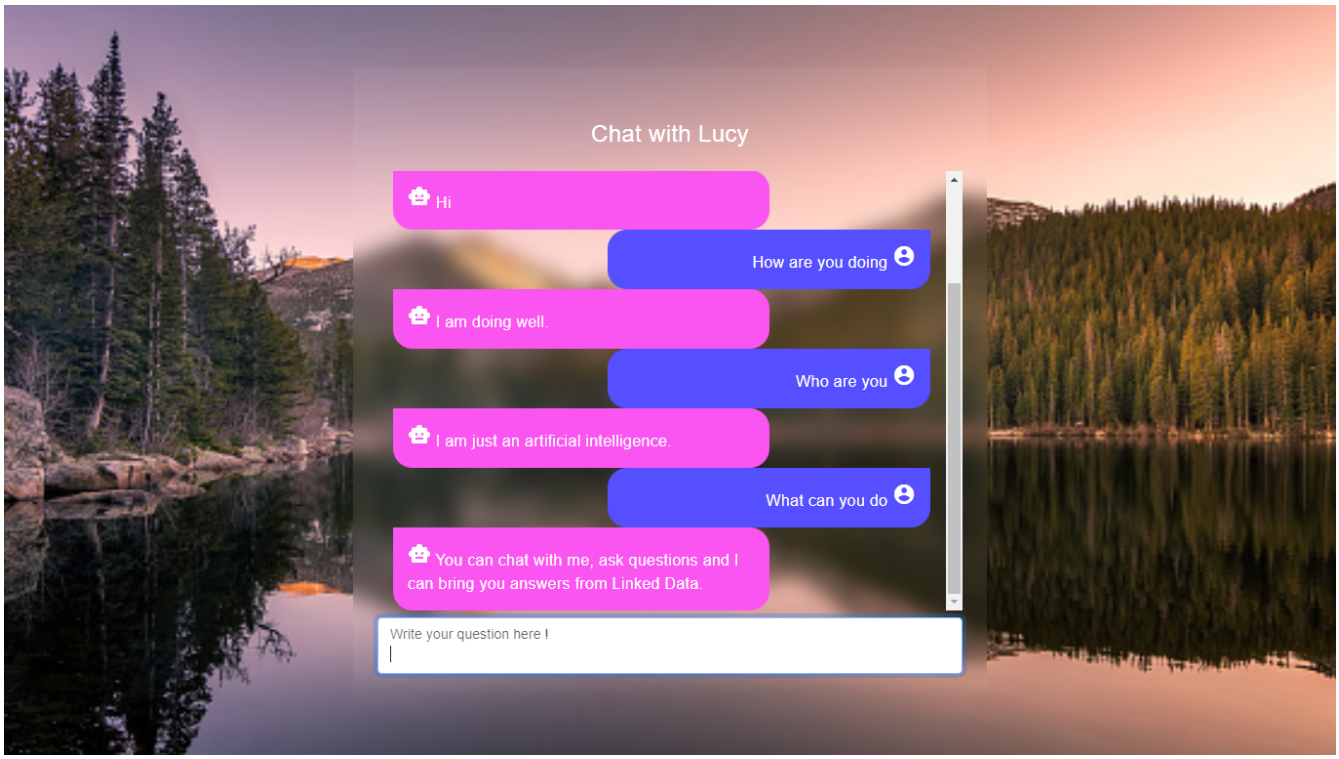
1) The overall system provides an interactive user interface for dialogue management that facilitates user interaction.

