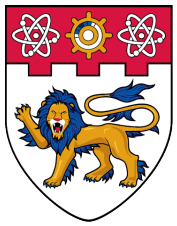
EXPLAINABLE Q&A SYSTEM BASED ON DOMAIN-SPECIFIC KNOWLEDGE GRAPH



NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

A thesis Submitted to the
School of Computer Science and Engineering
of the Nanyang Technological University

by

Xuejiao Zhao

in partial fulfillment of the requirement
for the degree of Doctor of Philosophy

March 6, 2021

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

.... 21 Aug, 2020 ....

Date

Xuejiao Zhao

Xuejiao Zhao

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

.... 21-Aug-2020 ....

Date

Chunyan Miao

Authorship Attribution Statement

This thesis contains material from 2 papers published in the following peer-reviewed journals or from papers accepted at conferences in which I am listed as an author.

Chapter 5 is published as **Xuejiao Zhao**, Zhenchang Xing, Muhammad Ashad Kabir, Naoya Sawada, Jing Li, and Shang-Wei Lin. HDSKG: Harvesting domain specific knowledge graph from content of webpages. 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 56-67. IEEE, 2017. and **Xuejiao Zhao**. Hdso: Harvest domain specific non-taxonomic relations of ontology from internet by deep neural networks (dnn). BSR winter school Big Software on the Run: Where Software meets Data, p74.

The contributions of the co-authors are as follows:

- A/Prof. Zhenchang Xing and I proposed the initial idea and designed the study together.
- I wrote the manuscript drafts. A/Prof. Zhenchang Xing, A/Prof. Shang-wei Lin and Dr. Muhammad Ashad Kabir revised the manuscript drafts.
- I performed all the laboratory work at the School of Computer Science and Engineering and the Nanyang Technological University.
- I conducted the evaluation and I mainly analyzed the data. Dr. Jing Li assisted the data analysis.
- Dr. Jing Li and Mr. Naoya Sawada assisted the creation of graph database.

21 Aug, 2020 ..... Xuejiao Zhao .....

Date Xuejiao Zhao

Acknowledgments

First and foremost, I would like to give my sincere thanks to my advisors Professor Chunyan Miao and Professor Zhenchang Xing for the continuous support of my Ph.D study. Before Professor Xing moved to the Australian National University, I got the chance from him to pursuit my Ph.D in Nanyang Technological University. I appreciate all his contributions of ideas, time and funding to make my Ph.D experience stimulating. I am grateful to Professor Miao for her continuous support of the second half of my Ph.D study. I thank her very much for her weekly meetings and carefully revised my submission. Her patient guidance helped me in all the time of my research and writing of this thesis. In addition, she also taught me a lot of life principles, so that I gradually become a calm, humble, independent thinking researcher. It is a great honour for me to pursuit my Ph.D under the supervision of the two advisors.

I am grateful to Prof.Huanhuan Chen from University of Science and Technology of China. He helped me a lot from paper writing to algorithm design, experiments, etc. I would like to thank my thesis committee: Professor Cong Gao, Professor Lin Qiu, Professor Yew Soon Ong, Professor Han Yu and Professor David Lo for their insightful comments and encouragement. The critical questions they proposed prompted me to broaden my research from different perspectives. I am fortunate to have had great colleagues and friends throughout the four years. First, I would like to express appreciation to the senior fellow apprentices: Dr.Ji Ling, Dr. Lingfeng Bao and Dr. Deheng Ye. I would like to thank my group members at the Lily research centre: Dr. Qiong Wu, Dr. Xinjia Yu, Miss Yinan Zhang, Dr. Hao Zhang, Mr. Yang Qiu, Mr. Zichao Deng, Mr. Chang Liu, Mr. Jiehuang Zhang, Dr. Qingyu Guo, Dr. Haipeng Chen, Dr. Yanhai Xiong and Miss. Zhiwei Zeng for making the group a vibrant and friendly one. Special thanks to Dr. Yonghui Xu, Dr.Budhitama Subagdja, Dr. Yundong Cai and Dr. Hangwei Qian for their patient help in revising and polishing this thesis and my oral defense, their professional comments made my thesis and oral defense a great improvement. I would like to express appreciation for my friends

at the Parallel and Distributed Computing Lab (PDCL) for their support: Dr. Ke Li, Mr. Shien Zhu, Miss. Hui Chen, Mr. Chunyun Chen and Mr. Yuming Jiang. At NTU, there are numerous other friends that I am afraid I cannot list exhaustively here. I owe all of them a big thanks!

There are some special persons who accompanied me during my Ph.D study and gave me the greatest happiness. I would like to thank Mr. Zhou and Mr. Sawada for their help and accompanying in the past. We will be good friends forever. Of course, the one I should show my greatest appreciation is Mr. Tan Wang. As my boyfriend, he is so patient, gentle and kind. We study together late every day, and then we take the last bus to leave the lab. We climb academic peaks and strive together to become who we want to be. We not only accompany each other in life, but also good partners in study. The time with his accompanying are the happiest time during my Ph.D study. He is such an outstanding and diligent researcher who has many merits that are worthy for me to learn. I cherish this rare relationship, and I hope he can give me more tolerance in the future.

Last but not the least, I would like to thank my family. I would like to express my deepest respect and gratitude to my parents - Mr.Zhao and Ms.Li - for their unconditional love, trust, and support! Their care and company are precious treasures in my life. This thesis is dedicated to them.

Xuejiao Zhao

Aug 2020

Singapore

Contents

Abstract

The rapid development of the Internet has brought an enormous amount of available information, which makes information fragment a serious problem. Traditional information retrieval (IR) systems provide a list of web sites in which the needed information may be found. But the users have to take a lot of time to digest many web pages and summarize the information they want, especially for some complex search tasks. To alleviate the problems of information fragment and accelerate IR, many research works of Question and Answering (Q&A) system attempt to assist the search engine by providing simple, accurate and understandable answers to natural language queries directly. However, without the original semantic context, these answers lack explainability that makes them difficult for users to trust and adopt.

In recent years, the knowledge graph is widely used to make explainable artificial intelligence (XAI) possible in many fields (e.g. recommendation system). Since it is a large-scale semantic network that represents knowledge by concepts and their relations, which is actually similar to the human cognitive process. Encouraged by the promising results of these fields, this thesis investigates whether and how the knowledge graph and its explainability can be leveraged to Q&A system to enhance the performance of IR.

Firstly, the existing Q&A systems lack a framework of the Q&A cognitive process based on the knowledge graph. In order to provide a human-centred explanation, the artificial intelligence (AI) system should align with the cognitive model of human and explain within the basic framework of human cognition. To equip the Q&A system with human-like cognitive capabilities, in this thesis, a brain-inspired cognitive framework of Q&A process named "*XBot*" is presented. *XBot* proposes five modules corresponding to the human cognitive process including perception, planning, reasoning, response and learning. Which is largely inspired by the literature of cognitive science. It can be used as a basis for designing a knowledge graph based Q&A system that can understand, answer questions and provide a human-centred explanation.

Secondly, the existing query representation and knowledge graph search methods are insufficient to represent and solve the complex multi-condition query, as well as explanation generation. And the features such as topological structure and indirect relations, etc. are not fully utilized in answer reasoning. In this thesis, a search engine assistant for developers named "*DeveloperBot*" based on the knowledge graph of the software engineering domain will be presented. *DeveloperBot* contains a query graph construction algorithm which splits a multi-condition query into several simple constraints, and meanwhile, determines their solving order. Then, a fast graph cyclic pruning reasoning algorithm inspired by the spreading activation model of cognitive science will be introduced. This algorithm models the constraint solving as subgraph search and decision-making process by deep neural network. In the end, the corresponding reasoning subgraph and confidence will be derived following the cognitive process as the qualitative and quantitative explanations to the final answers. These algorithms implement the *BotPerception*, *BotPlanning* and *BotReasoning* modules of *XBot* framework, respectively.

Thirdly, the existing knowledge graph extraction methods fall short of the precision and completeness of the textual knowledge extraction. And they can not extract the knowledge graph of the specified domain from the text materials as well. As a result, the scale of the extracted knowledge graph is very large and contains a lot of redundant information. In order to limit the scale of knowledge graph and accelerate the graph search, this thesis elaborates a knowledge graph extraction algorithm named "*HDSKG*" (Harvesting Domain-specific Knowledge Graph), which incorporates a dependency parser with a rule-based method to chunk the relation triples candidates (basic unit of knowledge graph) with high precision and completeness. Then it extracts novel features of these candidates to estimate their domain relevance by self-training SVM (Support Vector Machine) classifier. *HDSKG* is an implementation of the *BotLearning* module of the *XBot* framework.

Finally, to evaluate the performance and practical values of the proposed models, we apply *HDSKG* to construct a high quality knowledge graph of the software engineering domain for *DeveloperBot*. Then a prototype of *DeveloperBot* was implemented and a user study involving 24 participants was conducted. The result of user study shows that compared with just using Google, with the assist of *DeveloperBot*, users can not only find answers faster and with more accuracy, but also understand the answers more deeply. At the same time, the explanation of the answers can significantly improve the users' trust and adoption of the answers. Furthermore, the more complex the question is, the more effective the *DeveloperBot* can achieve.

List of Figures

List of Tables

Chapter 1

Introduction

This chapter will present the background and motivation of this thesis, and then briefly introduce the current work. In the end, the organization of this thesis will be presented.

1.1 Background and Motivation

"Can a machine think like a human?" This idea was proposed in the early 20th century when Alan Turing designed the Turing Machine and proposed the Turing Test [2, 3, 4]. The rapid development of automatic question and answering (Q&A) system research in recent years is a vivid practice of Turing test [5, 6, 7].

The development of Q&A system is not only an important indicator for testing the level of computer intelligence, but also has very important practical value [8]. For example, the rapid development of the Internet has brought an enormous amount of available information [9]. Search engines are the main way for Internet users to access these information [10]. Traditional search engines provide a list of web sites in which the needed information may be found. The users need to take a lot of time to digest many web pages and summarize the information they want, especially for some complex search tasks. The recent research works of Q&A system attempt to alleviate the problems of information overload by providing simple, accurate and understandable answers (also called direct answers) to natural language queries directly instead of links to sites [11, 12]. However, although the answers provided by the existing Q&A systems are simple and direct, without the original semantic context, these answers lack explainability that makes the answers difficult for users to trust and adopt [13, 14, 15, 16].

The knowledge graph is a large-scale semantic network that contains a large number of concepts and their relations, which makes explainable artificial intelligence (XAI) possible [9]. This is because the essence of the knowledge graph is to simulate the representation of the knowledge in the human brain [17]. The human cognitive process is actually a process of cognizing and explaining the world with concepts, attributes and relations [17, 18, 19]. There has been a lot of research works on the explainability of knowledge graph based system, the recommendation system is one of the most studied fields [20, 21, 22, 23]. The results of these research works show that the explanations improve the trust and adoption of users significantly [24, 20, 25, 26]. Inspired by the above works and encouraged by their promising results, this thesis investigates whether and how the knowledge graph and its explainability can be leveraged to Q&A system to enhance the performance of IR.

1.1.1 Explore the Cognitive Process of Q&A

In order to provide a human-centred explanation, the artificial intelligence (AI) system should align with the cognitive model of human and explain within the basic framework of human cognition [27, 28, 17, 18, 19, 29]. Brain-inspired Cognitive System is capable of perception, inference, and learning, etc. mimicking the cognitive mechanisms of the human brain, which can provide the AI system with human-like cognitive capabilities [30, 31, 32, 33]. But the existing Q&A systems lack a framework of the brain-inspired cognitive system of Q&A cognitive process based on the knowledge graph. So we conduct a study to investigate the content of cognitive science related to complex question perception, planning, reasoning based on semantic memory and explaining, etc. The results of this study show that the Q&A process is a typical problem solving cognitive process [34]. The explanations is about “why”, “how”, and “how confident” one decisions are made [35, 36].

In the human brain, there are various memory types (e.g., sensory memory, episodic memory, etc.) that store different contents. One of which called semantic memory stores the general knowledge of the world, concepts, rules and language that human accumulated throughout their lives in the form of connected nodes, which is similar to the structure of knowledge graph [37, 38, 39]. The Q&A process is a process that human retrieval the knowledge stored in the semantic memory [40, 41, 42]. This process including five steps: 1) Perception: encodes and explains external stimuli (query) as signals that the brain can recognize, 2) Planning:

divides a task into smaller, more manageable parts, decides the right executing order [43], 3) Reasoning: accesses the semantic memory for the answer, and this cognitive process is called spreading activation model [44], 4) Response: outputs the answer and explanations from the mouth, expression or body language [45], etc., 5) Learning: learns new knowledge by cognitive activities.

Based on the human Q&A cognitive process, a brain-inspired cognitive framework of Q&A process named *XBot* is proposed. *XBot* consists of five main modules consistent with the cognitive process of human: 1) *BotPerception* is the methods to represent the input query, 2) *BotPlanning* can divide a multi-condition query into numerous simple queries and decide the right solving order, 3) *BotReasoning* combines the logic rules and data to search the knowledge graph, make the decisions, reason a final answer, extract the explanation and compute the confidence, 4) *BotResponse* presents the answers by user interface (UI), 5) *BotLearning* is the techniques for constructing knowledge graph. *XBot* can be used as a basis for designing a knowledge graph based Q&A system that can understand, answer questions and provide human-centred explanation. It can customize in different domains which just like humans can learn various knowledge by different skills and become experts of a variety of domains.

1.1.2 Explainable Q&A System

To realize an explainable Q&A system based on domain-specific knowledge graph according to the *XBot* framework, there are still some problems to be solved. The first problem is that the existing query representation algorithms extract the entities of the query by simple token matching or entity linking [11, 12]. They are insufficient to analyze and represent the complex multi-condition query. Second, the existing knowledge graph search methods are not suitable for complex query solving and explanation generation, and the features such as topological structure and indirect relations, etc. are not fully utilized in answer reasoning.

To solve these problems, this thesis will present a series of implementations of the modules of *XBot* framework called *DeveloperBot*. It can be used as a search engine assistant to assist the answer search of the complex closed-end questions. Here closed-end questions are the questions that could be answered with a simple response, e.g., one-word answer. *DeveloperBot* is customized by loading the knowledge graph of the software engineering domain into the knowledge base of the *XBot* framework. *DeveloperBot* contains a query graph construction

algorithm which implements the functions of *BotPerception* and *BotPlanning*. Of which the *BotPerception* incorporates the Dependency Parsing into the Tree Parsing [46] to maximize the completeness of the entities and relations extraction. Meanwhile, the *BotPlanning* splits a multi-condition query into several simple constraints and determines their solving order, then constructs these ordered constraints into a multi-layer query graph for further search. In contrast, our model analyzes the expression pattern of natural language deeply and enhances the representation capacity of the complex multi-condition query. Then a fast graph cyclic pruning reasoning algorithm is presented as the implementation of *BotReasoning*. This algorithm will use the query graph to search a candidate subgraph from the knowledge graph by spread activation algorithm inspired by cognitive science [44]. Compared with the traditional candidate subgraph extraction algorithms, this algorithm can extract the candidate answers and corresponding subgraph more comprehensively. Then, the fast graph cyclic pruning reasoning algorithm will extract the candidate answers and corresponding features according to direct relation, indirect relation, the topological structure of subgraph and word embedding of the predicate of candidate answers triples, etc. Next, these features will be integrated into the decision making algorithms like Bayesian Decision Theory or Deep Neural Networks (DNN) to determine the final answers. In the end, the reasoning subgraph will be extracted following the cognitive process as the qualitative explanations of "why", "how" an answer is be recommended. And the confidence of direct answer will be computed as the quantitative explanation of "how confident" an answer is.

1.1.3 Domain-specific Knowledge Graph Construction

Different from the chatbot, Q&A system needs to give accurate answers to questions, rather than just a reasonable dialogue. Q&A system often needs to be very specialized in a certain domain (e.g., Chronic pain Q&A system, Taobao online customer service Q&A system, etc.). Furthermore, the quality and scale of the knowledge graph are directly related to the search speed and answer quality of the Q&A system. So the algorithms for high quality domain-specific knowledge graph construction is necessary.

There are many research works on knowledge graph construction including NELL [47], OpenIE [48], and Google [49], etc. But these existing methods fall short of the precision and completeness of the textual knowledge extraction. And can not extract the knowledge graph of the specified domain from the text materials as well. As a result, the scale of the extracted

knowledge graph is very large and contains a lot of redundant information. In order to limit the scale of knowledge graph and accelerate the graph search, this thesis elaborates a knowledge graph extraction algorithm named *HDSKG* (Harvesting Domain-specific Knowledge Graph) as an implementation of the *BotLearning* of *XBot* framework. *HDSKG* incorporates a dependency parser with a rule-based method to chunk the relation triples candidates (basic unit of knowledge graph) with high precision and completeness. Then, *HDSKG* extract novel features of these candidates to estimate their domain relevance by self-training SVM (Support Vector Machine) classifier.

To evaluate the performance proposed models, we apply *HDSKG* to construct a knowledge graph of the software engineering domain as the knowledge base of *DeveloperBot*. The results show that *HDSKG* can extract knowledge graph with high precision, completeness and domain relevance. For the *DeveloperBot*, it has the capacity to model and reason complex query efficiently. The result of decision making of *DeveloperBot* shows that, with topological structure, predicate similarity and other novel features of the subgraph, the Bayesian decision theory and DNN algorithm can extract the correct answers from the candidate answers with high accuracy and low confidences mean square error (MSE).

Then we implement a prototype of *DeveloperBot* and conduct a user study involving 24 participants to evaluate its practical values. *DeveloperBot* is used as a search engine assistant specific for developers for information retrieval. The results of this user study show that compared with just using Google, with the assist of *DeveloperBot*, users can find answers faster and with more accuracy. In addition, using the reasoning subgraph and answer confidence as the explanation of the direct answers can significantly improve the developers' trust and adoption to the answers. These explanations also assist the developers to understand the answers more deeply, improve the answer accuracy and form better search keywords. Furthermore, the more complex the question is, the more effective the *DeveloperBot* can achieve.

Next, we will elaborate on the key research challenges and contributions of this thesis.

1.2 Challenges and Contributions

These research background in Section 1.1 motivate the main works of this thesis. It is not-trivial to achieve an explainable Q&A system based on domain-specific knowledge graph since these exist many technical challenges. We list the challenges and contributions as follows:

- **Challenge 1**: In order to provide a human-centred explanation, the artificial intelligence (AI) system should align with the cognitive model of human and explain within the basic framework of human cognition. But the existing Q&A systems lack a framework of Q&A cognitive process based on the knowledge graph.

  Contribution 1: A brain-inspired cognitive framework of Q&A process named *XBot* is proposed, is largely inspired by the literature in cognitive science and consistent with the cognitive process of human. *XBot* can be used as a basis for designing a knowledge graph based Q&A system that can understand, answer questions and provide human-centred explanation. (These contributions relate to Chapter. 3)

- **Challenge 2**: The existing query representation algorithms extract the entities of the query by simple token matching or entity linking. They are insufficient to analyze and represent the complex multi-condition query.

  Contribution 2: Propose a query graph construction algorithm which incorporates the advantages of both Dependency Parsing and Tree Parsing to analyze the expression pattern of natural language deeply. This algorithm splits a multi-condition query into several simple constraints and construct the constraints into a multi-layer query graph, and meanwhile, determine the solving order of these constraints by dependency parsing. This query graph construction algorithm is an implementation of the modules named *BotPerception* and *BotPlanning* of *XBot*. (These contributions relate to Chapter. 1)

- **Challenge 3**: The existing knowledge graph search methods are not suitable for complex query solving and explanation generation, and the features such as topological structure and indirect relations, etc. are not fully utilized in answer reasoning.

  Contribution 3: A fast graph cyclic pruning reasoning algorithm is proposed to implement *BotReasoning*. This algorithm inspires from spreading activation model of cognitive science [44], it models the constraint reasoning process as subgraph search and decision-making process based on features like direct relation, indirect relation, the topological structure of subgraph and word embedding of the predicate of candidate answers triples, etc. This algorithm also extracts the reasoning subgraph following the cognitive process as the qualitative explanations of "why", "how" an answer is be recommended, and compute

the confidence of direct answer as the quantitative explanation of "how confident" an answer is. (These contributions relate to Chapter. 1)

- **Challenge 4**: The Q&A system needs high quality knowledge graph of a specific domain. But the existing methods fall short of the precision and completeness of the textual knowledge extraction. And can not extract the knowledge graph of the specified domain from the text materials as well.