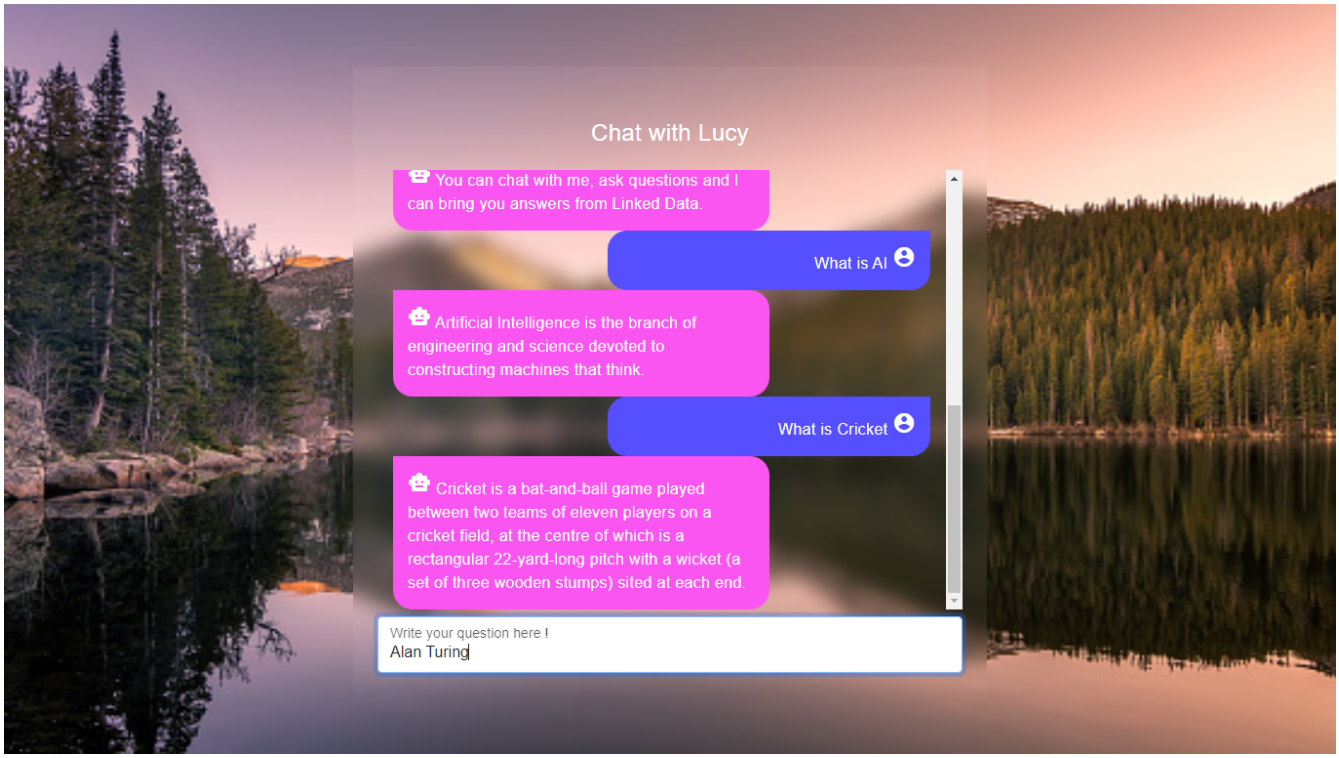
2) Multiple knowledge base and multilingualism.

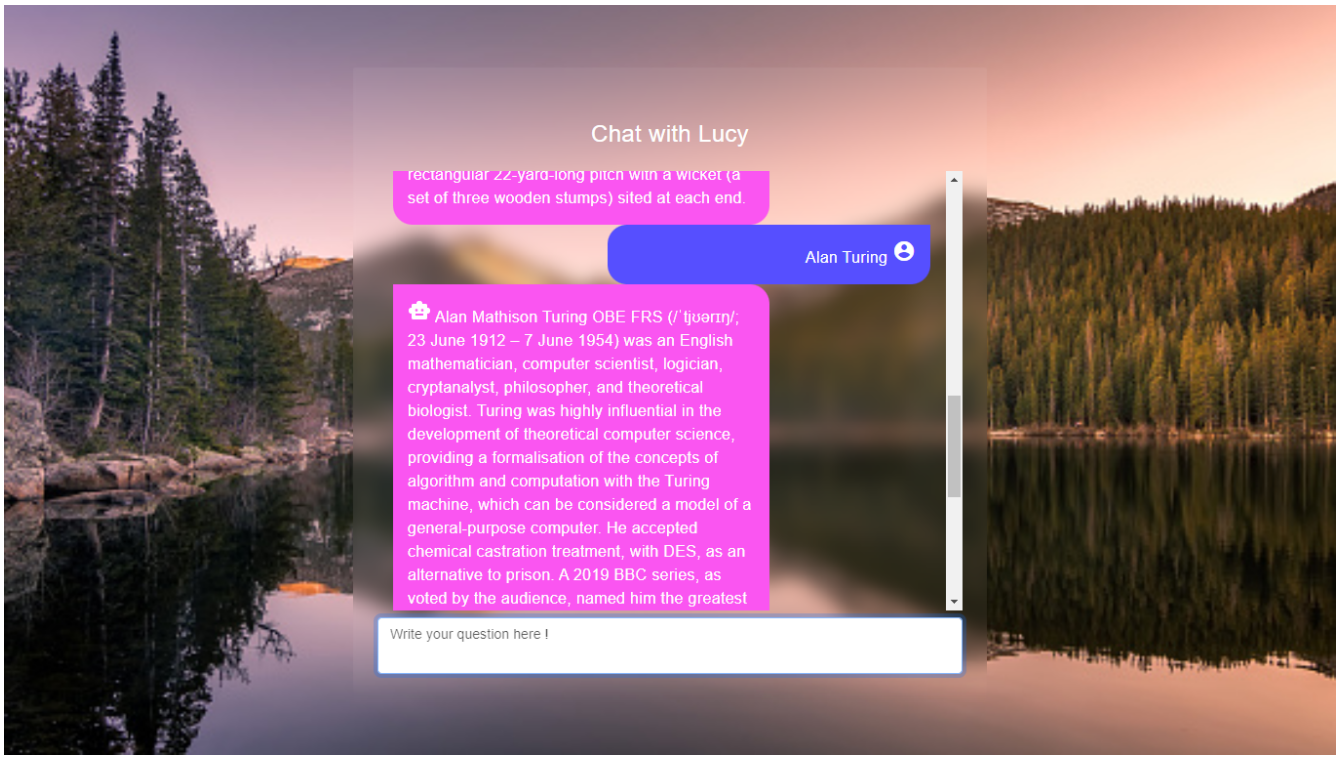
3) The overall K-Bot provides open access to involve a wide range of users.