  Contribution 4: A knowledge graph extraction algorithm named *HDSKG* is proposed. It can extract relation triples (basic unit of knowledge graph) with high precision and completeness, then select domain-specific relation triples by domain relevance computation. This algorithm is an implementation of the *BotLearning* module of *XBot* framework. (These contributions relate to Chapter. 5)

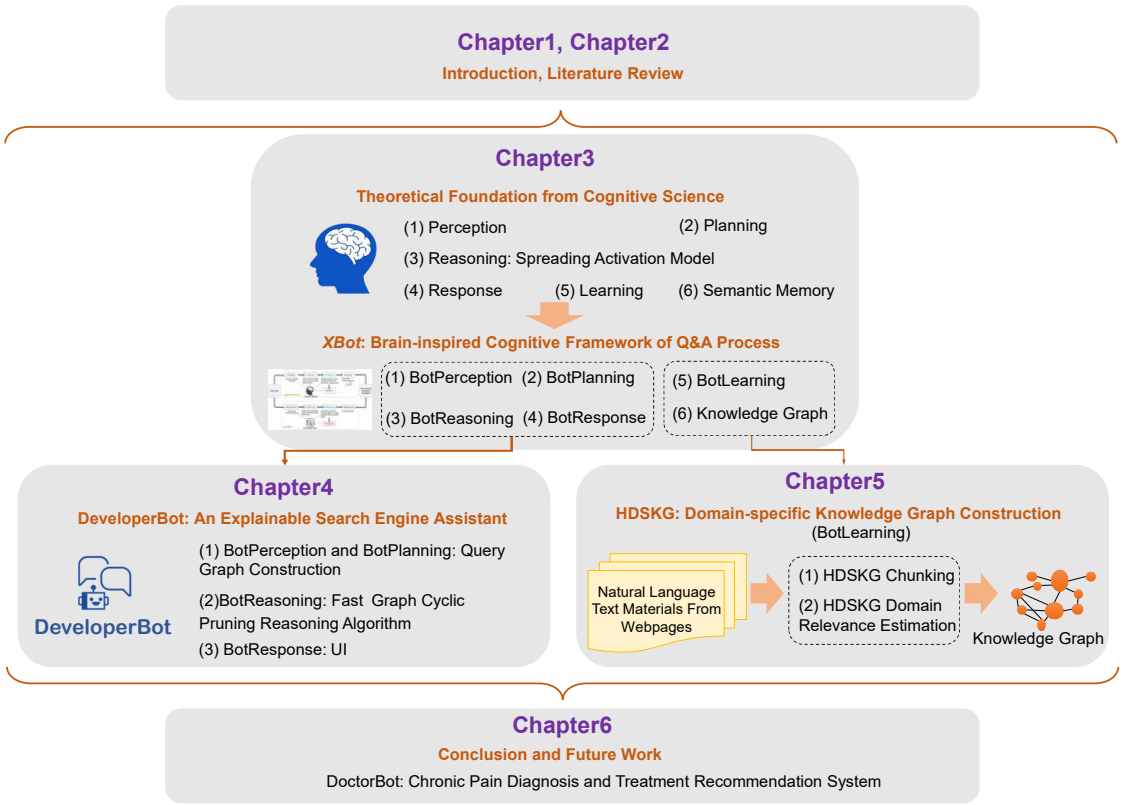


Figure 1.1: The Thesis Outline

1.3 Thesis Outline

Based on the above introduction, the outline of this thesis is shown in Fig. 1.1.

Chapter 2 In this chapter, we introduce the state-of-the-art studies of brain-inspired direct answer bot, Q&A systems based on knowledge graph and its explainability and domain-specific knowledge graph extraction techniques.

Chapter 3 In this chapter, we show the theoretical foundation from cognitive science. Then we propose a brain-inspired framework named *XBot*, which can be used as a basis for designing a knowledge graph based Q&A system that has the ability to think and answer questions like humans and have the capacity to answer questions with explainability by learned knowledge.

Chapter 4 In this chapter, we customize *XBot* to *DeveloperBot*, which is used as an explainable search engine assistant for developers and can provide direct answer and explanations to closed-end questions. This chapter is the realization of the *BotPerception*, *BotPlanning*, *BotReasoning* and *BotResponse* of *XBot*.

Chapter 5 In this chapter, we present a technique used to extract high quality domain-specific knowledge graph from unstructured text resources named *HDSKG*, which is the realization of *BotLearning* of *XBot*.

Chapter 6 In this chapter, we conclude the work and discuss possible future research directions. We also present some design and conception about the *DoctorBot* which we will focus on next.

Chapter 2

Background and Related Work

In this chapter, we will separately review existing studies about Q&A cognitive process, brain-inspired cognitive system, Q&A system based on knowledge graph and its explainability and domain-specific knowledge graph extraction techniques that are relevant to this thesis.

2.1 Q&A Cognitive Process and Brain-inspired Cognitive System

Brain-inspired Cognitive System technique is a popular research topic in recent years. These systems are capable of inference, perception, and learning mimicking the cognitive mechanisms of the brain. In this section, we reviewed the existing literature on (1) cognitive process and memory, and (2) brain-inspired cognitive system to understand current research status and indicate research gaps.

2.1.1 Cognitive Process and Memory

Artificial intelligence (AI) attempts to explore human cognitive processes by displaying "intelligent behaviour" and accomplish human tasks [50, 44]. AI researchers try to explain how humans recognize faces, recognize languages, answer questions, create mental images, write poems, and hundreds of other cognitive achievements [51]. Computers have been a popular metaphor for human thinking in recent decades [52, 53]. Computer operations are like human cognitive processes [54, 53]. To date, researchers acknowledge that computer operations currently differ from the cognitive processes and physical structure of the human brain significantly [51]. Each

metaphor has its limitations, and computers cannot accurately replicate human cognitive processes. For example, no artificial intelligence system can speak and understand languages like a human, because humans' background is much more extensive [51]. However, both humans and computers can operate on similar general principles. Researchers in artificial intelligence technology also make analogies and breakthroughs in computer technology by studying from human thinking [51, 33].

Researchers can construct computer programs to represent a suitable theory for describing human cognitive processes [54]. There are two main classifications, i.e., linear cognitive approaches and nonlinear cognitive approaches [55, 56, 57]. The information-processing method that symbolizes the linear cognitive approach emphasizes that the psychological process can be represented according to the information in a linear series of stages (one step at a time) in the system. However, this method of information processing cannot explain how we manage to perform the most complex cognitive tasks in daily life. On the contrary, the non-linear connectionist approach emphasizes the analogy of the human brain rather than serial computers as the basic model [55, 56]. This more complex design allows a connectionist approach to achieve greater complexity, flexibility, and accuracy when trying to explain human cognitive processes. The connectionist approach which includes the concept of spreading activation [44, 58] argues that cognitive processes can be understood by networks that link neuron-like units together. In addition, many operations can be performed simultaneously instead of all at once. In other words, human cognition is usually parallel rather than strictly linear [52, 59]. The commonly used interchangeable names for connectionism are the parallel distributed processing (PDP) method and the neural network method.

There were numerous studies about human memory in computer science [60, 61, 62]. Memory research showed that human memory mainly includes sensory memory, short-term memory (working memory), and long-term memory [63]. The three types of memory have mutual influences, will be lost, and will be due to unknown reasons Recall some information from Long-term memory. Besides, people will recall information better if it is specific rather than abstract, and attempts to determine whether the information applies to themselves [64, 52]. The application of cognitive processes and memories in computer science has been widely discussed. The cognitive processes and memories of the human brain are quite complex. Cognitive scientists and computer scientists are still exploring related knowledge. Many cognitive and memory

mechanisms have not yet reached a final consensus. However, some generic models have been widely accepted. Therefore, the framework and system of this research are inspired by these models.

2.1.2 Brain-inspired Cognitive System

As mentioned in the previous paragraph, in the computer science community, the cognitive mechanism of the human brain is an essential context in the development of computer science. Computer programs are designed to replicate human cognitive processes through analogies of the human brain [51]. The cognitive process of the computer analogy to the human brain is still widely discussed in the computer science and technology community, and it is also the mainstream direction of computer science development. In particular, the rise and development of artificial intelligence cannot be separated from the nourishment of neuroscience achievements [65, 66]. In recent years, advances in brain and neuroscience and cognitive science have made it possible for people to obtain partial activity data of brain issues in various cognitive tasks observed at different scales, such as brain regions, neural microcircuits, and neurons [33].

The working mechanism of the human brain obtained through multidisciplinary cross-cutting and experimental research is more reliable. Therefore, brain science is expected to provide a reference for breakthroughs in machine learning and brain-like computing. Under such a research context, the development of the brain-inspired cognitive system is continuously moving forward in various fields of computer science [33]. For example, paper [52] has proposed a generic framework of the brain-inspired system which was entitled “Self as Interacting Memory Systems” and was applied in a self-aware conversational robot. This kind of brain-inspired cognitive system can learn from the human cognitive system and deeply understand the primitive abilities of the brain [65, 52] to achieve advanced machine logic capabilities. Humans have the ability to learn how to learn. If the Brain-inspired cognitive system can learn how to learn and can imagine and plan, then it may create something that we humans can hardly create.

Brain science helps artificial intelligence, and through the brain-inspired cognitive system, there are also several directions in the major technical areas of artificial intelligence, such as implementing large-scale parallel neural networks, constructing a new type of learning theory using statistical correlation and feature correlation, developing deep learning and reinforcement learning for explainable recommendation [67], and exploring generic-purpose artificial

intelligence systems with self-learning capabilities for problem-solving [68, 65]. By reviewing related works, there has been relatively little research conducted on a complete framework and mechanism of Brain-inspired cognitive system for Q&A system in the field of natural language processing [32].

2.2 Q&A System based on Knowledge Graph and Explainability

Knowledge graph (KG) has been widely used in Q&A system [69], recommendation system and search engines in recent years due to its excellent knowledge representation capacity. Many research works on recommendation systems have taken advantage of the explainability of knowledge graph to improve the users' trust and adoption prominently [24, 20, 25, 26]. Zhang & Chen summarizes explainable recommendation applications in different domains [20], i.e., e-commerce, point of interest, social and multimedia. Of which Sayed & Al Muqrishi propose an IBRI-CASONTO semantics-based search based on ontological graph [70]. Ai et al. present an Indri system which is an explainable recommendation Q&A system to support modern language technologies [21]. Murdock & Tesauro et al. represent the statistical approaches to natural language Q&A system in the famous IBM Watson [71]. Lee et al. proposed a KG-based recommendation system for customer complaint handling [72]. Catherine, Rose, et al. [22] and Yu & Ren et al. [23] propose to use knowledge graph entities and meta-path as explanation of recommendations. There are relatively few research works of Q&A system explored to improve user trust and adoption by using the explainability of the knowledge graph. The content of the explanations of this thesis is partially inspired by the existing research works of recommendation systems.

The recent research works of Wang & Zou et al. [11] and Zhu & Ren et al. [12] are the two closest works to ours. Both their works and ours propose to assist the information needs of users expressed in natural language by knowledge graph based Q&A system. However, their works extract the entities of the query by simple token matching or entity linking. There are also some research works focus on complex question representation. Lan& Zou et al. [73] proposed to incorporate constraints and extend relation paths at the same time, to handle questions with constraints and questions with multiple hops of relations at the same time. In contrast, our

model analyzes the expression pattern of natural language deeply and construct the query into a multi-layer query graph for further solving. Which enhances the representation capacity of the complex multi-condition query. Next, although all of these works extract an inference subgraph for further reasoning, this thesis extracts the subgraph by spread activation model inspired from cognitive science [44], which can extract the candidate answers and corresponding subgraph more comprehensively.

In addition, the Q&A system of this thesis also considers more comprehensive features of the subgraph, such as indirect relation, topological structure, predicate similarity, etc., some of the features are inspired from the previous works. Furthermore, this thesis integrates all the features into a decision-making algorithm, which can predict the correct answers from all candidate answers with high accuracy. This also allows my work to quantify the confidence of the correct answers. In addition, my work presents the reasoning subgraph and the confidence as explanations of the correct answer, according to the result of the experiments, this significantly improves the answer adoption and user confidence.

The knowledge graph is also widely used in the software engineering domain to assist the information needs, development and debugging, etc. Sun & Xing et al. [74] propose open information extraction techniques to construct a task-oriented knowledge graph named *TaskKG* which enables activity-centric knowledge search. Xu and Xing et al. [75, 76] present a technique named *AnswerBot* to summarize the key points of answer posts of a technical question. Which help developers to capture the key points quickly without reading the details of the posts. From the above literature, the knowledge graph is quite useful in many areas of the software engineering domain. But as far as we know, using Q&A system with explainability based on knowledge graph as a search engine assistant has not been attempted before.

2.3 Domain-specific Knowledge Graph Extraction

Automatic knowledge graph construction is a popular research topic. According to the knowledge graph scale, extracted methods, data source and application domain, etc., we divide them into four categories.

2.3.1 Fixed Relations Extraction Techniques

The first category extract fixed relations using information extraction techniques to construct knowledge graphs. Fixed relations extracted technique extracts the relations from the text by the fixed relation vocabulary. The typical techniques such as CiteSeerX [77], DIG [78], Pujara et al. [79], and NELL [47], etc. focus on small-scale fixed relations extraction with high precision. Deepdive [80] extract relations by Markov Logic Networks (MLNs) and improve the extraction by the bootstrapping system [81]. Bootstrapping technique is similar to self-training in our system. There are also some relation classification methods based on deep learning techniques, such as CNN [82] and LSTM [83], and more recently work like Transformer [84] and Bert [85].

2.3.2 Open Information Extraction Techniques

The second category is open information extraction technique. Compared with fixed relations extracted techniques, these techniques do not use per-relation training data and not bound by the fixed relation vocabulary. It does not need to appoint some fixed relation in advance.

Abebe and Tonella et al. [86] presented a semi-automated approach to construct domain concept ontology from source code identifiers. OLLIE proposed by Michael Schmitz et al. [87] is the first rule-based open information extraction technique to extract not only verb-based relations. It used the technique called bootstrap learning of patterns to learn a set of extraction pattern templates by dependency parsing. There are also some other ruble-based representative techniques like Prismatic [88],PROPS [89] and PredPatt citewhite2016universal, etc.

There are also some research works focused on optimization algorithms for open information extraction technique. Cetto et al. [90] proposed Graphene which leveraged simplification rules to transfer complex text sentences into clean, compact structures. Fader et al. [91] used two simple syntactic and lexical constraints to handle the uninformative and incoherent information extraction problem.

HDSKG belongs to this category and the *HDSKG* chunking is inspired by some methods of these techniques. And *HDSKG* further adds the function for domain relevance estimation.

2.3.3 Extract Knowledge Graph by Structured Data Sources

The third category leverage structured data sources to build up the knowledge graph. Compared with the extracted knowledge graph from text data, these techniques mine the structured data

from the crowdsourcing like categories of Wikipedia and construct knowledge graph based on these data. YAGO [92] and YAGO2s [93] is a structured knowledge base which automatically extracted from Wikipedia with the data e.g., categories, redirects, infoboxes, etc. The knowledge graph of YAGO [92] and YAGO2s [93] is built by defining entity classes from the conceptual Wikipedia categories. As of 2012, YAGO2s has more than 10 million entities and 120 million relations about these entities, the accuracy is above 95% which is checked manually. But the scale and content of these knowledge graphs rely on the structured data of crowdsourcing materials.

2.3.4 Extract Knowledge Graph in Software Engineering Domain

The knowledge graph is also widely researched in the software engineering domain. Padhye et al. [94] propose a technique to extract the profiles of the usage of API to connect people, projects and libraries in a network. Subramanian et al. [95] link source code examples to API documentation by extraction of entity linking. To the best of our knowledge, my work is the first attempt for mining knowledge graphs which contain relation description for each entity from Q&A website.

2.4 Summary

This chapter has summarized the existing research works on the Q&A cognitive process, the brain-inspired cognitive system, the Q&A system based on knowledge graph and its explainability and the domain-specific knowledge graph extraction techniques that are relevant to this thesis.

Compared with the line of existing works on brain-inspired cognitive system, our work proposes a brain-inspired cognitive framework of Q&A process named *XBot* which can be used to explainable Q&A system based on the knowledge graph.

Compared with the line of existing works on Q&A system, the *BotPerception* and *BotPerception* modules of *DeveloperBot* fill the gaps of complex multi-condition query representation. And the *BotReasoning* module of *DeveloperBot* solves the issues of state of art research works on query reasoning and human-centric answer explanation generation.

Compared with the line of existing work on knowledge graph extraction technique, our work called *HDSKG* focuses on the improvement of extracted precision and completeness, and further selects domain-specific relation triples by domain relevance computation.

Chapter 3

XBot: a Brain-inspired Cognitive Framework for Q&A System

In order to provide a human-centred explanation, the AI system should align with the cognitive model of human and explain within the basic framework of human cognition. Brain-inspired Cognitive System is capable of perception, inference, and learning, etc. mimicking the cognitive mechanisms of the human brain, which can provide the AI system with human-like cognitive capabilities. But there has been relatively little research conducted on a complete framework and mechanism of the brain-inspired cognitive system for Q&A process. In this chapter, the content related to Q&A in the field of cognitive science is explored, such as question perception, planning, and reasoning based on semantic memory [42, 96, 43, 44]. The results show that the cognitive process of the human brain for these kinds of advanced tasks is complex and the cognitive scientists do not yet reach a final consensus [97]. But some of the research achievements have made it possible to connect question answering to the cognitive process. One of the popular models is the spreading activation model proposed by Collin and Loftus in 1975 [44]. This chapter summarizes the cognitive process that human performs question reasoning and incorporate the basic principle of spreading activation model, and proposes a brain-inspired framework named *XBot* which is consistent with the cognitive process of human. *XBot* can be a basis for designing the Q&A system. Compared with the existing Q&A systems based on knowledge graph, *XBot* based systems have the capacity to answer questions with human-centred explainability by learned knowledge. What kind of questions *XBot* can answer depends on what knowledge it has learned.

The rest of this chapter is organized as follows: In the Section 3.1, we introduce the background and motivation. Section 3.2.1 and Section 3.2.2 show some knowledge of cognitive

science domain. Some of them are existing knowledge, others are summarized, refined and re-organized from the literature of cognitive science domain. Section 3.2.3 shows the *XBot* framework and how *XBot* framework is aligned with the cognitive process of Q&A.

3.1 Background and Motivation

Several recent research works have proposed the desirable properties and criteria of human-centred explanations produced by AI systems [27, 35]. Most of these research works bring up the idea that in order to provide a human-centred explanation, the AI system should align with the cognitive model of human and explain within the basic framework of human cognition [27, 28, 17, 18, 19, 29].

Brain-inspired Cognitive System (BICS) is one of the popular research directions recently [33]. These systems are capable of perception, inference, and learning, etc. mimicking the cognitive mechanisms of the brain, which can provide the AI system with human-like cognitive capabilities [30, 31, 32, 33]. There have been many studies in this direction, such as the research on cognitive attention for self-driving cars [98], the research on knowledge learning and representation methods for cognitive robot [30], etc.

To realize a Q&A system based on the knowledge graph which can provide human-centred explanations, this chapter is concerned with a much less explored issue: the brain-inspired cognitive framework of the Q&A system based on the knowledge graph. The purpose of this exploration is to propose a framework for the Q&A system based on the knowledge graph, which aligns the human cognitive model and can augment the answer of the Q&A system with human-centred explanations. Therefore, a study is conducted to investigate the content of cognitive science related to complex question perception, planning, reasoning based on semantic memory and explaining, etc. The results of this study show that the Q&A process is a typical problem solving cognitive process of the human brain which searches for solutions to a problem or finds the path to achieve a goal [34]. After the problem is identified, the process of problem solving can be regarded as a search process in the memory to find the relations between the problem and the target solutions [34].

In the human brain, there are various memory types (e.g., sensory memory, episodic memory, etc.) that store different contents. One of which called semantic memory stores the general

knowledge of the world, concepts, rules and language that human accumulated throughout their lives in the form of connected nodes [37, 38, 39]. The Q&A process is a process that human retrieval the knowledge stored in the semantic memory, which is similar to the answer search in knowledge graph [40, 41, 42]. Because the essence of the knowledge graph is to simulate the representation of the knowledge in the semantic memory in human brain [40, 41, 42]. For example, the knowledge "Fish and bird are animal, the fish can swim and the bird can fly." are stored in the semantic memory by five nodes: fish, bird, animal, swim and fly. These five nodes are connected by two hierarchical relations: fish and bird are subclasses of animal, and two concept property relations: fish can swim, the bird can fly. In a knowledge graph, the concepts are represented as entities, and the hierarchical relation is the relation between two entities which connect by a predicate phrase "is_a". The concept-property relation is represented as a relation triples (subject, predicate, object). Here the subject and the object are entities and the predicate links the subject to the object [99].

In this chapter, a brain-inspired cognitive framework of Q&A process named *XBot* is proposed. *XBot* consists of five main modules aligned with the human Q&A cognitive process: (1) *BotPerception* is the methods to represent the input question, which corresponds to the *Perception* module for encoding and explaining the external stimuli of the human Q&A cognitive process, (2) *BotPlanning* can divide a question into smaller queries and decide the right solving order, which corresponds to the *Planning* module of human, (3) *BotReasoning* is the process to search the knowledge graph, reason the answers, make the final decisions, extract the reasoning subgraph and compute the confidence. This corresponds to the *Reasoning* module which accesses the semantic memory of human brain for the answer reasoning, (4) *BotResponse* presents the answers and explanations by UI which likes the *Response* module of human to express by mouth, eyes, etc. (5) *BotLearning* is the techniques for construction knowledge graph which corresponds to the *Learning* process to learn knowledge in semantic memory of the human brain. *XBot* can be used as a basis for designing a knowledge graph based Q&A system that can understand, answer questions and provide a human-centred explanation. It can customize in different domains which just like humans can learn various knowledge by different skills and become experts of a variety of domains.