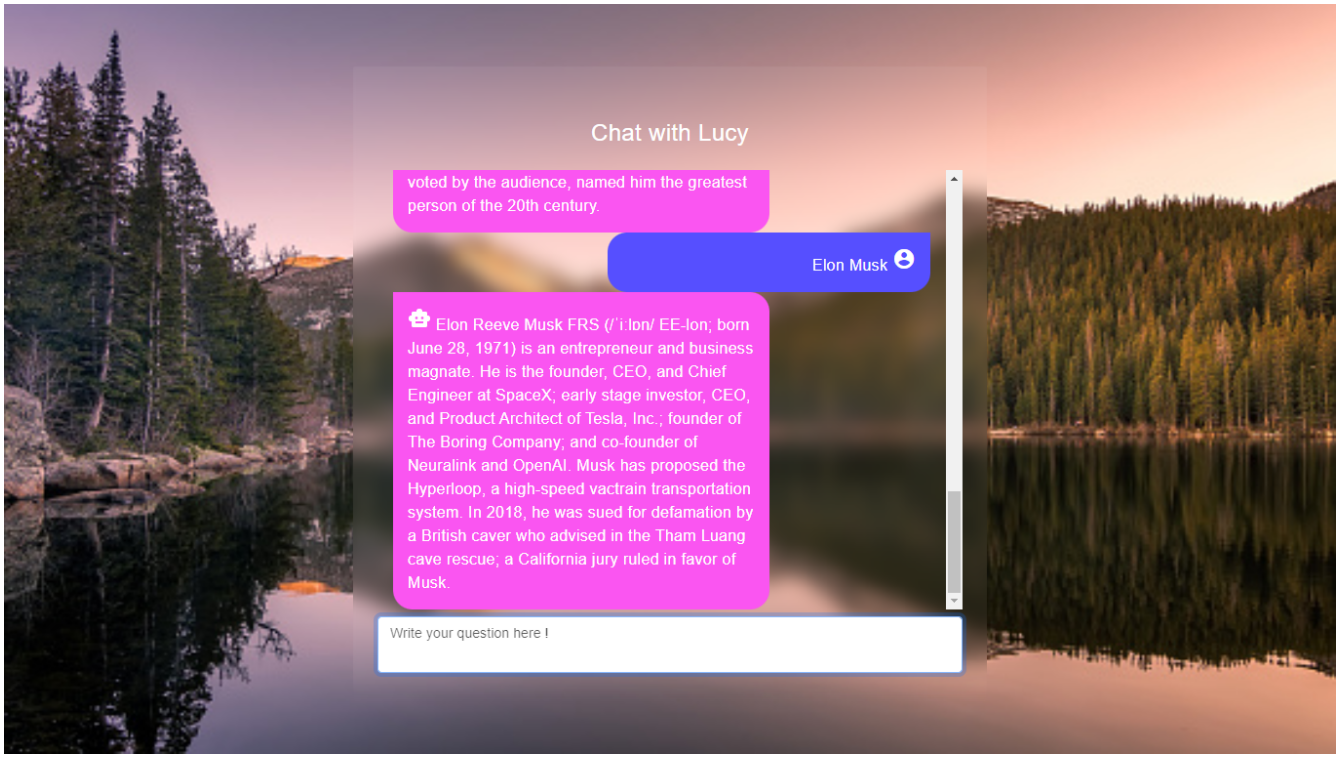
Chapter 8: RESULTS

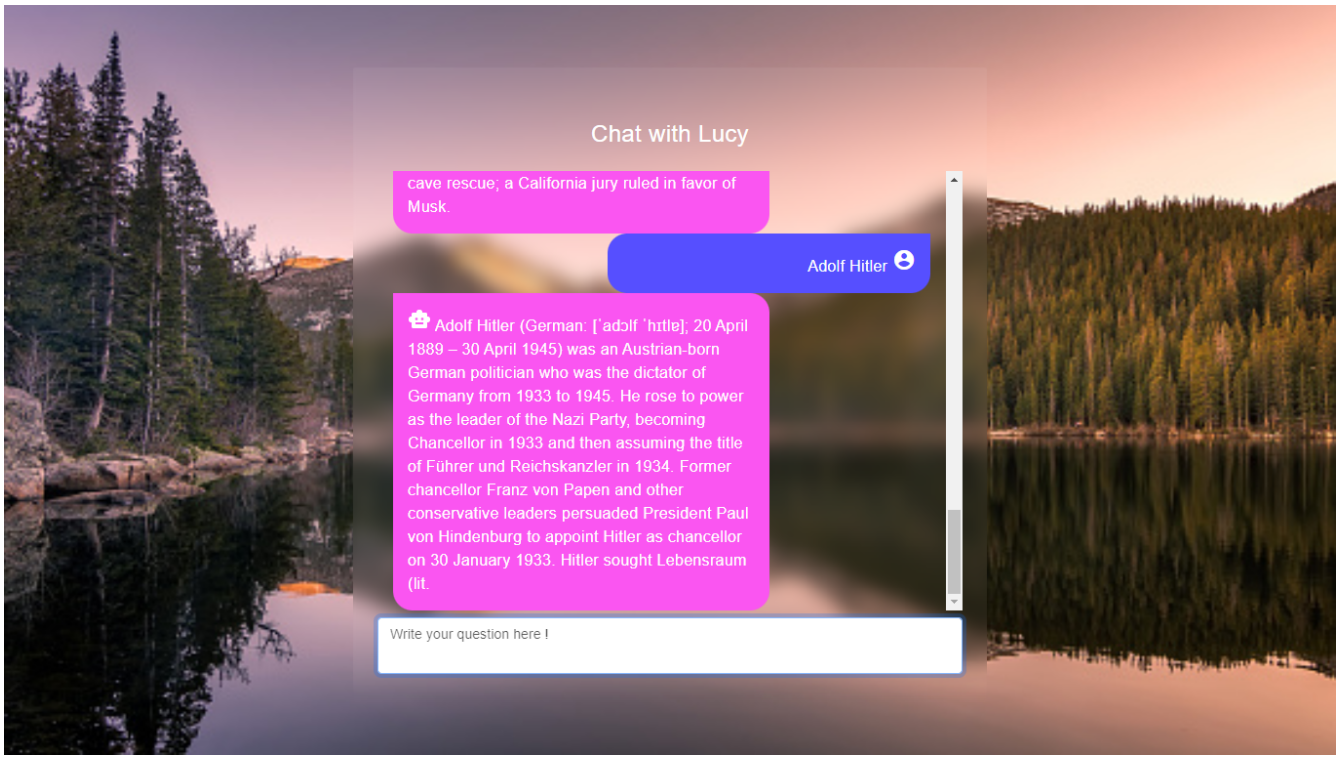





Chat with Lucy
You can chat with me, ask questions and I can bring you answers from Linked Data.
What is AI
Artificial Intelligence is the branch of engineering and science devoted to constructing machines that think.
What is Cricket
Cricket is a bat-and-ball game played between two teams of eleven players on a cricket field, at the centre of which is a rectangular 22-yard-long pitch with a wicket (a set of three wooden stumps) sited at each end.
Write your question here !
Alan Turing


Chat with Lucy
rectangular 22-yard-long pitch with a wicket (a set of three wooden stumps) sited at each end.
Alan Turing
Alan Mathison Turing OBE FRS (/ˈtjʊərɪŋ/; 23 June 1912 – 7 June 1954) was an English mathematician, computer scientist, logician, cryptanalyst, philosopher, and theoretical biologist. Turing was highly influential in the development of theoretical computer science, providing a formalisation of the concepts of algorithm and computation with the Turing machine, which can be considered a model of a general-purpose computer. He accepted chemical castration treatment, with DES, as an alternative to prison. A 2019 BBC series, as voted by the audience, named him the greatest
Write your question here !


Chat with Lucy
voted by the audience, named him the greatest person of the 20th century.
Elon Musk
Elon Reeve Musk FRS (/ˈiːlɒn/ EE-lon; born June 28, 1971) is an entrepreneur and business magnate. He is the founder, CEO, and Chief Engineer at SpaceX; early stage investor, CEO, and Product Architect of Tesla, Inc.; founder of The Boring Company; and co-founder of Neuralink and OpenAI. Musk has proposed the Hyperloop, a high-speed vactrain transportation system. In 2018, he was sued for defamation by a British caver who advised in the Tham Luang cave rescue; a California jury ruled in favor of Musk.
Write your question here !


Chat with Lucy
cave rescue; a California jury ruled in favor of Musk.
Adolf Hitler
Adolf Hitler (German: [ˈadɔlf ˈhɪtlɐ]; 20 April 1889 – 30 April 1945) was an Austrian-born German politician who was the dictator of Germany from 1933 to 1945. He rose to power as the leader of the Nazi Party, becoming Chancellor in 1933 and then assuming the title of Führer und Reichskanzler in 1934. Former chancellor Franz von Papen and other conservative leaders persuaded President Paul von Hindenburg to appoint Hitler as chancellor on 30 January 1933. Hitler sought Lebensraum (lit.
Write your question here !