3.2 *XBot* Framework

This section will elaborate on the *XBot* framework and its behind theoretical basis. The Section 3.2.1 will present how a knowledge graph database of an AI system can be mapped to semantic memory of the human brain. Section 3.2.1 and Section 3.2.2 will present the results of literature exploration related to the cognitive process of Q&A of the human brain. And then, Section 3.2.3 will show the *XBot* framework and how it aligns to the cognitive process of Q&A.

3.2.1 Semantic Memory and Knowledge Graph

Memory is the ability to encode, store, retain and subsequently retrieve information and past experiences in the human brain [56, 100].

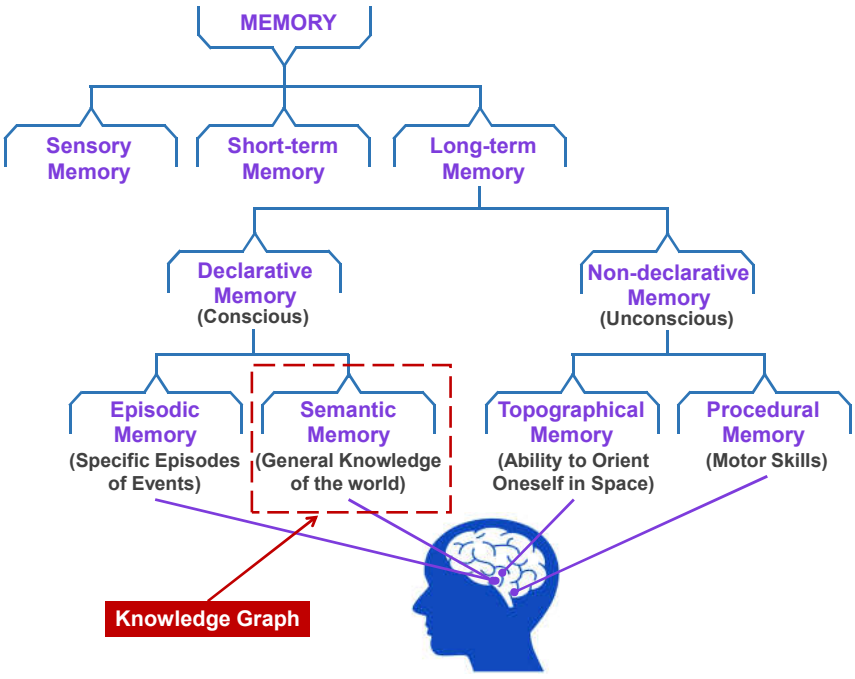


Figure 3.1: Types of Memory and Knowledge Graph

There are three main forms of memory storage: sensory memory, short-term memory and long-term memory as shown in Fig. 3.1. Sensory memory retains impressions of sensory information less than one second after the original stimulus has perceived [101, 100].

Short-term memory enables the brain to hold a small amount of information(about 7 +/- 2 pieces) for a very brief time [102, 103]. Long-term memory can store an infinite amount of

information for a long time [104, 105].

Declarative memory and non-declarative memory are two different types of long-term memory. Non-declarative memory is the knowledge that cannot be accessed through conscious processes, such as the ability to orient oneself in space, motor skills, etc [106, 107, 108, 109]. In contrast, declarative memory is the knowledge that can be accessed through a conscious process [110, 109].

Declarative memory can be further broken down into two subtypes: semantic memory and episodic memory. Episodic memory is the memory of experiences and specific episodes of events that occur at a certain time and place in coordinates of time and space [111, 112, 37, 38]. Semantic memory refers to the general knowledge of the world, concepts, rules and language that human accumulated throughout their lives [37, 38, 39]. This memory encodes abstract knowledge about the world, such as "Ottawa is the capital of Canada".

According to the previous studies of cognitive science domain, general knowledge is stored in the semantic memory of our brain in the form of connected nodes. These nodes represent different concepts of knowledge, and they are connected by relations of these nodes. There are two kinds of relations, one is hierarchical relations which indicate the concepts were organized from higher order categories down to lower order categories [113]. For example, a concept "Animal", the subclass is "Bird" or "Fish" as shown in Fig. 3.2. And "Fish" might be further linked to "Carp", "Salmon", etc. Except for hierarchical relations, there are other relations called concept-property relation which store properties and characteristics of concepts at each node [113]. For the node "Salmon", the concept property should be like "lives in the ocean", "are born in fresh water", etc.

According to the economics of cognitive efficiency, a concept property will be linked to the highest possible class. For example, the brain links property "can swim" to the node "Fish" instead of linking to every subclass of "Fish". Which indicates that all the subclass of "Fish" have the concept-property relation "can swim".

Knowledge graph in this thesis is a technical implementation of semantic memory. As shown in Fig. 3.2, in a knowledge graph, the concepts are represented as entities, and the hierarchical relation is the relation between two entities which connect by a predicate phrase "is_a". The concept-property relation is represented as a relation triples (subject, predicate, object). Here the subject and the object are entities, and the predicate links the subject to the object [99].

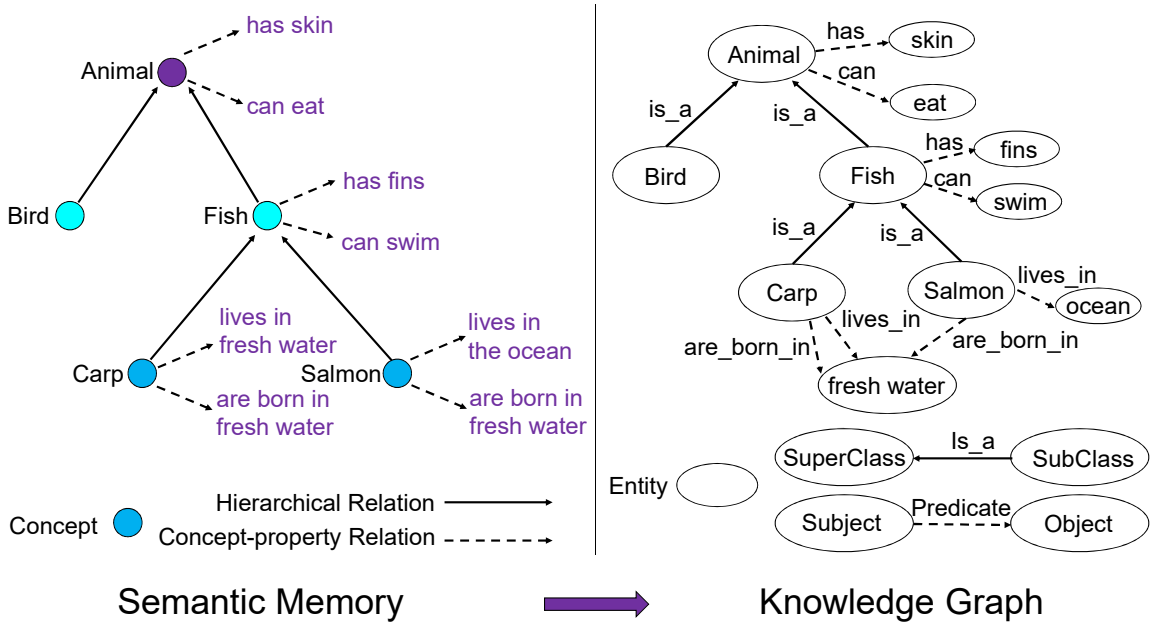


Figure 3.2: Implement Semantic Memory by Knowledge Graph

The relation (entity1, is_a, entity2) means that entity1 is the superclass of entity2, so entity2 inherits all the property relation triples of entity1. For example, although there are no relation triples (Salmon, has, fins) and (Salmon, can, swim) in the knowledge graph of Fig.3.2, these two should be inherited from the superclass of "Salmon".

3.2.2 Cognitive Process of Question Reasoning

Question reasoning is an advanced cognitive process of looking for answers according to some premises [114]. In order to perform this cognitive process, human has to access the knowledge already stored in their memory [40, 41, 42].

Assume a query is a stimulus from the external environment, we review a lot of literature on the human brain for decision making, logical reasoning, task planning, etc., and summarize several key steps related to the cognitive process of query reasoning as follows:

1) Sensation and Perception: the stimuli first reach our senses like version, hearing, etc., and then perception takes over and we start interpreting these stimuli [96]. These processes encode external stimuli as signals that the brain can recognize.

2) Planning: planning is the mental process to divide a task into smaller, more manageable parts, decide the right executing order, assign each task to the proper cognitive resources, and establish a plan of action [43]. Essentially, planning is a process by which the neurons of the mid-dorsolateral frontal cortex of human brain establish new paths or synaptic connections [43].

3) Reasoning: when a query task like "If the wild duck is a bird?" begins to be executed, the brain will access the semantic memory for the answer and this cognitive process is called spreading activation model [44]. The process of spreading activation model can be divided into two parts called the search process and decision-making process:

   - Search process: when a concept is stimulated, the corresponding concept node of semantic memory will be activated. Then the activation will spread to the peripheral nodes directly connected to the original node parallelly and then spread to various links along with the peripheral node. The more downward, the weaker the intensity, until no reaction at the end. The strength of a connection is proportional to the frequency of a person uses or thinks about. Activations from different concept spread along different lines. When these activations cross at a certain node and the sum of activations reaches the activity threshold, the network path (called subgraph) that generates this crossover will be further evaluated in the decision making process.
   - Decision-making process: the search process may get one or more crossovers, so the decision-making process will collect evidence based on subgraphs extracted by the search process to decide whether or not the concepts of the crossovers are correct answers. The superordinate connection, property comparison matching property and Wittgenstein strategy matching property found in subgraph constitute the positive evidence. And the negative superordinate connection, property comparison distinguishing property and Wittgenstein strategy distinguishing property mutually exclusive subordinates and counterexamples found in subgraph constitute the negative evidence [44]. Then, the positive evidence and negative evidence from different paths in a subgraph will be summed together to make a final decision that the answer exceeds either a positive or a negative criterion.

4) Response: when the query gets the answers, the response will output from the mouth, expression or body language, etc [45]. Unlike the responses of traditional machine learning systems, brain-inspired cognitive systems require that the responses should be explainable and can provide confidence for every response like the response given by a human.

5) Learning: learning is the end of a series of cognitive processes. These cognitive processes refer to any cognitive activities which can learn new knowledge like book reading, Q&A process, etc. New knowledge will be generated after the information processing of human cognitive processes. So the knowledge of semantic memory will be updated after the cognitive processes and this new knowledge will affect cognition in the future [115, 38, 56].

3.2.3 *XBot* Framework Aligned with Cognitive Process of Q&A

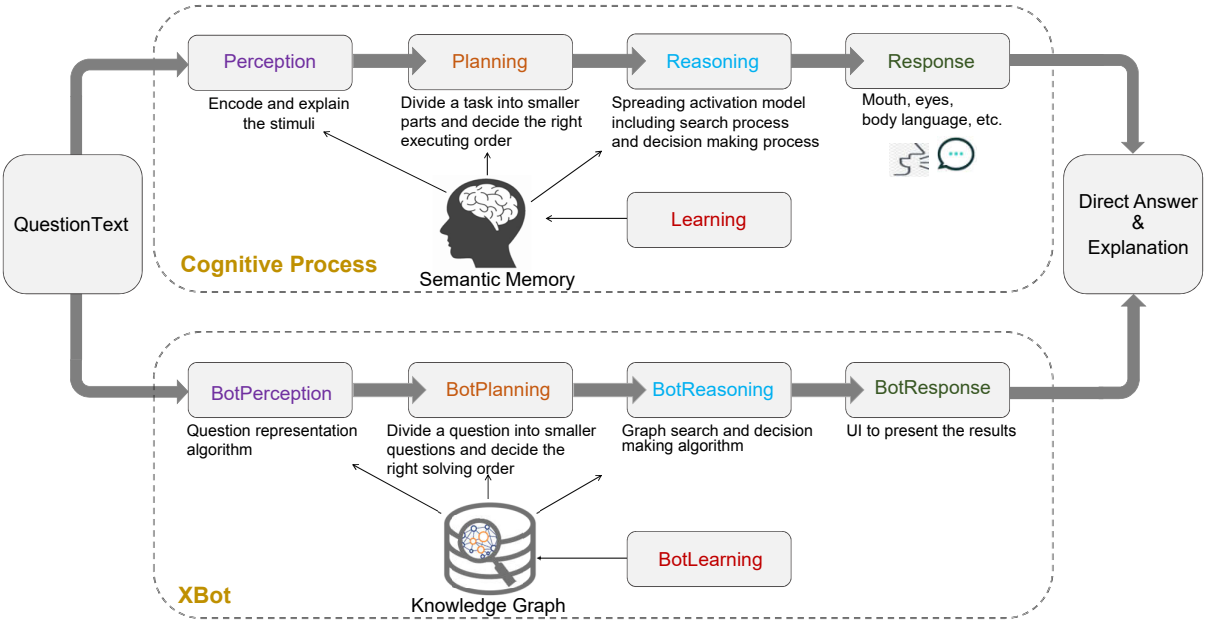


Figure 3.3: *XBot* Framework Aligned with Cognitive Process of Q&A

The lower half in Fig. 3.3 is the *XBot* framework. The upper half in Fig. 3.3 is the cognitive process that human perform intelligent tasks like question reasoning. Fig. 3.3 shows that how *XBot* emulates and aligns the cognitive process of the human brain step by step.

In *XBot*, the question representation algorithm of *BotPerception* emulates the process of perception of the cognitive process to encode and explain the stimuli. *BotPerception* will represent the question in a most suitable form which is determined by the specific scenarios. Just like the planning of the cognitive process, *BotPlanning* algorithm divides a question into smaller questions and decide the right solving order. The *BotReasoning* is the mimicking of spreading activation model of reasoning of cognitive process [44]. It contains the algorithm to search the knowledge graph and make the decision. Depends on different data, scenarios and questions, the graph search strategies and decision-making algorithm should also be different. The UI (user interface) of *XBot* is similar to the function of the mouth, eyes, body language, etc. of the cognitive process.

For example, for the question "Which graph databases support Python and can be accessed through the RDF query languages that support subgraph extraction?", the *BotPerception* receives the query and encodes it to ASCII [1]. After that, *BotPlanning* divides this question into four smaller questions as follows:

1) Which graph databases support python?

2) Which graph databases support Linux?

3) Which graph databases can be accessed through RDF query languages?

4) Which RDF query languages support subgraph extraction?

Next, *BotPlanning* determines should solve the 4th question first. Finally, *BotReasoning* reasons the answers of 4th question and uses the answers of the 4th question to instead the "RDF query languages" of 3rd question. Then, *BotReasoning* deduces the answers satisfied the 1st, 2nd and 3rd questions simultaneously, and generate the explanations for all the question-solving processes.

Based on the above brain-inspired computation, *XBot* can be used as the basis for designing a human-like knowledge graph-based Q&A system that has the ability to answer and explain questions by learned knowledge.

Based on the above brain-inspired computation, *XBot* can be used as the basis for designing a human-like knowledge graph based Q&A system that has the ability to answer and explain questions by learned knowledge.

[1] https://en.wikipedia.org/wiki/ASCII

3.3 Summary

This chapter conducted a study on the content of cognitive science related to human Q&A cognitive process. And presented a complete and detailed cognitive process of question reasoning of the human brain including perception, planning, reasoning, response and the interactions to semantic memory. Depended on the above investigation, this chapter proposed a brain-inspired cognitive framework of Q&A process named *XBot*. *XBot* consists of five main modules: *BotLearning*, *BotPerception*, *BotPlanning*, *BotReasoning* and *BotResponse* which is largely inspired by the literature in cognitive science and aligned with the cognitive process of human. It can be used as a basis for designing a knowledge graph based Q&A system that can understand, answer questions and provide a human-centred explanation. Compared with the existing Q&A systems based on knowledge graph, *XBot* has the capacity to answer questions with explainability by learned knowledge. *XBot* also can be customized in different domains which just like humans can learn different knowledge and become experts of different domains.

Chapter 4

DeveloperBot: An Explainable Search Engine Assistant based on Domain-Specific Knowledge Graph

In Chapter 3, the *XBot* framework is presented. This chapter will present a tool called *DeveloperBot* including a series of implementations of the modules of *XBot* framework.

Search engines play a critical role in developers' online information needs and problem-solving. Although search engines (such as Google) have the capacity to obtain a hyperlink list according to the query keywords quickly, we noticed that when the query is complex, developers need to repeatedly search and open a large number of web pages to filter and synthesize answers. As far as we know, existing Q&A systems for developers are few and lack explainability, which makes the answers hard for developers to trust and adopt.

In this chapter, *DeveloperBot* will be used as a search engine assistant to assist the solving of the closed-end questions. *DeveloperBot* is customized by loading the knowledge graph of the software engineering domain into the knowledge base of the *XBot* framework. *DeveloperBot* consists of four main modules: 1) *BotPerception* is a query graph construction algorithm which splits a multi-condition query into several simple constraints and constructs the constraints into a multi-layer query graph, 2) *BotPlanning* can divide a query into several simple constraints and decide the right solving order of the constraints by dependency parsing, (3) *BotReasoning* is a fast graph cyclic pruning reasoning algorithm inspired from spreading activation model of cognitive science [44], it models the constraint solving process as subgraph search and decision-making process and generates the explanations, (4) *BotResponse* presents the answers and explanations by UI.

The results of the experiments show that compared with just using Google, with the assist of *DeveloperBot*, users can find answers faster and with more accuracy. The explanation can not only significantly improve the developers' trust and adoption to the answers, but also assist the developers to understand the answers more deeply, improve the answer accuracy and form better search keywords. Furthermore, the more complex the question is, the more effective the *DeveloperBot* can achieve.

The rest of this chapter is organized as follows: In the Section 4.1, we introduce the background and motivation. Section 4.2 is a scenario to show that how *DeveloperBot* works and why *DeveloperBot* is needy. In the Section 4.3, we first introduce the architecture of *DeveloperBot* in Section 4.3.1. Next, we elaborate the details of the approach, e.g., knowledge graph construction in the Section 4.3.3, *BotPerception* and *BotPlanning* modules in the Section 4.3.4, *BotReasoning* module in the Section 4.3.5 and the UI of the system in the Section 4.3.6. The experiment and evaluation of *DeveloperBot* are shown in the Section 4.4.

4.1 Background and Motivation

Many studies and our preliminary survey show that web search engines are the most dominant, prominent and indispensable tool for developers to search online information resources or solve problems currently [116, 117, 118, 119]. Today's web search engines primarily follow the "query-response" or short lookup concept. The developers use these web search engines for a variety of search tasks by typing queries. Then they receive ranked lists of search results (or say hyperlinks) [116].

Recently, the community of search engine users is increasingly recognizing that traditional retrieval models are insufficient to satisfy complex information needs [120, 121, 116]. For example, if a developer types the following query in a search engine - "data visualization library for javascript". The received list of search results is reasonable since developers only need to click on a few hyperlinks to get the information they want. But try this query - "Which graph databases are compatible with Linux and support both Java and Python?". The list of returned results is far from satisfactory. In order to get the information they want, the developer has to open a lot of hyperlinks and synthesize the results of different sources related to their information needs [116]. The above two queries are very easy to understand for humans, but current NLP-based web search engines still can not perform well on complex queries [122].

Based on the above situation, in order to provide web search engine users with better support on complex search tasks, many research works propose to provide additional enhanced services for complex search tasks depending on their characters [116]. This is a consensus for major search engine companies (e.g. Google, Bing, Yahoo, etc.): automatic Question Answering (Q&A) system (also known as a direct answer search engine, or natural language search engine, etc.) is a more advanced next-generation search engine which returns real-time answers instead of a ranked list of hyperlinks [123, 124, 125, 126]. But it is one of the more difficult tasks in the field of natural language processing (NLP) because it requires a wide range of intelligence (e.g., parsing, search, information extraction, inference, etc.).

Furthermore, according to our survey of the state of the art Q&A system, without the original semantic context, the lack of explainability that makes the answers difficult for users to trust and adopt [75, 76, 127]. In order to provide a human-centred explanation, the Q&A system should align with the cognitive model of human and explain within the basic framework of human cognition [27, 28, 17, 18, 19, 29].

In this chapter, a tool called *DeveloperBot* including a series of implementations of the modules of *XBot* framework will be presented. *DeveloperBot* can be used as a search engine assistant to assist in solving the complex closed-end questions. *DeveloperBot* is customized by loading the knowledge graph of the software engineering domain into the knowledge base of the *XBot* framework. *DeveloperBot* consists of four main modules, (1) *BotPerception* is a query graph construction algorithm which incorporates the Dependency Parsing into the Tree Parsing [46] for maximizing the completeness of extracted answers constraints to a query graph, (2) *BotPlanning* can split a query into several simple constraints and decide the right solving order of the constraints by dependency parsing, (3) *BotReasoning* is a fast graph cyclic pruning reasoning algorithm combined with logic rules and data, which will use the query graph to search a candidate subgraph from the knowledge graph by spread activation algorithm firstly. Secondly, this algorithm extracts the features according to the topological structure of subgraph and word embedding of the predicate of candidate answers triples. The last, it makes a decision for the final answers based on Bayesian Decision Theory or Deep Neural Networks (DNN) and extract the reasoning subgraph and compute the confidence as explanations, (4) *BotResponse* presents the answers by UI (user interface).

The result of decision making shows that, with topological structure, predicate similarity and other novel features of the subgraph, the Bayesian decision theory and DNN algorithm

can extract the correct answers from the candidate answers with about 98% and 99% accuracy respectively, and the mean square error (MSE) of confidences are 0.0083 and 0.00022, respectively. We implement a prototype of *DeveloperBot* and use it as a search engine assistant specific for developers for evaluation. Then a user study involving 24 developers was conducted with five searching tasks. The participants of the user study express that *DeveloperBot* has the capacity to model and reason complex problems. They think that *DeveloperBot* is a good supplement to the search engine. The experimental data also shows similar results: compared with just using Google, with the assist of *DeveloperBot*, users can find answers faster and with more accuracy. In addition, using the reasoning subgraph and answer confidence as the explanation of the direct answers can significantly improve the developers' trust and adoption to the answers. These explanations also assist the developers to understand the answers more deeply, improve the answer accuracy and form better search keywords. Furthermore, the more complex the question is, the more effective the *DeveloperBot* can achieve.

This chapter makes the following three major contributions:

1) A tool called *DeveloperBot* including a series of implementations of the modules of *XBot* framework is proposed, which is customized by loading the knowledge graph of software engineering domain into the knowledge base of *XBot* framework. *DeveloperBot* can be used as a search engine assistant to assist the solving of the complex closed-end questions with human-centred explainability.

2) A novel query graph construction algorithm is proposed to split a multi-condition query into several simple constraints and construct the constraints into a multi-layer query graph (*BotPerception*), and meanwhile, determine the solving order of these constraints (*BotPlanning*). In contrast, this algorithm analyzes the expression pattern of natural language deeply and enhances the representation capacity of the complex multi-condition query.

3) Further, a fast graph cyclic pruning reasoning algorithm named *BotReasoning* is proposed, which inspires from spreading activation model and models the question answering as subgraph search and decision making process by Bayesian decision theory or DNN. *BotReasoning* also extracts the reasoning subgraph and computes the confidence values as qualitative and quantitative explanations to the corresponding direct answers, respectively.

Based on the novel features, *BotReasoning* can extract answers with higher accuracy and estimate answer confidences with lower MSE.

4) The prototype of the *DeveloperBot* system is implemented and customized by a knowledge graph of software engineering domain as a proof-of-concept. A user study is also conducted to evaluate its practical values.

4.2 Motivating Scenario

In this section, we select a scenario that the developers (say Tan) may encounter when they searching online information resources, and show how our search engine assistant *DeveloperBot* can assist for complex query understanding and multi-conditional querying.

Assume that Tan is assigned a development task to extend a Python-based project on a Linux operating system as shown in Fig. 4.1. This project involves a large number of queries to a heterogeneous knowledge graph including RDF graphs and property graphs[1]. Tan wants to choose the most suitable graph database for this project according to the existing environment. Considering upgrades and maintenance in the future, this database should satisfy the following constraints natively:

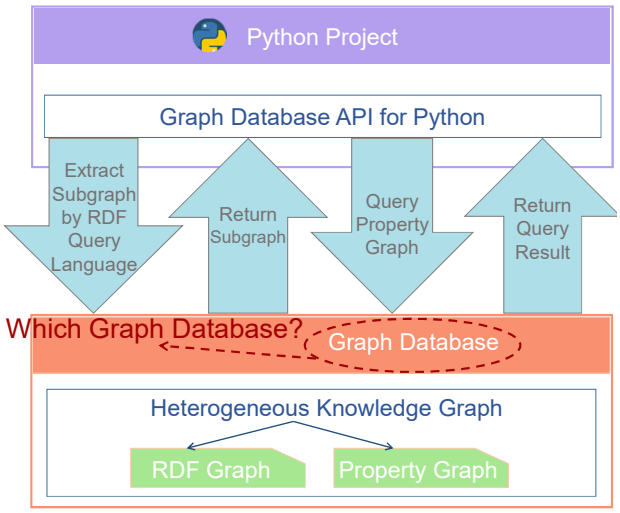


Figure 4.1: The System Framework of the Motivating Scenario

[1]https://neo4j.com/blog/rdf-triple-store-vs-labeled-property-graph-difference/

- This database should officially support the standard Python API.
- Since this database needs to store two different graph data, RDF graphs and property graphs, graph database which can be accessed by RDF query language will be the best choice (instead of RDF database that only supports RDF graph).
- This database needs to perform a critical operation: a large number of specific subgraph retrievals, so the RDF query language supported by this database is required to support subgraph extraction at the same time.

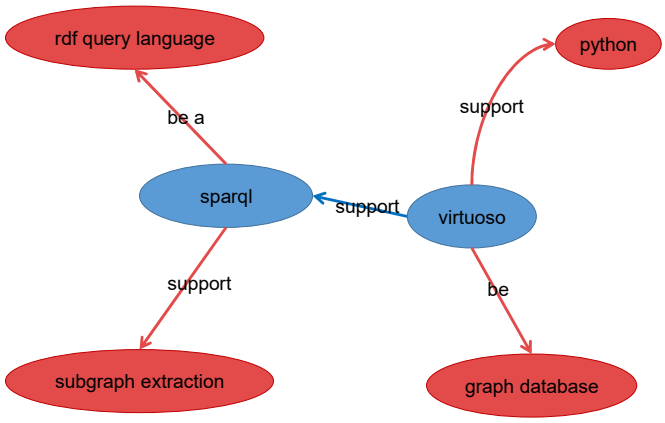


Figure 4.2: The Reasoning Subgraph Explanation of Direct Answer "Virtuoso"

According to the above requirements, Tan forms the query "Which graph databases support Python and can be accessed through the RDF query languages that support subgraph extraction?" and then input the query to the search box of *DeveloperBot*.

For this query, *DeveloperBot* gives the following response: First, the response includes three direct answers: Amazon Neptune, Virtuoso and DB2, which are recommended graph databases that meet the requirements of Tan. Second, each direct answer contains a reasoning subgraph to explain the reasons for the recommendation. For example, the reasoning subgraph of the direct answer "Virtuoso" is shown in Fig. 4.2. This graph explains that the direct answer "Virtuoso" is recommended because the "Virtuoso" is a graph database, and it supports Python and SPARQL. And the SPARQL is an RDF query language that supports subgraph extraction. Third, *DeveloperBot* also gives its confidence for every direct answer. The above operations allow Tan to locate the three answers that meet his requirements quickly.

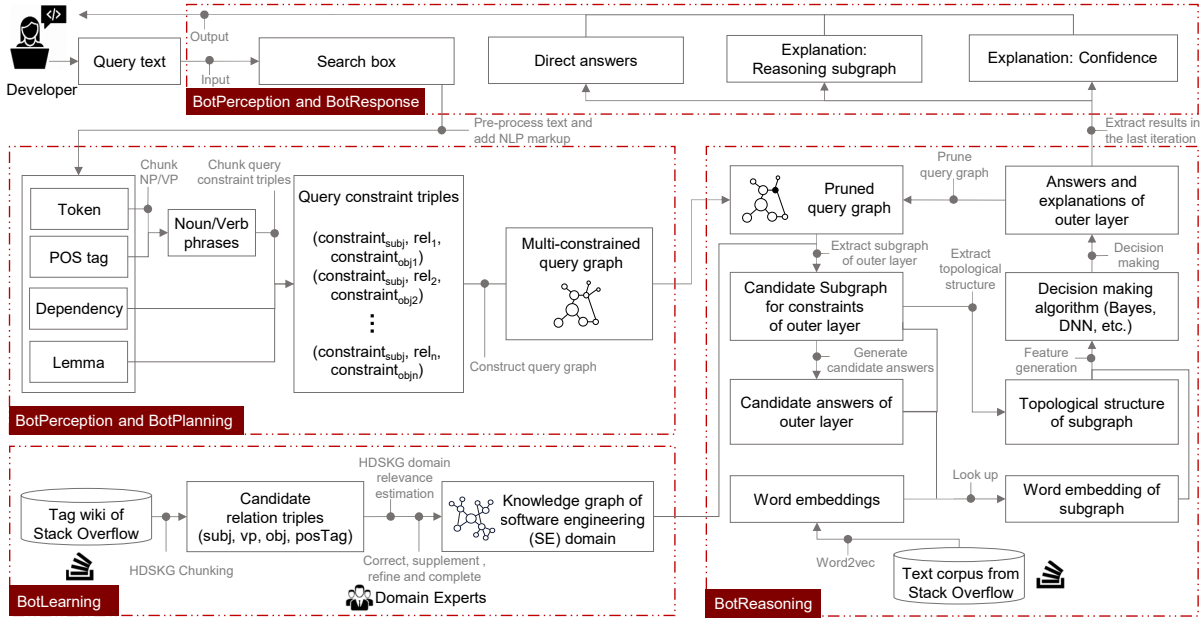


Figure 4.3: The Framework of *DeveloperBot*

4.3 Approach

4.3.1 Framework of *DeveloperBot*

Fig. 4.3 is the framework of *DeveloperBot*. The input of *DeveloperBot* is the query text from developer (e.g., "Which mobile operating system is based on Linux kernel?".). The *DeveloperBot* has three outputs as shown below:

- *Direct answers:* the simple and accurate answers provided by our system (rather than a list of web sites).
- *Explanation - Reasoning subgraph*: a reasoning subgraph extracted from the knowledge graph which indicates "why" and "how" this answer is be presented.
- *Explanation - Confidence*: the confidence of *DeveloperBot* for every direct answer which indicates "how confident" the system is in this answer.

DeveloperBot contains four main parts as shown in the blocks of red dot line Fig. 4.3. These four blocks correspond to different implementations of the framework of cognitive model *XBot*.

BotPerception and *BotResponse* is implemented by *User Interface*, which is a frontend web-based application that consists of a search box for the user to input their query text and an area to visualize the direct answers and corresponding explanations. The *BotPerception* and *BotPlanning* are implemented by a *Multi-constrained Query Graph Construction* algorithm. These parts receive the query text from the *User Interface*, pre-processes the query text, add NLP (Natural Language Processing) markup like token, POS tag, dependency, etc., chunk NP and VP, chunk query constraint triples, determine the solving order and construct a multi-constrained query graph. The knowledge graph of the software engineering domain is constructed offline at the *Domain Knowledge Graph Construction* part which implements *BotLearning*. *BotLearning* extract knowledge graph from tag wiki of Stack Overflow by combining an algorithm called *HDSKG* and the knowledge of domain experts in software engineering.

For the *BotReasoning* part, a *Fast Graph Cyclic Pruning Reasoning Algorithm* is proposed as the implementation. Firstly, *BotReasoning* extracts a candidate subgraph for the outer layer of the query graph from the domain-specific knowledge graph. Secondly, it generates the candidate answers and extracts the novel features of them. Thirdly, the vector representation of subgraph and constraint triples of candidate answers will be obtained by looking up the result of word2vec. Here a word2vec model is pre-trained offline by a text corpus from Stack Overflow. Fourthly, a decision making algorithm (e.g., Bayesian decision theory, DNN) will choose right answers from the candidate answers, and then use them to prune the outer layer of query graph. *BotReasoning* repeats the above iteration operation until the last constraint triples is solved, then extracts the direct answers and their explanations to the UI.

4.3.2 Algorithm of *DeveloperBot*

The pseudo code of *DeveloperBot* is shown in Algorithm. 1. The *BotPerception* and the *BotPlanning* subroutines receive Q, pre-processes the query text, add NLP (Natural Language Processing) markup like token, POS tag, dependency, etc., chunk NP and VP, chunk query constraint triples, determine the solving order and construct QG.

For *BotReasoning* the subroutines, firstly, *BotReasoning* extracts a SG for the outer layer of QG from KG. Secondly, it generates the candidate answers and extracts the novel features of them. Thirdly, a decision making algorithm (e.g., Bayesian decision theory, DNN) will choose AS from candidate answers, and then iterates AS to the object of previous layer of QG and

Algorithm 1 DeveloperBot

Input: Q, KG
Output: Direct Answers, Explanations

Subroutine *BotPerception*(Q)
Add NLP markup of Q
Tree Parsing of Q
Check Patern1-Patern7
return Constraint triples of Q
EndSubroutine
Subroutine *BotPlanning*(Constraint triples of Q)
Determine the solving order of the Constraint triples of Q
$QG \leftarrow$ Constraint quads
return QG
EndSubroutine
Subroutine *BotReasoning*(QG, KG)
$PQG \leftarrow QG$
while ($PQG \neq \varnothing$) **do**
    $currentCQ \leftarrow$ outer layer of PQG
    CA, $SG \leftarrow$ subgraphSearch($currentCQ$, KG)
    $AS \leftarrow$ decisionMaking(featureExtraction(CA, SG))
    Iterate AS to pervious layer of PQG, prune $currentCQ$ from PQG
    Generate and combine explanations
end while
return Direct Answers and Explanations
EndSubroutine
$QG \leftarrow$ *BotPlanning*(*BotPerception*(Q))
Direct Answers and Explanations $\leftarrow$ *BotReasoning*(QG, KG)
return Direct Answers and Explanations

prune $currentCQ$ from PQG. *BotReasoning* repeats the above iteration operation until the last constraint triples of PQG is solved, then returns the direct answers and their explanations.

4.3.3 Knowledge Graph of Software Engineering Domain Construction

The backend knowledge base of the *DeveloperBot* is a knowledge graph of software engineering mainly extracted by the advance domain knowledge graph extraction techniques, e.g., *HDSKG*[128, 66], *HDSO* [129]. In addition, this knowledge graph is also augmented by the knowledge of domain experts in software engineering, synonym set and NLP techniques like NER (Named-entity recognition) [130]. The technique *HDSKG* will be elaborated in the next

chapter, this section will present a summarization of the knowledge augmentation. Refer to our paper for more technical details [128, 131, 129].

4.3.3.1 Augment Knowledge Graph by Knowledge of Domain Experts

The knowledge graph extracted from the crowdsourcing data has its limitations. For example, some relations are very important in the software engineering domain, but they are rarely mentioned in natural language crowdsourcing data because these relations are too common sense (e.g. "A graph database is a database."). In addition, there are some equivalence relations in the software engineering domain. For example, if there is a relation (P4V; is; cross_platform)[1]. A cross-platform application may run on Microsoft Windows, Linux, and macOS[2]. So we can derive relations that (P4V; run_on; Microsoft_Windows), (P4V; run_on; Linux) and (P4V; run_on; macOS)[3]. So the common sense knowledge of domain experts is used to augment the knowledge graph.

4.3.3.2 Augment Knowledge Graph by Synonym Set

Crowdsourcing data is edited by a different user, so there are often different expressions for the relation. For example, a relation (doxygen, is_available_for, Linux) is the same mean as (doxygen, run_on, Linux). There are also some different expressions point to the same concept like "object-database" and "object-oriented-database" are the same meaning. Stack Overflow provides a synonym set[4] with more that 4000 synonyms in the software engineering domain. These synonyms are incorporated into the domain knowledge graph with relation "is_synonym_of" (e.g. mpmjs; is_synonym_of; npm) for further analysis.

4.3.3.3 Augment Knowledge Graph by NLP Techniques

In addition, there are also some implicit relations between some entities, which can be identified by syntax. The "head"[5] of a phrase is the word that determines the syntactic category of that phrase. For example, "red dog" is a "dog", the word "red" of the phrase modify the

[1] https://stackoverflow.com/tags/p4v/info
[2] https://en.wikipedia.org/wiki/Cross-platform_software
[3] https://www.perforce.com/downloads/helix-visual-client-p4v
[4] https://stackoverflow.com/tags/synonyms
[5] https://en.wikipedia.org/wiki/Head_(linguistics)

head, and is, therefore, the head's dependents. Here we extend the syntax rule "head" to the entity level. Assume there are two entities in the knowledge graph named $entity_1$ and $entity_2$, if $entity_1$ is the head of $entity_2$, a hierarchical relation ($entity_2$; is_a; $entity_1$) will be built in the knowledge graph. For example, if $entity_1$ is "programming_language" and $entity_2$ is "object-oriented_programming_language". The entity "programming_language" is the head of "object-oriented_programming_language" since the "object-oriented" modify this head entity "programming_language", so "object-oriented" is dependent of "programming_language". So we can derive a relation that (object-oriented_programming_language; is_a; programming_language)

Named-entity recognition (NER) (also known as entity identification, entity chunking and entity extraction) seeks to locate and classify sequences of words in a text into pre-defined categories such as person and company names, or gene and protein names, organizations, locations, medical codes, time expressions, etc. The *DeveloperBot* incorporates the tool named *coreNLP* released by Stanford to recognize all the entities in the final knowledge. graph [132, 133, 134, 135].

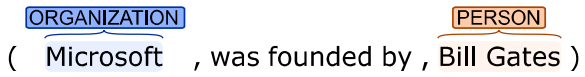


Figure 4.4: Result of Named-entity Recognition for Relation Triples (Microsoft, was_founded_by, Bill_Gates)

As shown in Fig. 4.4, the relation triples (Microsoft, was_founded_by, Bill_Gates) is recognized two named-entities, they are "Microsoft"-ORGANIZATION and "Bill Gates"-PERSON. So two relation triples (Microsoft, is_a, organization) and (Bill_Gates, is_a, person) will be added into the knowledge graph.

4.3.4 *BotPerception* and *BotPlanning*: Multi-constrained Query Graph Construction

In order to understand the syntax of the technical questions of developers, rough observation and analysis are made on the closed-ended questions of Stack Overflow. The results show that these questions are often complex and diverse, and the constraints of the query are often scattered in various grammars, such as attributive clauses, coordinate clause, etc. Therefore, some traditional query understanding methods like greedy technical terms matching [136] or tree parsing [12] are not suitable for multi-constraints complex query understanding.

In this part, we will elaborate on how *BotPerception* combines dependency into constituency-based parse trees[1], and extracts the constraints in the query completely. Meanwhile, the *BotPlanning* will divide a query into smaller queries and decide the right solving order. Then the constraints and query solving order will be constructed into the query graph as constraint quads.

4.3.4.1 What is Query Graph of *DeveloperBot*?

Here we use the original sample query *"Which graph databases support Python and Linux, and can be accessed through the RDF query languages that support subgraph extraction?"* to analyze which features need to be represented by a query graph. In this sample query, the direct answers should be one or more graph databases that meet the three constraints: 1) support Python, 2) support Linux, 3) can be accessed through the RDF query languages. And the RDF query languages should meet one constraint: support subgraph extraction. Here the "graphics databases" is a category constraint (or called taxonomic constraint) that indicates which sub-type the direct answer belongs to in the hierarchical types. The "support Python", "support Linux" and "can be accessed through the RDF query languages" are property constraints (or called non-taxonomic constraints). Property constraint represents the feature of the direct answers by a phrase beginning with a VP followed by an NP. For a query graph, the constraints used to describe the direct answer are called the first layer constraints.

Similarly, the "RDF query languages" is a category constraint and "support subgraph extraction" is a property constraint. And these two constraints are called second layer constraints. If there are some constraints which used to describe the concept in the second layer, it is called third layer constraints and so on.

4.3.4.2 Pre-process Text

Given a search query, relying on synonym set created by the community of Stack Overflow, *DeveloperBot* first implements a matching method to identify and replace the synonyms in a search query, such as replace jdk11 with Java-11, DBMS (database management system) for database, Microsoft Windows to windows. Then, *DeveloperBot* splits the query sentence into words by spaces. Next, it simplifies the plural words into singular words through stemming analysis.

[1]https://en.wikipedia.org/wiki/Parse_tree

4.3.4.3 Annotate Query with Multi-Dimensional NLP Markup

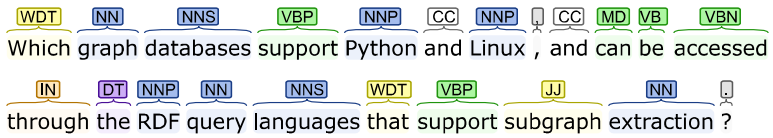


Figure 4.5: POS tagging results of the sample query

Natural language markup is widely used in natural language preprocessing, sentence structure analysis and semantic analysis, etc [137, 138, 139]. The *DeveloperBot* incorporates the tool named *coreNLP* released by Stanford to its NLP markup annotation component [132, 133, 134, 135]. Then, we apply the tokenization, POS (Part of Speech) tagging, dependency parsing and lemmatization to the query. The task of tokenization is to break the sentence into small pieces, called tokens, and drop some characters like punctuation. POS tagging is the process of labelling words with their appropriate Part-Of-Speech (or, more generally, grammatical properties), such as NN: Noun, singular or mass, WP: Whpronoun, VBZ: Verb, 3rd person singular present, etc. Fig. 4.5 shows the POS tagging result of a complex query with multiple constraints.

Dependency is the grammatical structure of a sentence, mainly the relationship between the control word (also called "head") and the subordinate word (the word that modifies the head).