Chapter 9: CONCLUSION

In this paper, we proposed a knowledge graph-based chatbot system over linked data, optimized for community interaction. The proposed K-Bot system takes advantage of large-scale, publicly available knowledge bases, multilingual, speech to text, and external APIs. Besides, K-Bot leverages the technologies of machine learning and natural language understanding, including named entity recognition, factoid, and recurrent questions, as well as dialogue management. Usability analysis shows that the proposed K-Bot has improved the end-to-end user experience in terms of interactive question answering and performance. It is more convenient for information retrieval, information acquisition, intent classification, query understanding, and continuous learning.

The future work will be adding more text-based data sources with privacy preservation, answers generation-based, extending it to more knowledge bases and other languages, and integrating with third-party services (Slack, Facebook,
Skype, etc.).

Chapter 10: REFERENCES

[1] A. M. TURING, “I.—COMPUTING MACHINERY AND INTELLIGENCE,” Mind, vol. LIX, no. 236, pp. 433–460, 10 1950. [Online].
Available: https://doi.org/10.1093/mind/LIX.236.433

[2] J. Weizenbaum, “Eliza, a computer program for the study of natural language communication between man and machine,” Commun. ACM, vol. 9, no. 1, pp. 36–45, Jan. 1966. [Online].
Available: http://doi.acm.org/10.1145/365153.365168

[3] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A training algorithm for optimal margin classifiers,” in Proceedings of the Fifth Annual Workshop on Computational Learning Theory, ser. COLT ’92. New York, NY, USA: Association for Computing Machinery, 1992, p. 144–152. [Online].
Available: https://doi.org/10.1145/130385.130401

[4] K. M. Colby, Human-Computer Conversation in A Cognitive Therapy Program. Boston, MA: Springer US, 1999, pp. 9–19. [Online].

[5] X. Li and D. Roth, “Learning question classifiers,” in Proceedings of the 19th International Conference on Computational Linguistics - Volume 1, ser. COLING ’02. USA: Association for Computational Linguistics, 2002, p. 1–7. [Online].
Available: https://doi.org/10.3115/1072228.1072378

[6] B. Heller, M. D. Proctor, D. Y.-O.-N. Mah, L. Jewell, and B. Cheung, “Freudbot: An investigation of chatbot technology in distance education,” 2005.

[7] R. chatbot. (2008)
Available: http://brilligunderstanding.com/rosedemo.html.

[8] J. Jia, “Csiec: A computer assisted english learning chatbot based on textual knowledge and reasoning,” Knowledge-Based Systems, vol. 22, no. 4, pp. 249–255, 2009, artificial Intelligence (AI) in Blended Learning. [Online].

Available: http://www.sciencedirect.com/science/article/pii/S0950705109000045
[9] H. Al-Zubaide and A. A. Issa, “Ontbot: Ontology based chatbot,” in International Symposium on Innovations in Information and Communications Technology, Nov 2011, pp. 7–12.
[10] H. Kazi, B. S. Chowdhry, and Z. Memon, “Medchatbot: An umls based chatbot for medical students,” 2012.
[11] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2, ser. NIPS’13. USA: Curran Associates Inc., 2013, pp. 3111–3119. [Online].
Available: http://dl.acm.org/citation.cfm?id=2999792.2999959
[12] B. AbuShawar and E. Atwell, “Alice chatbot: Trials and outputs,” Computación y Sistemas, vol. 19, 12 2015.
Available: https://doi.org/10.1093/mind/LIX.236.433
[13] B. Hettige and A. Karunananda, “Octopus: A multi agent chatbot,” 11 2015.
[14] B. E. Comendador, B. Francisco, J. Medenilla, S. Nacion, and T. Serac, “Pharmabot: A pediatric generic medicine consultant chatbot,” Journal of Automation and Control Engineering, vol. 3, pp. 137–140, 01 2015.
[15] D. A. Ali and N. Habash, “Botta: An arabic dialect chatbot,” in COLING, 2016.
[16] M. Qiu, F.-L. Li, S. Wang, X. Gao, Y. Chen, W. Zhao, H. Chen, J. Huang, and W. Chu, “AliMe chat: A sequence to sequence and rerank based chatbot engine,” in Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 498–503. [Online].
[17] K. Oh, D. Lee, B. Ko, and H.-J. Choi, “A chatbot for psychiatric counseling in mental healthcare service based on emotional dialogue analysis and sentence generation,” 2017 18th IEEE International Conference on Mobile Data Management (MDM), pp. 371–375, 2017.