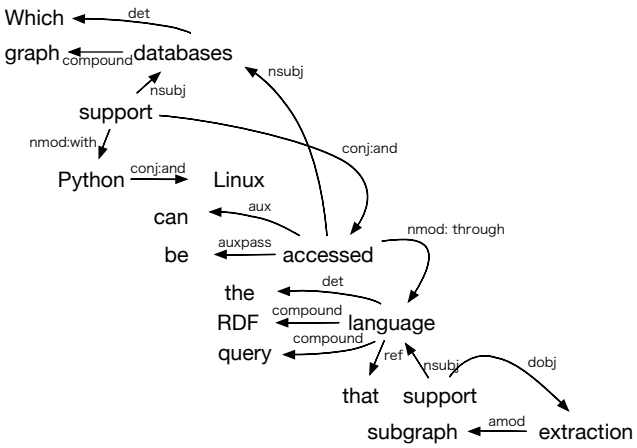


Figure 4.6: Graphical representation of the Dependencies for the query: *"Which graph databases support Python and Linux, and can be accessed through the RDF query languages that support subgraph extraction?"*

Fig. 4.6 shows a directed graph representation of the whole dependencies for the sample query. The nodes of Fig. 4.6 are the words of the query, and the edges in Fig. 4.6 are grammatical relations of the words.

Tab. 4.1 presents some detailed description of some key dependencies relations of query *"Which graph databases support Python and Linux, and can be accessed through the RDF query languages that support subgraph extraction?"*.

Table 4.1: Dependencies Description for Query: *"Which graph databases support Python and Linux, and can be accessed through the RDF query languages that support subgraph extraction?"*

Dependency	Governor	Dependent	Semantic relationship between the words depicted by denpendency
det	databases-3	Which-1	"Which-1" is determiner of "databases-3"
nsubj	support-4	databases-3	"databases-3" is nominal subject of "support-3"
nsubjpass	accessed-12	databases-3	"databases-3" is passive nominal subject of "accessed-12"
dobj	support-4	Python-5 Linux-7	"Python-5" and "Linux-7" are direct objects of "support-4"
nmod:through	accessed-12	languages-17	"languages-17" is nominal modifier of "accessed-12"
acl:relcl	languages-17	support-19	"support-19" is a relative clause modifying the "languages-17"
conj:and	support-4	accessed-12	"support-4" and "accessed-12" are connected by coordinating conjunction "and"

As Tab. 4.1 shows, "det" is the abbreviation of determiner. There are three interrogative determiners: what, which, whose What is for asking for information specifying something. Which are interrogative determiners for asking for information specifying one or more people or things from a definite set. Whose means "belonging to which person". In our illustrative query, "Which-1" is the interrogative determiners of the "databases-3", it determines that the answer to this question is to specify one or several databases that meet the constraints from the set of all database.

"nsubj" is the abbreviation of a nominal subject. It is a noun phrase which is the passive syntactic subject of a clause [140]. For our illustrative query, nsubj (support-4, databases-3) and

nsubj (supports-19, languages-17) means that "databases-3" and "languages-17" is the syntactic subject of the verb "support-4" and "supports-19", respectively.

"nsubjpass" is the abbreviation of the passive nominal subject. In the sample query, "databases-3" is passive nominal subject of "accessed-12".

The "dobj" is the direct object of a VP, and is also an NP which is the accusative object of the verb. Here "Python-5" and "Linux-7" is direct object of "supports-4".

The "nmod" means and uses between two content words, as the preposition of one content word is now viewed as a case depending on its complement. In general, the "nmod" indicates some further adjunct relation specified by the case [141]. Here the nmod (accessed-12, languages-17) means that "languages-17" is nominal modifiers of "accessed-12". Notice that some times there will appear a preposition after "nmod" like "nmod: through". This "through" means the complement use the preposition "through" to modify the Dependent.

"acl:relcl" means a relative clause modifier of a noun is a relative clause modifying the noun. Here the "support-19" is a relative clause modifying the "languages-17"

"conj" is conjunct and indicates the relation between two elements connected by a coordinating conjunction, such as "and", "or", etc. In the illustrative query, the "support-4" of the relation is the first conjunct and "accessed-12" depend on it via the coordinating conjunction "and".

The goal of lemmatization is to reduce inflectional forms and derivational related forms of a word to a common base form.

4.3.4.4 Constituency-based Tree Parsing

In this step, a fulltext parsing technique called "Tree Parsing" is adapted to segment a sentence into its subconstituents [142, 143, 144, 12]. As shown in Tab. 4.2, *DeveloperBot* sets four subconstituents called WHNP, WHVP, NP and VP. Here NP and VP are abbreviations of noun phrase and verb phrase, respectively. Subconstituent WHNP represents a phrase beginning with "which", "what" or "whose" following by a noun phrase (e.g. Which relational databases). If a phrase starts with "who", "when" or "what" and following by a verb phrase, WHVP will be the name of this subconstituent (e.g. Who developed).

The second column of Tab. 4.2 is the parsing expressions of the above four subconstituents. These parsing expressions are represented by POS tags derived from Fig. 4.5. Here (WDT) represents "which" and "what"; (WP$) is wildcard of "whose"; (WP) stands for "who" or

Table 4.2: Parsing Expression of Subconstituent

Name	Parsing Expression
INNP	(IN)+(CD)*(DT)?(CD)*(JJ)*(CD)*(VBD|VBG)*- (NN.*)*(POS)*(CD)*(VBD|VBG)*(NN.*)*- (VBD|VBG)*(NN.*)*(POS)*(CD)*(NN.*)+
WHNP	(WDT|WP$)+(CD)*(DT)?(CD)*(JJ)*(CD)*(VBD|VBG)*- (NN.*)*(POS)*(CD)*(VBD|VBG)*(NN.*)*- (VBD|VBG)*(NN.*)*(POS)*(CD)*(NN.*)+
WHVP	(WP|WRB$)+(MD)*(VB.*)+(JJ)*(RB)*(JJ)*(VB.*)?- (DT)?(IN*\|TO*)+ (WP\|WRB$)+(MD)*(VB.*)+
NP	(CD)*(DT)?(CD)*(JJ)*(CD)*(VBD|VBG)*(NN.*)*- (POS)*(CD)*(VBD|VBG)*(NN.*)*- (VBD|VBG)*(NN.*)*(POS)*(CD)*(NN.*)+
VP	(MD)*(VB.*)+(CD)*(JJ)*(RB)*(JJ)*(VB.*)?(DT)?(IN*|TO*)+ (MD)*(VB.*)+(JJ)*(RB)*(JJ)*(VB.*)?(DT)?(TO*)+(VB)+ (MD)*(VB.*)+(JJ)*(RB)*(JJ)*(VB.*)?(DT)?(IN*)+(VBG)+ (MD)*(VB.*)+(JJ)*(RB)*(JJ)*(VB.*)?(DT)?(IN*|TO*)+ (MD)*(VB.*)+(JJ)*(RB)*(JJ)*(VB.*)+ (MD)*(VB.*)+

"what" (when "what" followed by noun phrase); (WRB) is wildcard of "when". (MD) is modal; (VB.) stands for different categories of verb such as VB - verb base form, VBG - verb gerund or present participle, VBN - verb past participle, VBP - verb non-3rd person singular present, VBZ - verb 3rd person singular present. (NN.*) stands for different categories of noun such as NN - singular or mass noun, NNS - plural noun, NNP - singular proper noun, NNPS - plural proper noun. (JJ) represents an adjective; (RB) is an adverb; (DT) presents an article; and (IN*) means any preposition or subordinating conjunction.

In the parsing expressions, "?" stands for whether or not there is such a determinant; "*" means zero or more determinant; "+" means must have such a determinant; "-" means continue to next row.

Using the tree parsing, the tokens of the sentence are segmented to WHNP, WHVP, VP and NP. For example, the sentence *"Which graph databases support Python and Linux, and can be accessed through the RDF query languages that support subgraph extraction?"* is segmented

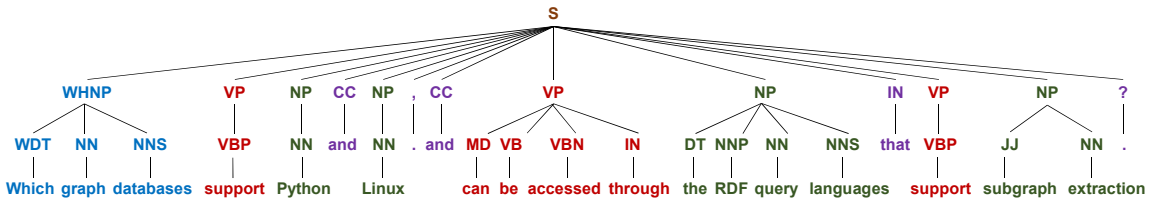


Figure 4.7: Result of Tree Parsing

into WHNP: *"Which graph databases"*; VP: *"support"* and *"can be accessed through"*; NP: *"Python"*, *"Linux"*, *"the RDF query languages"* and *"subgraph extraction"* as shown in Fig. 4.7.

4.3.4.5 Construct Query Graph with Dependency Parsing

Assume there is a query Q, the query graph constructed by *BotPerception* and *BotPlanning* is represented as QG. A QG is composed of the basic units called constraint quads, and a constraint quads consists of four basic elements which is represented as (category constraints, VP of property constraints, NP of property constraints, layer number).

Firstly we use the constituency-based tree parsing to parse Q and get all the WHNP, WHVP, NP and VP as shown below:

$$\begin{aligned} Q = (&WHNP_1, WHNP_2, ..., WHNP_i, ..., WHNP_m; \\ &WHVP_1, WHVP_2, ..., WHVP_j, ..., WHVP_n; \\ &NP_1, NP_2, ..., NP_k, ..., NP_p; \\ &VP_1, VP_2, ..., VP_s, ..., VP_q)\,, \end{aligned} \tag{4.1}$$

where $WHNP_i$, $WHVP_j$, NP_k and VP_s are the ith WHNP, jth WHVP, kth NP and sth VP respectively in Q. There are m WHNP, n WHVP, p NP and q VP in Q.

$$\begin{aligned} WHNP_i &= (whnp_{i1}, whnp_{i2}, ..., whnp_{if}, ..., whnp_{ix}) \\ WHVP_j &= (whvp_{j1}, whvp_{j2}, ..., whvp_{jw}, ..., whvp_{jy}) \\ NP_k &= (np_{k1}, np_{k2}, ..., np_{kr}, ..., np_{ku}) \\ VP_s &= (vp_{s1}, vp_{s2}, ..., vp_{st}, ..., vp_{sv}), \end{aligned} \tag{4.2}$$

where $whnp_{if}$, $whvp_{jt}$, np_{kr} and vp_{st} are fth, wth, rth and tth word of $WHNP_i$, $WHVP_j$, NP_k and VP_s, respectively. Assume that the total number of words of $WHNP_i$, $WHVP_j$,

NP_k and VP_s are x, y, u, and v. Note that the subscripts of the subconstituents and the words of subconstituents are arranged in the order in which they are in the query sentence.

Secondly, the dependency parsing technique is used for markup the dependency for all words of Q. Because of the variability of natural language, it has a lot of uncertainties (e.g. There are many expressions for the same meanings, or the meaning of a word in different contexts is completely different, etc.) After in-depth observation and analysis to the questions on Stack Overflow and our data set, we summarize the query sentence structures commonly used by developers and propose the following 7 query constraints and their layers extraction patterns to construct a query graph [1].

Pattern 1: If in a query Q, the number of WHNP $m = 1$, the number of WHVP $n = 0$, and WHNP is detected at the beginning of the Q. The NP following the interrogative word is represented as $WHNP_{NP} = (whnp_{np1}, whnp_{np1}, ..., whnp_{npe}, ..., whnp_{npw})$. $whnp_{npe}$ is the eth word of $WHNP_{NP}$, and there are w words in $WHNP_{NP}$ in total. The $WHNP_{NP}$ will be regarded as a category constraint of direct answers.

After determining category constraints, we will extract the property constraints corresponding to the category constraints. If the following 4 dependency pair are detected:

- $dp[nsubj\ (vp_{st}, whnp_{npe}), dobj\ (vp_{st}, np_{kr})]$
- $dp[nsubjpass\ (vp_{st}, whnp_{npe}), dobj\ (vp_{st}, np_{kr})]$
- $dp[nsubj\ (vp_{st}, whnp_{npe}), nmod\ (vp_{st}, np_{kr})]$
- $dp[nsubjpass\ (vp_{st}, whnp_{npe}), nmod\ (vp_{st}, np_{kr})]$

Then a property constraints (VP_s, NP_k) will be extracted into the QG. Where $dp[a, b]$ means that a and b appear simultaneously in the result of dependency parsing of Q.

For example, in the query *"Which graph databases support Python and Linux, and can be accessed through the RDF query languages that support subgraph extraction?"*, *"Which graph databases"* is a WHNP at the beginning of the query phrase and *"graph databases"* is the $WHNP_{NP}$ as shown in Fig. 4.7. So (graph_databases) will be extracted as a category constraint of direct answers.

[1] Here the 7 patterns presented is typical patterns covering most of the scenarios. There are still a small number of missing scenarios.

Furthermore, there are three dependency pairs (1) $dp[nsubj\ (support-4, databases-3)$ and $dobj(support-4, Python-5)]$, (2) $dp[nsubj\ (support-4, databases-3)$ and $dobj\ (support-4, Linux-7)]$ and (3) $dp[nsubjpass\ (accessed-12, databases-3)$ and $nmod\ (accessed-12, languages-17)]$ are detected as shown in Fig. 4.6. Here "support-4" is vp_{11} which means the first word in the first VP of Q. The "databases-3" is $whnp_{np2}$ representing the second word of NP of WHNP and so on.

In *BotPerception*, all property constraints belonging to category constraints of direct answers are classified as the first layer of Q, which means that the order of solving these constraint quads is last. So from the sample query, **Pattern 1** can extract three constraint quads:

- (graph_databases, support, Python, WHNP)
- (graph_databases, support, Linux, WHNP)
- (graph_databases, can_be_accessed_through, RDF_query_languages, WHNP)

Pattern 2: If a constraint quads with layer number = c has been extracted. Assume the "NP of property constraints" of this constraint quads is:

$$NPSC = (npsc_1, npsc_2, ..., npsc_g, ..., npsc_d), \tag{4.3}$$

where $npsc_g$ is the gth word of $NPSC$, and there are d words in $NPSC$ in total.

If the following 4 dependency pair are detected:

- $dp[nsubj\ (vp_{st}, npsc_g), dobj\ (vp_{st}, np_{kr})]$
- $dp[nsubjpass\ (vp_{st}, npsc_g), dobj\ (vp_{st}, np_{kr})]$
- $dp[nsubj\ (vp_{st}, npsc_g), nmod\ (vp_{st}, np_{kr})]$
- $dp[nsubjpass\ (vp_{st}, npsc_g), nmod\ (vp_{st}, np_{kr})]$

Then a constraint quads (NPSC, VP_s, NP_k, $c+1$) will be extracted into the QG.

For example, in the query *"Which graph databases support Python and Linux, and can be accessed through the RDF query languages that support subgraph extraction?"*, a constraint quads (graph_databases, can_be_accessed_through, RDF_query_languages, 1) has been extracted, so NPSC = "RDF query languages". A dependency pair $dp[nsubjpass\ (accessed-$

$12, databases - 3), nmod\ (accessed - 12, languages - 17)]$ is detected. Here "accessed-12" is vp_{23}, "databases-3" is $npsc_3$, "languages-17" is np_{34} as shown in Fig. 4.7. So the following constraint quads will be extracted into the QG:

- (RDF_query_languages, support, subgraph_extraction, 2)

In this constraint quads, the number 2 means that this constraint quads belongs to the second layer of QG, so it is solved second to last.

Therefore, *BotPerception* decomposes a complex query into smaller queries (constraint triples like (subject, predicate, object)) according to its semantic structure. And then, *BotPlanning* determines the order in which these queries are solved based on their semantic relations and adds the layer number into the constraint triples to expand it to constraint quads. The larger the number of layers means that the corresponding constraint quads is located at a more outer layer in QG. This also means that these constraint quads need to be solved before continuing to solve the following constraint quads.

Pattern 3: If in a query Q, the number of WHNP $m = 0$, the number of WHVP $n = 1$, and WHVP is detected at the beginning of the Q. The VP following the interrogative word is represented as $WHVP_{VP} = (whvp_{vp1}, whvp_{vp1}, ..., whvp_{vpc}, ..., whvp_{vpz})$. $whvp_{vpc}$ is the cth word of $WHVP_{VP}$, and there are z words in $WHNP_{NP}$ in total.

If the $whvp_{11}$ is "who" and $WHVP_{VP}$ is not "is", "PERSON" will be added to QG as a category constraint of direct answers. For example, a query "Who created Python?" will be constructed as:

- (PERSON, created, Python, WHVP)

While the $whvp_{11}$ equals to "when", a category constraint "DATE" will be added into QG. If the $whvp_{11}$ is "what" and $WHVP_{VP}$ is not "is", a category constraint $ANYENTITY$ will be put into QG. For the query "What is built on top of the HDF5 library?", the QG will be:

- (ANYENTITY, is_built_on_top_of, HDF5_library, WHVP)

While $ANYENTITY$ represents a wildcard of any entity that means the direct answer should be any entities which meet the property constraint like library, method, application, etc.

Pattern 4: If in a query Q, the number of WHNP $m = 0$, the number of WHVP $n = 1$, and WHVP is detected at the beginning of the Q. The VP following the interrogative word

is "is" and the $whvp_{vp1}$ is not "when" like query "What is Java Servlet?". *BotPerception* and *BotPlanning* will regard this question as a definition query and construct the QG as:

- (ANYENTITY, ANYRELATION, Java_Servlet, WHVP)
- (Java_Servlet, ANYRELATION, ANYENTITY, WHVP)

While $ANYRELATION$ represents a wildcard of any relation.

Pattern 5: If in a query Q, the number of WHNP $m = 0$, the number of WHVP $n = 0$. At the same time, the first word of Q is "List" and the rest of the Q is only an NP. e.g. "List graph database". *BotPerception* will construct the QG as:

- (graph_database, ANYRELATION, ANYENTITY, WHVP)

If the first word of Q is "List", and the rest of the Q is not only a NP. Or the Q is only an (NP, VP, NP). *BotPerception* will regard the "List" as "Which" and construct the QG like **Pattern 1**.

Pattern 6: If INNP is detected at the beginning of a query Q and the following VP is "is a" like query "If Maemo is a mobile operating system?", we will construct the QG as:

- (Maemo, is_a, mobile_operating_system, INNP)

where tfc means this is a true or false question and categorization task.

Pattern 7: If INNP is detected at the beginning of a query Q and the following VP is a VP and not "is a" like query "If HSQLDB support Linux?", we will construct the QG as:

- (HSQLDB, support, Linux, INNP)

where tfp means this is a true or false question and property validation.

4.3.5 *BotReasoning*: Fast Graph Cyclic Pruning Reasoning Algorithm

As described in the 4.3.4, a QG contains category constraints, property constraints and layers of corresponding constraint quads will be constructed. In this section, a fast graph cyclic pruning reasoning algorithm will implement *BotReasoning* and take the query graph QG to perform the reasoning of the direct answers, corresponding explanation extraction and confidence computation, etc.

BotReasoning will extract the layer information of the QG and the answer reasoning will be performed in order from the outer layer to the inner layer. The reasoning order of the constraint quads in the same layer is determined by random. Once a result of constraint quads is found, *BotReasoning* will replace the corresponding object in the lower layer by the reasoning results. For each constraint quads, the reasoning process partially refers to the spread activation model [44] and the semantic memory is corresponding to the knowledge graph we construct in 4.3.3. The search process and decision-making process of the spread activation model correspond to the subgraph search and decision-making process of *BotReasoning*, respectively.

4.3.5.1 Subgraph Search

Assume the current constraint quads for subgraph search is $(c_{subj}, c_{predicate}, c_{obj}, c_{layer})$. In this subsection, we will introduce subgraph search which utilizes the two concepts c_{subj} and c_{obj} in a constraint quads (A constraint quads contains two concepts (subject or called category constraint and object or called property constraint) and a predicate.) to search a subgraph SG from KG. This SG is a graph which most relevant to the two concepts of the constraint quads. The pseudo code of subgraph search is shown in Algorithm. 2.

Assume that a knowledge graph KG is composed of n nodes:

$$KG = [node_1, node_2, ..., node_i, ..., node_n], \tag{4.4}$$

where $node_i$ is the ith node of KG. Each node in KG having an initial associated activation value $a_i \in \mathbb{R}$ and $0 < a_i < 1$. A link $link_{ij}$ connects source $node_i$ to target $node_j$ and the weight of $link_{ij}$ is w_{ij} where $w_{ij} \in \mathbb{R}$ and $0 < w_{ij} < 1$.

The nodes of KG have an active threshold AT, which $AT \in \mathbb{R}$ and $0 < AT < 1$. In the subgraph search process, only the nodes in the KG with activation value exceed the AT will be activated and extracted into subgraph for further evaluation.

There is a decay factor DF where $DF \in \mathbb{R}$ and $0 < DF < 1$. Once activation spreads from one node to another, the activation value will decay according to DF.

At the beginning of the subgraph search process, *BotReasoning* links the concepts c_{subj} and c_{obj} to the corresponding nodes in KG (As the natural language ambiguity problems of nodes is already a research topic in the knowledge graph domain, so this work does not focus it in depth. [145, 146]) Then the initial activation value of these two nodes (assume $node_{subj}$ and

Algorithm 2 Subgraph Search

Input: $(c_{subj}, c_{predicate}, c_{obj}, c_{layer})$: current constraint quads for subgraph search
KG: knowledge graph
ST: iteration times
Output: SG: the searched subgraph, CN: the crossover nodes

Define a set **S** to store the nodes that can spread activation
Link c_{subj} and c_{obj} to nodes $node_{subj}$ and $node_{obj}$ in KG and activate the spreading
Insert $node_{subj}$ into **S**
while $(ST > 0)$ **do**
  if (**S** $\neq \varnothing$) **then**
    Insert **S** into SG_{subj}
    for (each $node_i \in$ **S**) **do**
      Spread activation to every neighbouring $node_j$ according to Equation.4.5
      Adjust activation value of every neighbouring $node_j$ according to Equation.4.6
    end for
    Replace all the nodes of **S** by activated nodes in this round
  end if
  $ST - -$
end while
Insert $node_{obj}$ into **S**
while $(ST > 0)$ **do**
  if (**S** $\neq \varnothing$) **then**
    Insert **S** into SG_{obj}
    for (each $node_i \in$ **S**) **do**
      if (neighbouring $node_j \in SG_{subj}$) **then**
        Insert $node_j$ into CN
      else
        Spread activation to every neighbouring $node_j$ according to Equation.4.5
        Adjust activation value of every neighbouring $node_j$ according to Equation.4.6
      end if
    end for
    Replace all the nodes of **S** by activated nodes in this round
  end if
  $ST - -$
end while
$SG \leftarrow SG_{subj} \cup SG_{obj}$
return SG and CN

$node_{obj}$) will be greater than active threshold AT, which is also called that these two nodes are activated. Once the original nodes are activated, they will initiate to spread the activation to all the neighbouring nodes parallelly.

Assume $node_{subj}$ is the ith node of KG, link $link_{ij}$ connects the source node $node_i$ to the target node $node_j$. While the activation spread from $node_i$ to $node_j$, $node_i$ will compute a_{j_temp} according to the following formula:

$$a_{j\_temp} = a_j + (a_i * w_{ij} * DF). \tag{4.5}$$

Then, a_j will adjust its value to a'_j according to the result of a_{j_temp}:

$$a'_j = \begin{cases} 1, & if a_{j\_temp} \geq 1, \\ a_{j\_temp}, & if AT \leq a_{j\_temp} < 1, \\ a_j, & if a_{j\_temp} < AT. \end{cases} \tag{4.6}$$

As shown in Equation.4.6, if the activation value a_{j_temp} of target node $node_j$ is greater than 1, $node_j$ will be activated and *BotReasoning* will set its new activation value a'_j to 1. If the $AT \leq a_{j\_temp} < 1$, $node_j$ will be activated and the a_j will be renewed to a_{j_temp}. When the $a_{j\_temp} < AT$, $node_j$ can not be activated so the a'_j will keep the ordinal activation value a_j.

Once $node_j$ is activated, $node_j$ will initiate the next spreading activation cycle to the neighbouring nodes of $node_j$. The subgraph search algorithm will repeat to spread the activation according to the above regulars, and every node can only be activated once. During the subgraph search process, the activation spread path from the same source node will be recorded. The spreading will be terminated while the following four situations:

- The spreading arrives the endmost nodes of the KG and no more nodes to spread the activation.
- All the nodes reach a stable activation state, that means all the spreading paths arrive the nodes that cannot be activated and can no longer spread to their neighbouring nodes.
- The activation from two different source nodes arrived at the same node. Once the activation from two different source nodes arrived at the same node, *BotReasoning* will propagate back to the source node according to the recorded paths and marks the backpropagation paths in the subgraph for further evaluation.
- The number spreading iteration times exceed the preset upper bound ST.

4.3.5.2 Decision Making

In the Section 4.3.5.1, the candidate answer CN and corresponding subgraph is extracted from KG. In this section, a decision-making algorithm will evaluate the candidate answers and extract corresponding explanation, and compute the confidence according to SG. Inspire by the reasoning method of paper [44] of the cognitive science domain, here we propose two typical decision-making methods: Bayesian Decision Theory and Deep Neural Networks (DNN). Because Bayesian Decision Theory is more consistent with the decision-making process of the human brain proposed by cognitive scientists [44, 147], and DNN is the algorithm that obtains the best performance in practice. Here we integrate logical rules of human, predicate similarity, the topological structure of subgraph to four features to answer the closed-ended questions.

Predicate Similarity: To measure the semantic relevance between two predicates, a technique named word embedding [1] is adopted to *botReasoning* of *DeveloperBot*. Word embedding transforms the words or phrases into the form of continuous vectors. [1]. With the vectors, the words or phrases can do relevance computation, even can get new words by plus and minus between words. Here we introduce one of the word embedding technique used in *DeveloperBot* call *word2vec*.

word2vec is an unsupervised word representation in natural language processing (NLP) domain. It assumes that words appear in similar context may have similar meanings [148]. Therefore in the embedding space, the semantically similar words are close to each other. The input of *word2vec* is the result of *1 of V*. Each input word will be coded as a vector of size V with all zeros except for the element corresponding to the word's order in the vocabulary. *1 of V* generates a fixed-size vocabulary from the materials with V members in total. So V is the dimension of *1 of V*.

word2vec can use either of the two model architectures proposed by Mikolov et al. to generate a distributed representation of the words: one is called CBOW (Continuous Bag-of-Words Model) and another is called Skip-gram (Continuous Skip-gram Model) [149, 1]. Fig. 4.8 shows the structure of the CBOW model [1]. There are 3 layers in CBOW, they are input layer, projection layer and the output layer, respectively. The CBOW model predicts the current word by utilizing the surrounding context words. And the CBOW model assumes that the sequence of the words of the context does not affect the result of the prediction.

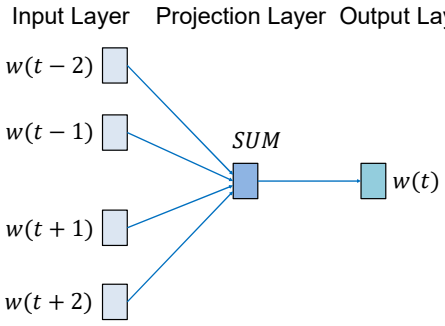


Figure 4.8: The Structure of CBOW [1]

Supposing $w(t)$ is any word in the corpus, the $Context(w(t))$ is the n words before and after the target word $w(t)$. So we have a training sample $(Context(w(t)), w(t))$, and the window is $2n+1$, here we assume the $n=2$. The objective function of CBOW model is to maximize the sum of log probabilities of the surrounding context words conditioned on the center word:

$$\sum_{w \in C} logp(w(t)|Context(w(t))), \tag{4.7}$$

where C is the vocabulary of all words.

For the projection layer, we accumulate the $2n$ vectors from input layer as follows:

$$X_w = \sum_{i=1}^{2n} v(Context(w(t))_i). \tag{4.8}$$

From the projection layer to the output layer there is a Huffman Tree, using to compute the $p(w(t)|Context(w(t)))$.

$$p(w(t)|Context(w(t))) = \frac{e^{y_{w(t)}, i_{w(t)}}}{\sum_{i=1}^{2n} e^{y_{w(t),i}}}. \tag{4.9}$$

Using this equation, we can compute a score to each word according to the vocabulary.

Many studies have shown superior performance of *word2vec* in measuring lexical relevance (e.g., [150, 151]). To achieve the above functions, a word corpus needs to be constructed to train the *word2vec* model. Specifically, this word corpus should be domain-specific and the same context to the word to be represented.

Properties Analysis of Subgraph SG: Fig. 4.9 illustrates an example of SG, the left half is SG_{subj} activated by category constraint of QG, and the right half is SG_{obj} activated by property

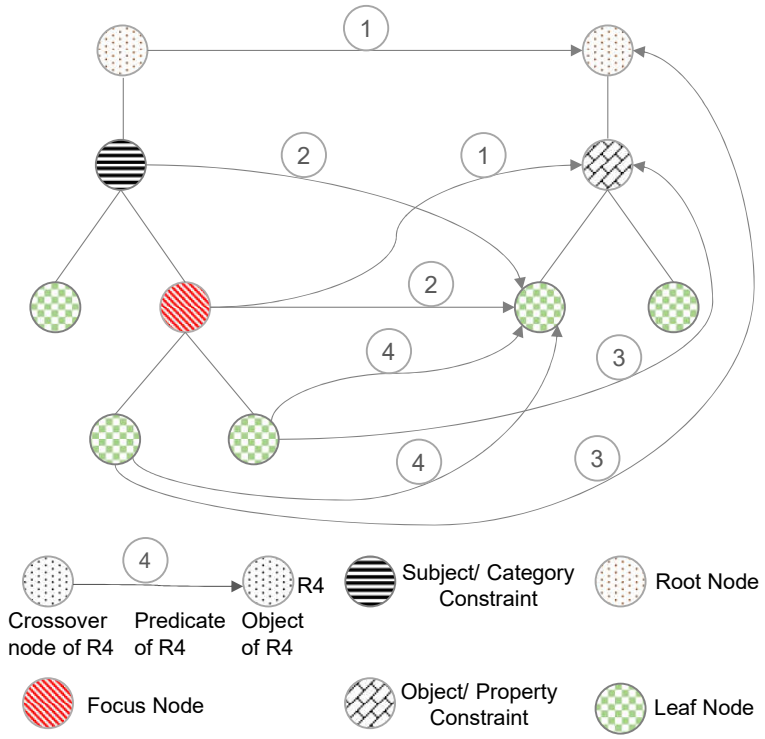


Figure 4.9: Diagram of Topological Structure of SG

constraint of QG. The root node and leaf node refer to the most superclass node and most subclass node of SG respectively.

For the red focus node in Fig. 4.9, assume it is a candidate answer we will do decision-making, the following four kinds of potential topological structure which determine its properties. $R1$ refers to the focus node or any superclass of the focus node is crossover node, and the crossover node connects to the property constraint node or any superclass of the property constraint node by the predicate of $R1$. $R2$ indicates that the focus node or any superclass of the focus node is the crossover node which connects to any subclass of the property constraint node by the predicate of $R2$. $R3$ means that any subclass of the focus node is the crossover node, and connects to the property constraint node or any superclass of the property constraint node by the predicate of $R3$. $R4$ represents a topological structure that any subclass of the focus node is the crossover node and connects to any subclass of the property constraint node by the predicate of $R4$.

In addition to the topological structure, the predicates of relations also affect the properties of the focus node. In natural language, there are many different ways of expressing the same

meaning. Like the "Mysql support Ubuntu", the same meaning also can be expressed as "Mysql can run on Ubuntu", "Mysql can be used on Ubuntu". Give a predicate to compare with the predicate of a relation of a focus node, if these two predicates do not express the opposite meaning, the relation will be called positive relation. On the contrary, give a predicate "support", if the relation of a focus node has a predicate "do not support", this relation will be called negative relation. From the above description, we know that this positive relation and negative relation may appear in any of $R1$, $R2$, $R3$, and $R4$ at the same time. Therefore, the traditional methods of weakening the influence of predicates are not suitable for accurate answer extraction [11].

In the knowledge graph, generally, the properties in a superclass indicate that all the entities in its subclass also have the same properties. For example, if a relation triples in KG is (ArangoDB, support, Linux). This means all the nodes in the subclass of ArangoDB (e.g. "ArangoDB_3.5", "ArangoDB_3.6", etc.) have the property "support Linux". Similarly, ArangoDB and its subclasses have the property that supports all the Linux distributions at the subclasses of Linux (e.g. "Ubuntu", "Debian", etc.). To the contrary, the relation triples (ArangoDB, support, Ubuntu) can not deduce (ArangoDB, support, Linux).

Features Generations for Closed-end Question: The human brain has different reasoning logics and strategies for different questions. Here we mainly focus on the reasoning logic of the closed-end question. The analysis of true or false question-categorization task or true and false question-function validation task will be discussed in the future works. In order to find the right answers from candidate answers, sufficient evidence needs to be gathered to determine whether it surpasses the positive or negative criteria.

Table 4.3: Types of Relations Found in SG that Constitute Positive Or Negative Evidences

No.	Positive Evidence	Negative Evidence
1	The candidate node has positive $R1$	The candidate node has negative $R1$
2	The candidate node has positive $R2$	The candidate node has negative $R2$

Tab. 4.3 lists the positive and negative evidence that can be used to make different decisions. In general, when property comparison, the weights of positive and negative evidence are not equal. A few negative evidence may lead to negative decisions, whereas many positive evidence are needy in order to make a positive decision.

As we knew, the human can not only answer a question by the existing knowledge in the brain but also can give a prediction to the questions which do not have direct mapping knowledge but have some related knowledge and give a confidence value about it. For example, a question "Which graph database support Linux?", "ArangoDB" is one of the leaf node activated by category constraint "graph database". There are neither positive direct relation (ArangoDB, support, Linux) nor negative relation (ArangoDB, do_not_support, Linux) in the SG. But there are relations (ArangoDB, runs_on, centos), (ArangoDB, runs_on, fedora), (ArangoDB, runs_on, red_hat), (ArangoDB, runs_on, debian), (ArangoDB, runs_on, ubuntu), (ArangoDB, runs_on, suse), (ArangoDB, runs_on, arch_linux). And in our knowledge graph, "centos", "fedora", "red_hat", "debian", "ubuntu", "suse" and "arch_linux" are subclass of Linux, so these relations are $R2$ relations. But these relations do not cover all subclasses of Linux, there are still some subclasses like "linux_mint" and "elementary_os".

Table 4.4: Features to Represent a Focus Node

#	Name	Gloss
1	p_r1_pdictSim	If positive $R1$ does not exist, the value takes 0. Otherwise, takes the maximum value of predicates similarity.
2	p_r2_pdictSim	If positive $R2$ does not exist, the value takes 0. Otherwise, the value takes the sum of predicates similarity of all $R2$ divide the number of subclasses of property constraint.
3	n_r1	If negative $R1$ does not exist, the value takes 0. Otherwise, takes 1.
4	n_r2	If negative $R2$ does not exist, the value takes 0. Otherwise, takes 1.

For a SG generated by a constraint quads, the candidate answers are all the leaf nodes of the SG activated by category constraint. For the $R1$, we will give the properties of superclasses and the nodes between the property constraint and leaf nodes to all the corresponding leaf nodes. Combine the description of **Words Representation** and **Properties Analysis of Subgraph** SG, we synthesize the above properties into four features to represent a focus node (i.e. the candidate answer) as shown in Tab.4.4. In these four features, feature #1 named p_r1_pdictSim represent the influence of $R1$. If positive $R1$ does not exist, the value of feature #1 takes 0. Otherwise, the value of feature #1 takes the maximum value of predicates similarity of all $R1$:

$$p\_r1\_pdictSim = max(pdictSim\_R1_i), i = (1, 2, ..., n), \tag{4.10}$$

where $pdictSim\_R1_i$ is the predicates similarity of ith $R1$ computed by the method of **Words Representation**, n is the number of $R1$.

Feature #2 named p_r2_pdictSim represent the influence of $R2$. If positive $R2$ does not exist, the value takes 0. Otherwise, the more subclasses of property constraints connected by focus nodes, the higher the probability that the focus node meets the property constraint. So the value of feature #2 takes the sum of predicates similarity of all $R2$ divide the number of subclasses of property constraint:

$$p\_r2\_pdictSim = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} pdictSim\_R2_{ij}, i = (1, 2, ..., m), j = (1, 2, ..., n), \tag{4.11}$$

where $pdictSim\_R2_{ij}$ is the predicates similarity of ith leaf nodes of c_{subj} connected to jth subclass of c_{obj}. m is the number of leaf nodes of c_{subj} and n is the number of subclasses of c_{obj}. Take the Fig. 4.9 as a example, here $m = 2$, and $n = 2$, assume the value of $pdictSim\_R2_{21} = 1$ and $pdictSim\_R2_{22} = 0.5$, the value of feature #2 is $\frac{(1+0.5)}{4} = 0.375$. For the feature #3 and feature #4, if negative $R1$ or $R2$ do not exist, the value takes 0, otherwise, takes 1.

Bayesian Decision Theory: Here we briefly review Bayesian decision theory [152]. Bayesian decision theory is a statistical method to the problem of pattern classification. Different from frequentist theory, Bayesian approaches integrate human prior knowledge and data statistics, which is very suitable for classifying the cases with some prior knowledge and a small number of sample data [153]. It assigns discriminant functions $g_i(x)(i = 1, 2, ..., m)$ to classes $\omega_i(i = 1, 2, ..., m)$, respectively.

For a given observation x, the posterior will govern our decision:

$$\omega^* = arg\max_i p(\omega_i|x), \tag{4.12}$$

where $p(\omega_i|x)$ is a posteriori probability density of x.

According to Bayesian formula:

$$\begin{aligned} p(\omega_i|x) &= \frac{p(x|\omega_i)p(\omega_i)}{p(x)} \\ &= \frac{p(x|\omega_i)p(\omega_i)}{\sum_{i=1}^{m} p(x|\omega_i)p(\omega_i)}, \end{aligned} \tag{4.13}$$

where $p(\omega_i)$ is prior distribution and $p(x|\omega_i)$ is class-conditional density function.

For the situation of two classes, the probability of error is:

$$p(error|x) = \begin{cases} p(\omega_1|x), & \text{if we decide } x \text{ belong to } \omega_2, \\ p(\omega_2|x), & \text{if we decide } x \text{ belong to } \omega_1. \end{cases} \tag{4.14}$$

So according to the minimum error rate for Bayesian decision theory:

$$g(x) = g_1(x) - g_2(x) \begin{cases} > 0, & \text{we decide } x \text{ belong to } \omega_1, \\ < 0, & \text{we decide } x \text{ belong to } \omega_2. \end{cases} \tag{4.15}$$

Obviously, in order to apply the Bayesian decision method, we need to know:

- prior distribution $p(\omega_i)$, $(i = 1, 2, ..., m)$
- class-conditional density function $p(x|\omega_i)$, $(i = 1, 2, ..., m)$

If we assume $p(x|\omega_i)$ is follow the Gaussian distribution and x is 1-dimensional continuous random variable, then $p(x|\omega_i)$ is given by:

$$p(x|\omega_i) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}, \tag{4.16}$$

where μ is the mean or expectation of the Gaussian distribution, σ^2 is its variance.

According to Bayesian formula 4.13 and minimum error rate, Bayesian discriminant function $g_i(x)$ and $g_(x)$ is given by the following:

$$g_i(x) = \ln_p(x|\omega_i) + \ln_p(\omega_i) - \ln_p(x), \tag{4.17}$$

For any two classes ω_i and ω_j:

$$\begin{aligned} g_(x) &= \ln_p(x|\omega_i) + \ln_p(\omega_i) - \ln_p(x) - \ln_p(x|\omega_j) - \ln_p(\omega_j) + \ln_p(x) \\ &= \ln_{p(x|\omega_i)} - \ln_p(x|\omega_j) + \ln_p(\omega_i) - \ln_p(\omega_j). \end{aligned} \tag{4.18}$$

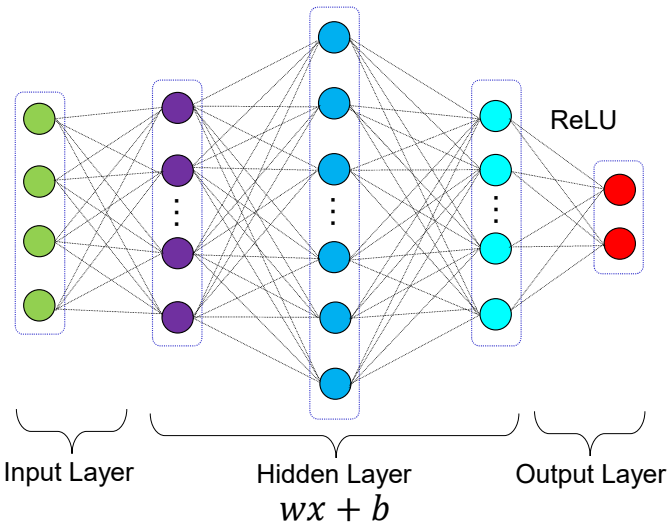


Figure 4.10: Deep Neural Network

For the closed-end question, the m of $\omega_i (i = 1, 2, ..., m)$ is two. ω_1 means to accept this answer as the right answer, and ω_2 means reject this answer.

Deep Neural Network: Fig.4.10 is the framework of the deep neural network used by our system. A DNN classifier with three hidden layers is created.

In the input layer, there are input features $x_i, (i = 1, 2, ..., m)$, where the m is the number of input features. Following is the hidden layer, all layers are fully connected. The first and the second hidden layer contain 10 units and the third hidden layer contains 20 units. The output layer is presented as $a_j, (j = 1, 2, ..., n)$, where the m is the number of classes.