[18] Divya, Indumathi, Ishwarya, Priya Shankari, and K. Devi, “A self-diagnosis medical chatbot using artificial intelligence,” Journal of Web Development and Web Designing, vol. 3.

[19] P. P.-T. Hsu, J. Zhao, K. Liao, T. Liu, and C. Wang, “Allergybot: A chatbot technology intervention for young adults with food allergies dining out,” in Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems, ser. CHI EA ’17. New York, NY, USA: ACM, 2017, pp. 74–79. [Online].
Available: http://doi.acm.org/10.1145/3027063.3049270

[20] L. Cui, F. Wei, S. Huang, C. Tan, C. Duan, and M. Zhou, “Superagent: A customer service chatbot for e-commerce websites,” in Proceedings of ACL 2017, System Demonstrations. Association for Computational Linguistics, July 2017, pp. 97–102. [Online].
Available: SuperAgent: A Customer Service Chatbot for E-commerce Websites - Microsoft Research

[21] A. Xu, Z. Liu, Y. Guo, V. Sinha, and R. Akkiraju, “A new chatbot for customer service on social media,” in Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, ser. CHI ’17. New York, NY, USA: ACM, 2017, pp. 3506–3510. [Online].
Available: http://doi.acm.org/10.1145/3025453.3025496

[22] S. Neumaier, V. Savenkov, and S. Vakulenko, “Talking open data,” in The Semantic Web: ESWC 2017 Satellite Events, E. Blomqvist, K. Hose H. Paulheim, A. Ławrynowicz, F. Ciravegna, and O. Hartig, Eds. Cham: Springer International Publishing, 2017, pp. 132-136.

[23] M. Qiu, F.-L. Li, S. Wang, X. Gao, Y. Chen, W. Zhao, H. Chen, J. Huang, and W. Chu, “Alime chat: A sequence to sequence and rerank based chatbot engine,” 01 2017, pp. 498–503.
Available: https://www.aclweb.org/anthology/P17-2079

[24] R. Zhao, O. J. Romero, and A. Rudnicky, “Sogo: A social intelligent negotiation dialogue system,” in Proceedings of the 18th International Conference on Intelligent Virtual Agents, ser. IVA ’18. New York, NY, USA: ACM, 2018, pp. 239–246. [Online]. Available:
http://doi.acm.org/10.1145/3267851.3267880

[25] L. Vaira, M. A. Bochicchio, M. Conte, F. M. Casaluci, and A. Melpignano, “Mamabot: a system based on ml and nlp for supporting women and families during pregnancy,” in IDEAS, 2018.
[26] C. Segura, A. Palau, J. Luque, M. R. Costa-jussà, and R. E. Banchs, “Chatbol, a chatbot for the spanish “la liga”,” 2018.
[27] R. G. Athreya, A.-C. Ngonga Ngomo, and R. Usbeck, “Enhancing community interactions with data-driven chatbots–the dbpedia chatbot,” in Companion Proceedings of the Liga Web Conference 2018, ser. WWW ’18. Republic and Canton of Geneva, Switzerland: International World WideWeb Conferences Steering Committee, 2018, pp. 143–146. [Online]. Available: https://doi.org/10.1145/3184558.3186964
[28] X.-S. Vu, A. Ait-Mlouk, E. Elmroth, and L. Jiang, “Graph-based interactive data federation system for heterogeneous data retrieval and analytics,” in The World Wide Web Conference, ser. WWW ’19. New York, NY, USA: ACM, 2019, pp. 3595–3599. [Online]. Available: http://doi.acm.org/10.1145/3308558.3314138
[29] X.-S. Vu and L. Jiang, “Generic multilayer network data analysis with the fusion of content and structure,” in Proceedings of the 20th International Conference on Computational Linguistics and Intelligent Text Processing, April, 2019, 2019.
[30] S. Keyner, V. Savenkov, and S. Vakulenko, “Open data chatbot,” in The Semantic Web: ESWC 2019 Satellite Events, P. Hitzler, S. Kirrane, O. Hartig, V. de Boer, M.-E. Vidal, M. Maleshkova, S. Schlobach, K. Hammar, N. Lasierra, S. Stadtmüller, K. Hose, and R. Verborgh, Eds. Cham: Springer International Publishing, 2019, pp. 111–115.