In each hidden layer, the input is $z^p, (p = 1, 2, ..., P)$, where P is the number of input features, it equal to the output unit of the pervious layer. The x will be fed into the node of the first hidden layer, so $P = m$ for the first hidden layer. z^p will be multiplied by a corresponding weight w^p. Then the sum of those results of all input feature will plus a bias $b_k, k = (1, 2, ..., K)$, where k is the serial number of the hidden layers, K is the number of hidden layers. These processes is presented as follow:

$$z^{pk+1} = \sum_{p=1}^{P} z^{pk} \times w^{pk} + b_k \tag{4.19}$$

where z^{pk+1} is the result of p unit of $(k + 1)$th hidden layer. The w^{pk} is the weight of p unit of kth hidden layer.

The result of the last hidden layer will be fed into an activation function. In this case, the activation function is a rectified linear unit (ReLU). Finally, the activations are summed going into the output nodes.

$$a_j = \sum_{p=1}^{P} ReLU(z^{pK} \times w_j) \tag{4.20}$$

where the z^{pK} is the output of last hidden layer, the w_j is the weight of j output node.

After that, we can minimize the softmax cross-entropy loss over all labelled node between the ground-truth and the prediction. With the guide of labelled data, we can optimize the proposed model via backpropagation and learn the embeddings of nodes.

4.3.5.3 Final Answer and Explanation Generation

BotReasoning extracts the layer information of QG and the answer reasoning will be performed from the outer layer to the inner layer. The reasoning order of the constraint quads in same layer is determined by random. Once the answers $AS = (as_1, as_2, ..., as_n)$, where $n \geq 0, n \in \mathbb{Z}$ of a constraint quads $currentCQ$ are found, *BotReasoning* will replace the corresponding object constraint in the inner layer by these answers and prune the $currentCQ$ from QG. *BotReasoning* will generate n pruned QG, and continues the reasoning process for every QG. If there are multiple constraints in the same layer and multiple answer sets are deduced, the union of these sets is computed. These iterations will terminate until the final answers and explanations of the most inner layer of every QG are achieved.

The corresponding explanations (reasoning subgraph and answer confidence) are extracted following the cognitive process of *DeveloperBot*. *BotPerception* records the entities identified in a query to the reasoning subgraph, these are also the initially activated nodes ($node_{subj}$ and $node_{obj}$) for spreading activation in the knowledge graph. After every constraint reasoning, *BotReasoning* takes all the CR and the relational paths from $node_{subj}$ or $node_{obj}$ to any nodes of CR into the reasoning subgraph of previous iteration (if any). Therefore, the final reasoning subgraph involves the information of whole cognitive process including *BotPerception*, *BotPlanning* and *BotReasoning*.

The answer confidences $AC = (ac_1, ac_2, ..., ac_m)$, where $m \geq 0, n \in \mathbb{Z}$ for $currentCQ$ are computed by the decision-making algorithm multiplicative the answer confidence from

outer layer. During the answer reasoning, the confidences ac_i will be multiplicative to every confidence of the inner layer. The confidence of an answer reasoned from multiple constraints in the same layer is obtained by multiplying the answer confidences of all the constraints in the same layer.

4.3.6 *BotResponse*: UI of *DeveloperBot*

A proof-of-concept tool of *DeveloperBot* is implemented to developers with the interactive searching interface as shown in Fig. 4.11. The user can enter their query, for example, "Which IDE support web2.0?" at the input bar and press button "Direct Search". The application will search the backend knowledge graph and returns a ranked list of direct answers. For example, the returned answers in Fig. 4.11 for the sample question is pycharm, aptana, etc. There are also brief introductions of corresponding answers shown under the direct answers.

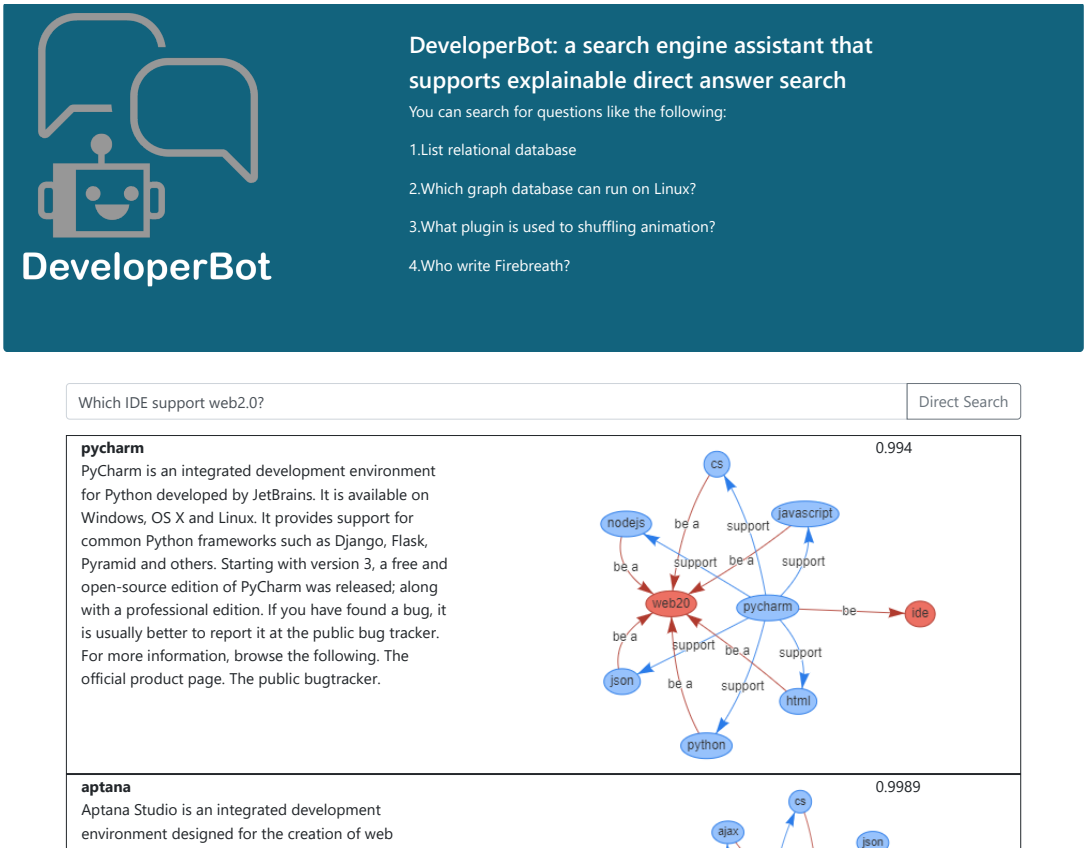


Figure 4.11: *botResponse*: the User Interfaces of *DeveloperBot*

At the right side of each direct answer, there is a reasoning subgraph which explains why this answer is recommended. e.g. pycharm is recommended because Pycharm is an IDE, Pycharm supports noode.js and noode.js is one of the web2.0 technique, etc. These explanations represent in the reasoning graph are (pycharm, be, ide), (pycharm, support, noodejs), (noode.js, be_a, web20). The entities extracted by *DeveloperBot* from the query are presented as red nodes. The rightmost number present the confidence of the corresponding direct answer. Which is computed according to the predicate similarity and topological structure of extracted subgraph by query.

4.4 Experiment

In this section, a series of experiments are conducted to illustrate the performance of *DeveloperBot* on complex questions answering, decision-making for candidate answers, explainability and the acceptability of users, etc. We report these experiments to answer the four research questions about the effectiveness and usefulness of our approaches:

- RQ1: How is the performance of different decision-making algorithms in identifying the correct answer?
- RQ2: How is the usability of the prototype of *DeveloperBot*?
- RQ3: Whether *DeveloperBot* can improve the performance of developers during answer search? If the performance is different under different task complexity?
- RQ4: How do users comment on the explanations provided by *DeveloperBot* and how these explanations affect user behaviour during answer search?

The result of user study shows that compared with just using Google, with the assist of *DeveloperBot*, users can not only find answers faster and with more accuracy, but also understand the answers more deeply. Furthermore, the more complex the question is, the more effective the *DeveloperBot* can achieve. At the same time, the explanation of the answers can significantly improve the users' trust and adoption of the answers.

4.4.1 Data Acquisition

Knowledge Graph: The original knowledge graph[1] is the one proposed in Chapter. 5. This knowledge graph involves the general knowledge of the software engineering domain and contains 44800 concepts, 9660 unique verb phrase and 35279 relation triples. Then, this knowledge graph is cleared up to drop some unreasonable relation triples and augmented by the synonyms, expertise of domain experts and NLP techniques. After the augmentation, the final knowledge graph acquires 39022 concepts, 8173 unique predicates and 35938 relation triples. These knowledge cover the popular programming languages(e.g., Java, Javascript, Python, etc.), frameworks(e.g., Flex, Django, etc.), database(e.g., neo4j, Mysql, etc.), tools(e.g., d3, MSBuild, etc.) of software engineering domain.

Training Corpus of Word Embedding: The tag wiki of Stack Overflow is used as the training corpus. Tag wiki is not only well maintained but also have standard grammatical structure and comprehensive technical coverage. Furthermore, then the semantic context of tag wiki matches to the knowledge graph well. As a result, we acquire 20539 documents, then split 78466 sentences from the documents as training corpus of word embedding.

4.4.2 Experimental Setup and Involved Tools

In the subgraph search process of this experiment, the initial activation values of all nodes are set to zero. Here the active threshold AT is set to 0.8 and the decay factor DF is set to 0.85. For starting the activation, the activation values of linked nodes of subject constraint and object constraint will be initialized to greater than AT. The maximum number of iterations is set to 30. DNN model used in this experiment is a 5 layers classifier whose hidden layer is set to [10, 20, 10]. For the decision-making process of Bayesian Decision Theory, the $p(\omega_i = 0) = 0.15, p(\omega_i = 1) = 0.85$.

The *Stanford CoreNLP* is used to do the NLP mark up, dependency parse [154] and named entity recognition. The word2vec library of *Gensim* is used to compute word embedding [155]. The *nltk* is used to do the VP, NP chunking [156]. *neo4j* is used to store the final relation triples for supporting the knowledge graph search and result visualization [157].

[1] http://neo4j.tuntunkun.org/xuejiao

4.4.3 Evaluation

In this section, quantitative evaluation and a user study are conducted to evaluate the performance of *DeveloperBot*. The quantitative evaluation will evaluate some quantitative metrics like the accuracy of answers in decision-making, MSE of confidence, etc. The user study will show that compared with just using the search engine, how *DeveloperBot* can assist the search engine during information retrieval of developers.

4.4.3.1 Quantitative Evaluation

In the quantitative evaluation, 109 ground truth questions made by domain experts generated 1819 candidate answers are used for training the decision-making model. Tab. 4.5 shows some example of the ground truth questions and corresponding answers.

Table 4.5: Example of the Ground Truth Questions and Corresponding Answers

#	Question	Answer
1	Which programming languages support by Tango.	C,C++,Java,Python,MATLAB, LabVIEW,IGOR Pro
2	Who develops delphi?	Anders Hejlsberg
3	Which java library supports system events monitoring?	JNotify
4	Which graph database supports Gremlin?	orientdb,infinitegraph, amazon neptune,sparksee

To evaluate the answer accuracy and mean squared error (MSE) of confidence of the decision-making process, these questions and answers are used to do 10-fold cross-validation. In every iteration, 9 folders of the questions are used to train the decision-making algorithm, and 1 folder is used to do testing. The average of the 10 times cross-validation will be taken as the final results of different metrics.

Here we used balanced accuracy instead of accuracy due to the training data is unbalanced. Balanced accuracy returns the average accuracy per class:

$$BalancedAccuracy = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right), \tag{4.21}$$

where TP, TN, FP, FN are true positives, true negatives, false positives, and false negatives, respectively.

In addition to balanced accuracy, precision, recall and f1-score are also used as evaluation metrics to evaluate the performance of decision-making process of *DeveloperBot*. Here precision is a metric to measure the quality and reliability of answers. It expresses the proportion of the true right answer to decide by our algorithm. The recall is a measure of completeness, it refers to the percentage of total right answers decided by our algorithm. F1-score combines precision and recall is the harmonic mean of precision and recall [158]:

$$Precision = \frac{TP}{TP + FP}, \tag{4.22}$$

$$Recall = \frac{TP}{TP + FN}, \tag{4.23}$$

$$F1 - score = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{4.24}$$

4.4.3.2 User Study

Participants: In the user study, 24 developers from different backgrounds were recruited.

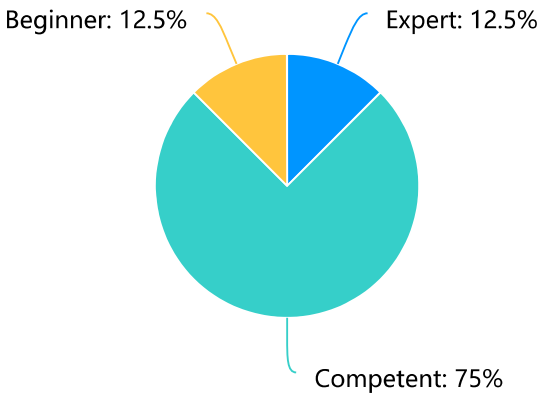


Figure 4.12: Programming Proficiency of Participants

Some of them are the employees or Ph.D students from the School of Computer Science and Engineering, Nanyang Technological University. Some of them are professional developers from IT companies like NTT, ST Engineering, etc. Some of the developers are undergraduate students or professors from computer science major. There are even some participants who come

from the mechanical engineering major. Based on our pre-experiment survey, all the participants have experience in programming. These developers are represented to P1,P2,...,P24 and their proficiency is shown in Fig. 4.12.

The participants also have a diverse programming background and their programming languages involved Python, Java, Javascript, SQL, C#, etc. as shown in Fig.4.13.

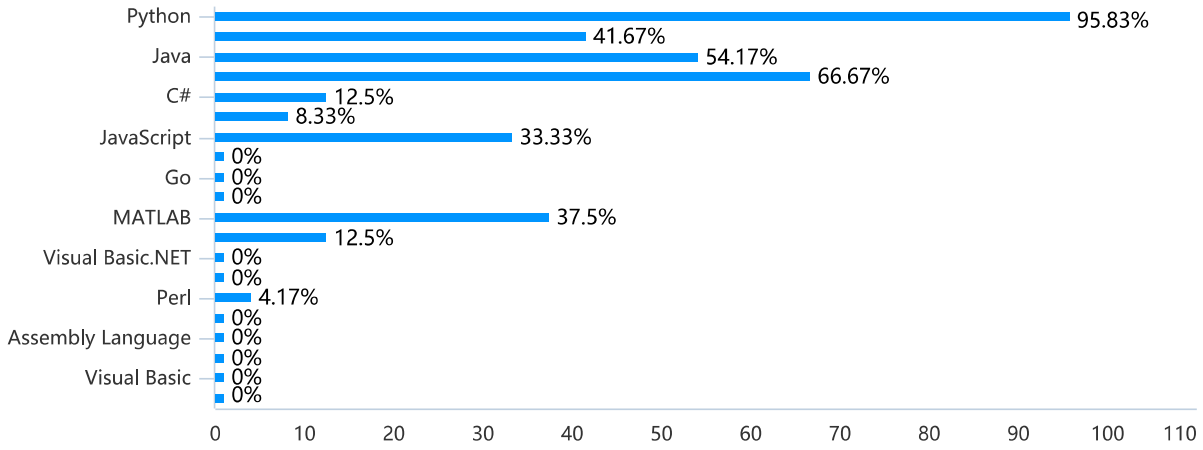


Figure 4.13: The Top 5 Programming Languages most Used by the Participants

In each proficiency level, developers are randomly assigned to two equal-sized groups: G1 and G2. All of them use Google very frequently and none of them claims to be familiar with the questions in the experimental tasks. In order to exclude the effect of experience differences and fatigue of the users, these 24 developers will act as experimental group and control group circularly as shown in Tab. 4.6.

Table 4.6: The Tool (s) the Participants Used for Each Task

Task \ Participant	G1	G2
1	Google	Google + *DeveloperBot*
2	Google + *DeveloperBot*	Google
3	Google	Google + *DeveloperBot*
4	Google + *DeveloperBot*	Google
5	Google	Google + *DeveloperBot*

Task: In this study, we have chosen 5 answer search tasks (represent as $Task1$ $Task5$) from the ground truth questions. The answer to these tasks cover celebrity of software engineering domain, library, operating system, IDE, and database, etc. as shown in Tab. 4.7. According to the preliminary test, the difficulty of these five tasks increases in order. According to the Tab. 4.6, every developer has the opportunity to solve the simple or hard task by only Google or Google + *DeveloperBot*. So they have a comprehensive comparison to the differences with or without the search engine assistant *DeveloperBot*. For all the tasks, the participants need to evaluate all the answers given by Google or *DeveloperBot* comprehensively, and select x best answers according to the required number of every task [1].

Table 4.7: The Tasks for User Study

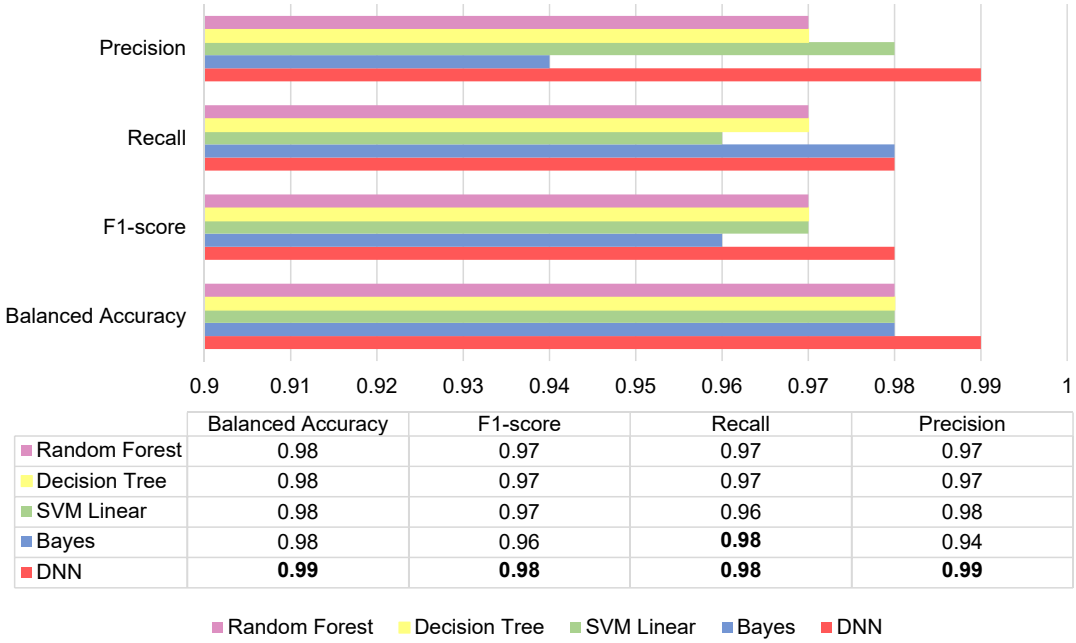
Task No.	Task Description	Number of Best Answer(s)
1	Who develops Hiawatha?	1
2	List data visualization library	5
3	Which mobile operating system is based on Linux kernel?	4
4	Which IDE support web2.0?	3
5	Which graph databases support Python and can be accessed through the RDF query languages that support subgraph extraction?	3

Procedure: This user study begins with a brief introduction of *DeveloperBot* by the experimenter. Then the experimenter demonstrates and explains all the features of *DeveloperBot* to the participants step-by-step and asks the participants to walk through two training tasks to familiarize the features of *DeveloperBot* under the guidance of the experimenters. After the training, the participants will work on the five tasks without interventions of the experimenters. After finishing each question, participants of both groups are asked to provide their confidence to the information they collect (on 5-point likert scale with 1 being least confident and 5 being most confident). A screencapture software is required to run throughout the whole process which allows us to analyze the participants' behaviour after the experiment. In the end, all the participants will be asked to fill in the System Usability Scale (SUS) questionnaire [159] and do an open discussion focusing on their options about the different features of *DeveloperBot*.

[1] Note that the x is known to the developers.

4.4.4 Results and Results Analysis

4.4.4.1 Result: performance of different decision-making algorithms (RQ1)



	Balanced Accuracy	F1-score	Recall	Precision
Random Forest	0.98	0.97	0.97	0.97
Decision Tree	0.98	0.97	0.97	0.97
SVM Linear	0.98	0.97	0.96	0.98
Bayes	0.98	0.96	**0.98**	0.94
DNN	**0.99**	**0.98**	**0.98**	**0.99**

Figure 4.14: Performance of Different Decision-making Algorithms

Here we will present the result of quantitative evaluation and answer the RQ1. Fig.4.14 shows the performance of different decision-making algorithms including Random Forest, Decision Tree, SVM Linear, Bayes (Bayesian Decision Theory), and DNN. With the accurate representation of candidate answers by the features extracted by our algorithm, any decision-making algorithm can get 0.98 balanced accuracy. Compare with other algorithms, DNN achieves the highest precision 0.98, recall 0.98 and f1-score 0.99, respectively. From Fig.4.14, we can infer that some decision-making algorithm can achieve high precision, but at the cost of a low recall, like SVM Linear. The Bayesian Decision theory gets a high recall, but the precision is low.