PROJECT REPORT

ON

K-Bot: A Knowledge graph based chatbot for natural language understanding over linked data

Submitted in partial fulfilment of requirements to

CS-452 PROJECT-2

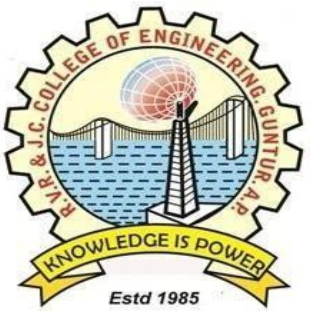
By

(Batch No: 15)

M. Datta Sai (Y18CS094)

M. Ramesh (Y18CS097)

M. Pavan Sai (Y18CS093)



NOV 2021

R.V.R & J.C COLLEGE OF ENGINEERING (AUTONOMOUS)

(NAAC - 'A+' GRADE)

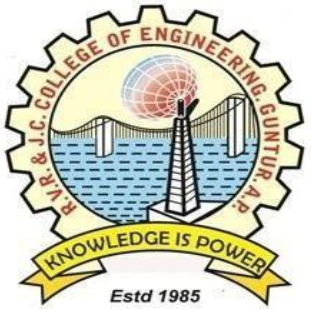
(Approved by **AICTE**, Affiliated to ACHARYA NAGARJUNA UNIVERSITY)

Chandramoulipuram, Chowdavaram,

GUNTUR – 522 019

R.V.R & J.C COLLEGE OF ENGINEERING (AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

This is to certify the project titled **"K-Bot: A Knowledge graph based chatbot for natural language understanding over linked data"** is the study of **M. Datta Sai (Y18CS094), M. Ramesh (Y18CS097), M. Pavan Sai (Y18CS093)**, submitted for partial fulfilment of requirements for the **CS 452 – Project-2** during the academic year 2021 – 2022.

Dr. B. Varaprasad Rao
Guide & Project In-charge

Dr. M. Sreelatha
Prof & HOD, CSE

ACKNOWLEDGEMENT

The successful completion of any task would be incomplete without proper suggestions, guidance and environment. Combination of these three factors acts like backbone to our project **"K-Bot: A Knowledge graph based chatbot for natural language understanding over linked data".**

We are very glad to express our special thanks to **Dr. B.Varaprasad Rao,** Guide for the project and also in-charge for the project who has inspired us to select this topic, and for his valuable advice in preparing this project topic and also his encouragement and support to carry out this project successfully.

We express our sincere thanks to **Dr. M. Sreelatha**, Head of the Department of Computer Science and Engineering for her encouragement and support to carry out this project successfully.

We are very much thankful to **Dr. A. Sudhakar,** Principal of R.V.R & J.C COLLEGE OF ENGINEERING, Guntur for having allowed us to deliver this project.

We also submit our reserves thanks to Lab staff in the Department of Computer Science and Engineering and to all our friends for their cooperation during the project.

M. DATTA SAI (Y18CS094)
M. RAMESH (Y18CS097)
M. PAVAN SAI (Y18CS093)

CONTENTS