Fig.4.15 is the MSE of confidence in different decision-making algorithms. Fig.4.15 shows that DNN achieves the lowest MSE for the estimation of answer confidence. Although Bayesian Decision Theory gets the highest MSE, for all decision-making algorithm, the estimation of answer confidence is within the acceptable range.

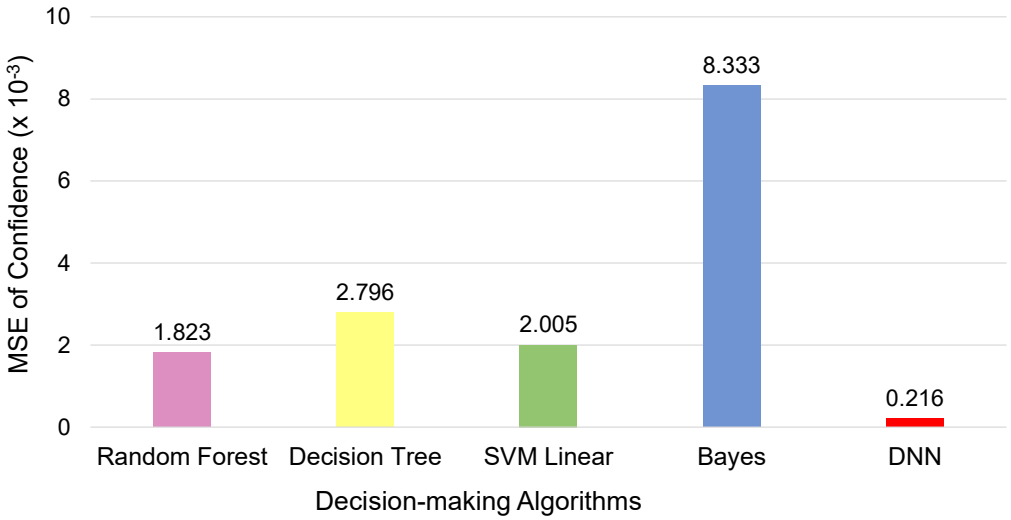


Figure 4.15: MSE of Confidence of Different Decision-making Algorithms

To explore the effects of different features, we train 3 DNN models using different features. These 3 models use all features, only predicate similarity feature and only topological structure feature, respectively. The only predicate similarity feature is computed by the maximal predicate similarity of $R1$, if it exists. Only topological structure features involve 4 features, the 1 of first and second features represent the existing of positive $R1$ and any positive $R2$, the 1 of third and fourth features represent the existing of negative $R1$ and any negative $R2$.

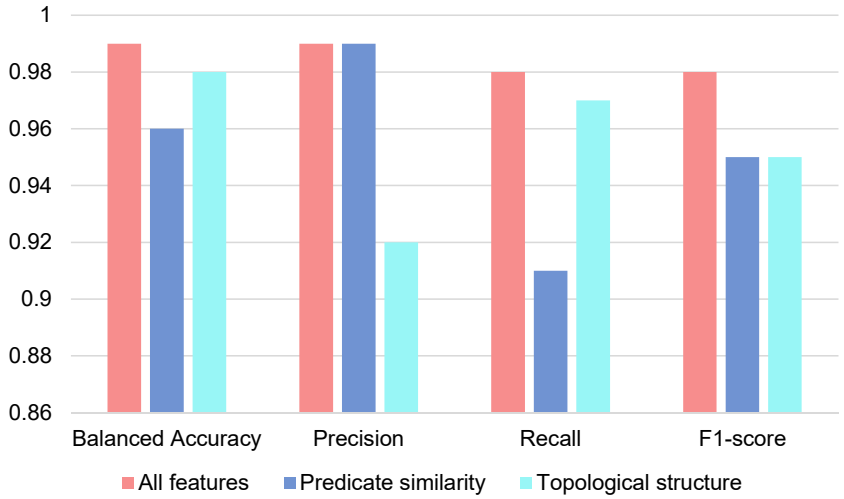


Figure 4.16: Performance of Decision-making Algorithms with Different Features

As shown in Fig.4.16, the decision-making algorithm with all features performs best for the metrics like balance accuracy, precision, recall and f1-score. This figure also indicates that the predicate similarity contributes to the precision significantly, because, with only predicate similarity feature, the precision almost can achieve as same as all features. But at the same time, only predicate similarity feature obtains very low recall. This shows that the predicate similarity is a good feature in terms of identifying the right answers, but without the topological structure, a lot of right answers are missed.

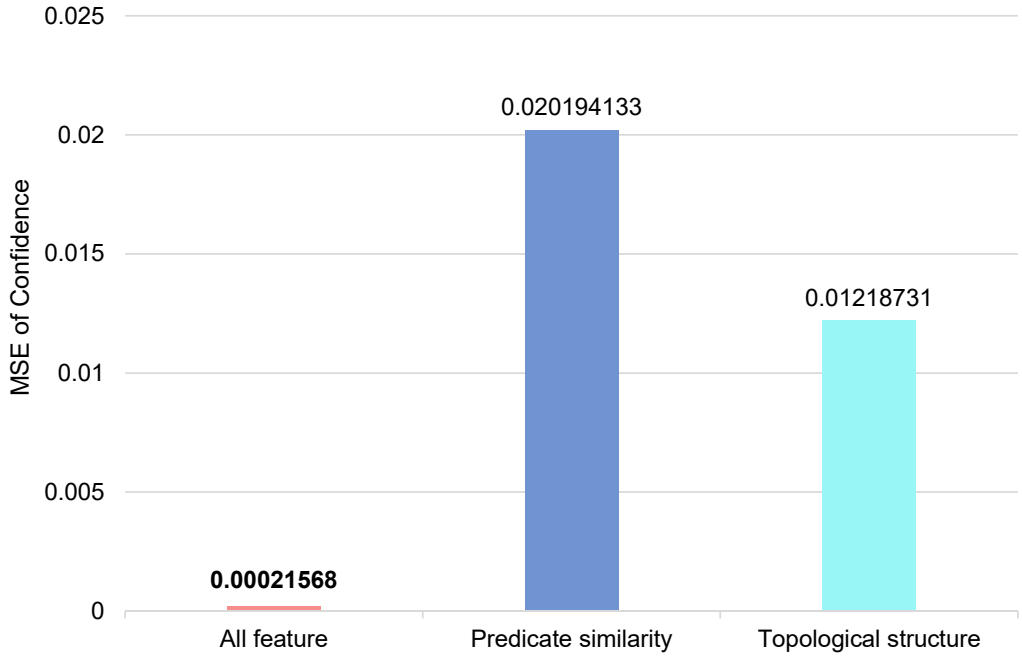


Figure 4.17: MSE of Confidence with Different Features

Fig.4.17 indicates that the decision-making algorithm can get the lowest MSE by using all features. Compared with the topological structure, predicate similarity achieves higher MSE. From the above exploration, we can infer that both predicate similarity and topological structure we design for decision-making are useful for identifying the right answer from the candidate answers.

4.4.4.2 Result: usability of the prototype of *DeveloperBot* (RQ2)

Five-point Likert scales is used to evaluate a system's usability and acceptability. The participant should finish ten system design and usability questions in the System Usability Scale (SUS)

questionnaire. The answers of this questionnaire anchor with $1 = StronglyDisagree$, $2 = MildlyDisagree$, $3 = Neutral$, $4 = MildlyAgree$ and $5 = StronglyAgree$.

Fig. 4.18 shows that most of the participants agree or strongly agree that *DeveloperBot* is easy to use and they can learn to use it very quickly. They also feel the features of the *DeveloperBot* are well integrated, they are confident during using *DeveloperBot* and they would like to use *DeveloperBot* frequently.

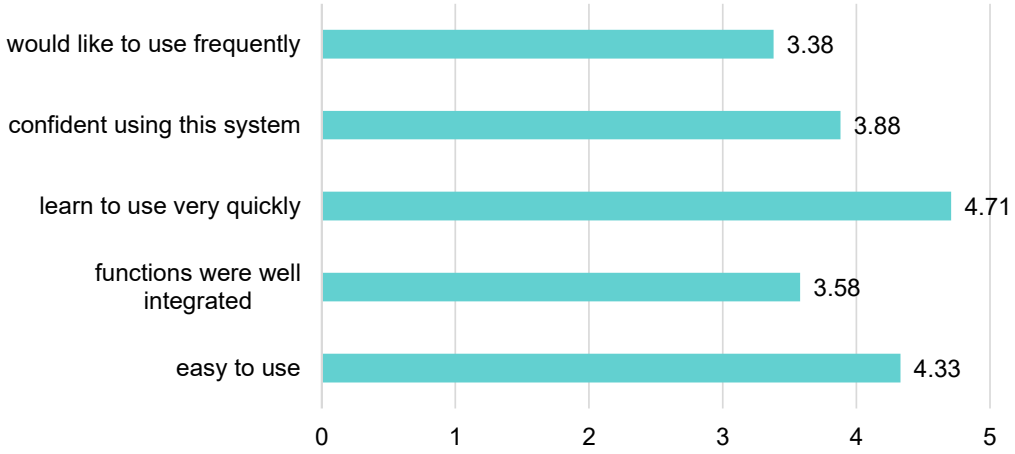


Figure 4.18: Average Score of Positive Metric of SUS

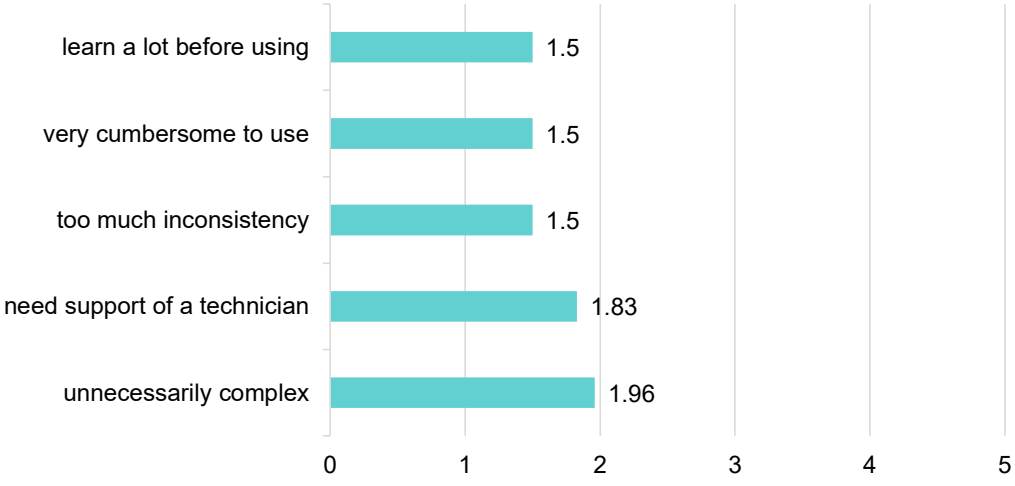


Figure 4.19: Average Score of Negative Metric of SUS

Fig. 4.19 further confirms the simplicity and consistency of *DeveloperBot*. Furthermore,

the "would-like-to-use-frequently" metric and "confident-using-this-system" metric of the *DeveloperBot* are 3.38 and 3.88, respectively, which indicate that participants appreciate the assistance of the *DeveloperBot* and prefer to use *DeveloperBot* during information retrieval.

4.4.4.3 Result: accuracy, confidence and speed improvement (RQ3)

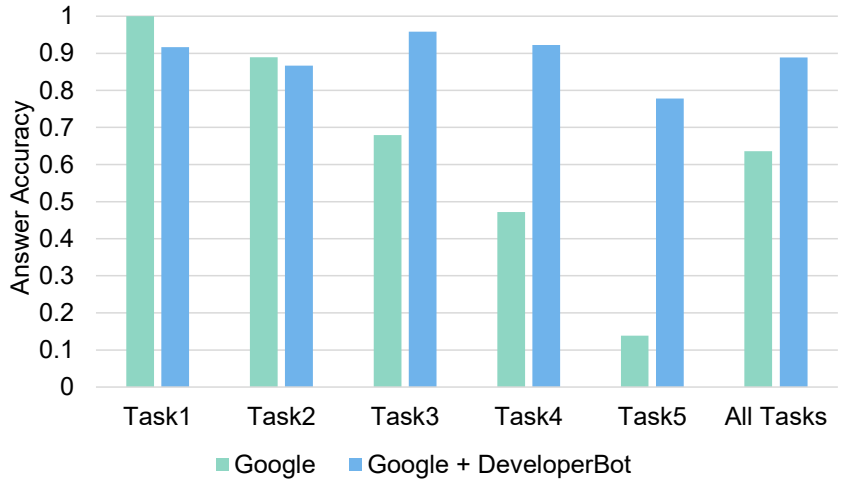


Figure 4.20: Answer Accuracy for Each Task

Here we present the result of the first research question: whether *DeveloperBot* can improve the accuracy, confidence, and speed of developers in answer search? The answer accuracy here refers to the correct answer rate for the number of best answers required for each task. Fig.4.20 is the result of the average answer accuracy of the two groups (12 participants per group). It shows that for the task 1 and task 2, both the participants who use only Google and the participants who use only Google + *DeveloperBot* can finish the tasks with high accuracy (above 0.86). The answer accuracy of the participants who use only Google even reaches 1 on task 1. These prove our pre-experiment conjecture: Google has a very good performance in simple information search tasks.

On the other hand, compared with just using Google, the participants who use Google + *DeveloperBot* can finish the task 3, task 4 and task 5 with higher answer accuracy. Furthermore, the answer accuracy of the participants who only use Google gradually decreases as the complexity of the task increases and has fallen to 0.14 for task 5. But the answer accuracy of the participants who use Google + *DeveloperBot* has been maintained at a high level (above 0.78).

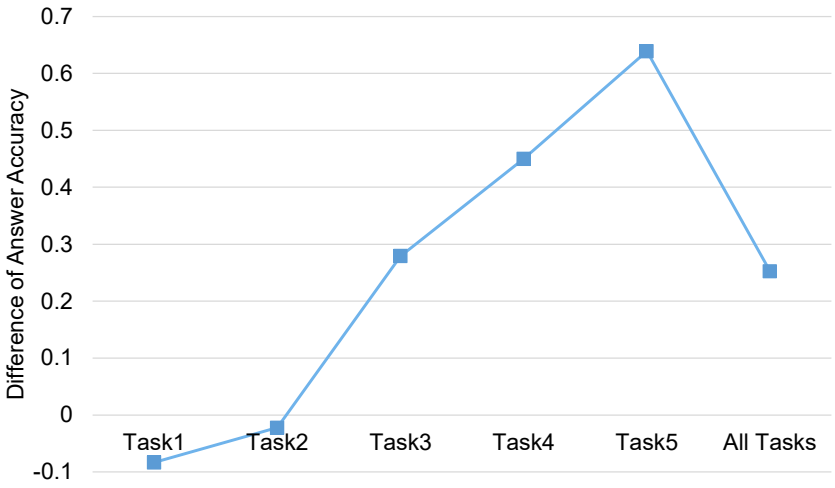


Figure 4.21: The Trend of the Difference of Answer Accuracy with Increasing Task Complexity

This trend can be seen in Fig. 4.21 more clearly. With the complexity of the tasks increasing, the Google + *DeveloperBot* group shows an overwhelming advantage in the answer accuracy, and their difference shows a clear trend of increasing. For task 5, the difference in answer accuracy even reaches 0.64. These fully illustrate that *DeveloperBot*, as a search engine assistant, can significantly improve the answer accuracy during search closed-end questions, especially in complex search tasks.

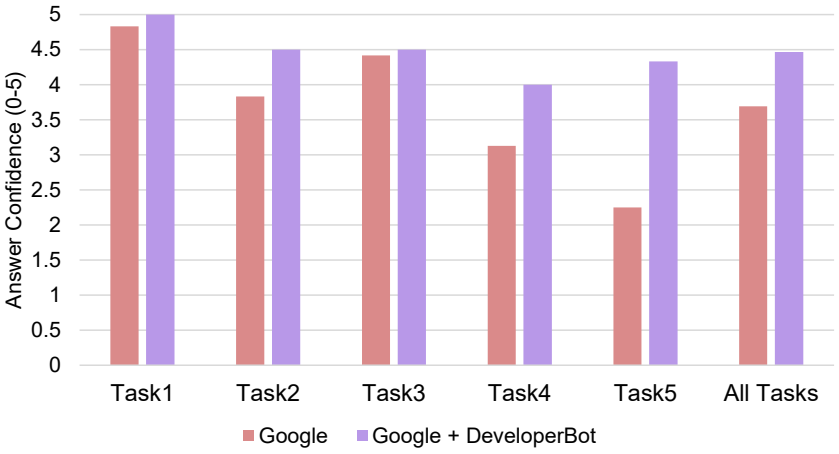


Figure 4.22: Answer Confidence for Each Task

Fig.4.22 is the result of the average answer confidence of the user study. This figure indicates

that for the easy tasks like task 1, task 2 and task 3, the participants' confidence of the answers by using Google is almost the same as using Google + *DeveloperBot*. But for the complex tasks like task 4 and task 5, the answer confidence of participants just using Google decline steeply while the participants using Google + *DeveloperBot* have maintained a relatively high value (4 and 4.3 for task 4 and task 5, respectively). These indicate that for the easy tasks, the participants using only Google are more confident with the answers due to the answers are straightforward and do not need reasoning or summarization. For the complex tasks, even they pay a long time on answer search, they still feel less confident to the answers because the participants may lose during they try to summarize and reason the right answers through many web pages.

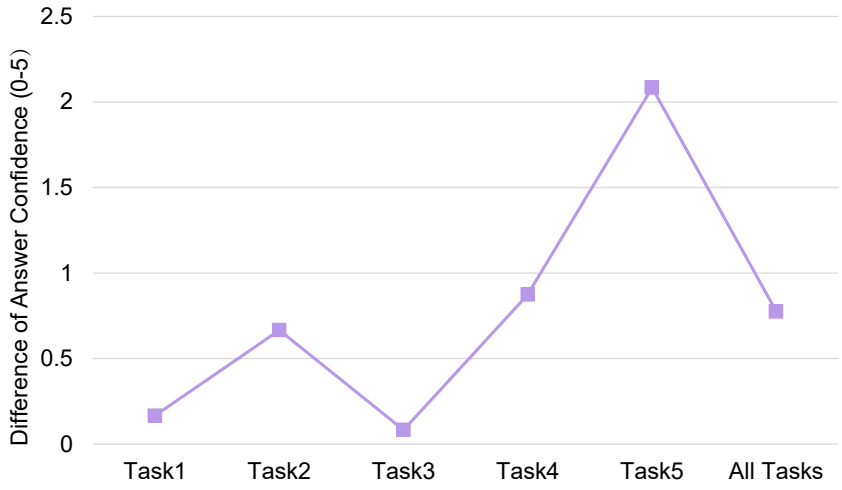


Figure 4.23: The Trend of the Difference of Answer Confidence with Increasing Task Complexity

Fig. 4.23 indicates that, with the complexity of tasks increasing, the difference of answer confidence of Google + *DeveloperBot* group shows a clear increasing trend except for task 3. The confidence values of task 3 of the only Google group and Google + *DeveloperBot* group are close (4.42 and 4.5, respectively). According to the observation of the screen recording and results of open discussion, this is because there is a web page that lists some direct answers of task 3, and each answer has corresponding text explanation. The answer confidence of only Google group increases by reading this text explanation. But this is at the cost of increasing the time used on task 3 compared with using Google + *DeveloperBot* group as shown in Fig. 4.24. These prove that *DeveloperBot*, as a search engine assistant, can improve the answer confidence by the provided explanation.

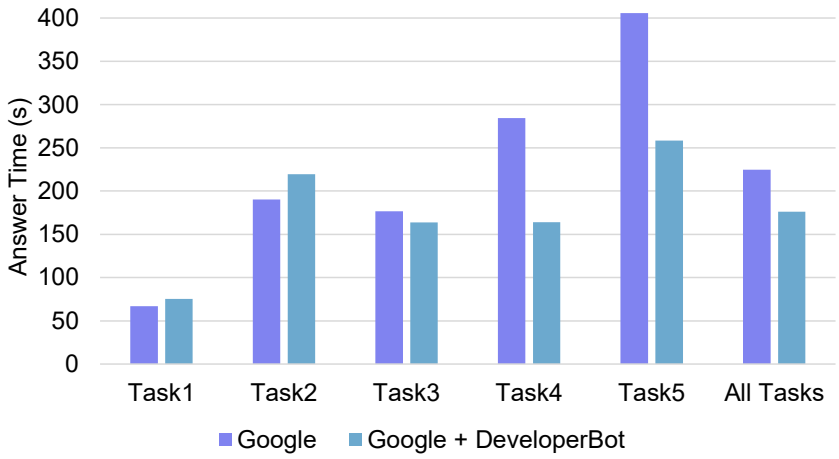


Figure 4.24: Answers Search Time for Each Task

Fig.4.24 is the result of answers search time (the time from the participants start to read the task to they finish the task.) of the user study. It shows that, in addition to task 3 which can find a webpage containing direct answers and corresponding text explanations, the answers search time of the Google group increases significantly with the complexity of the tasks increasing. For the Google group, the average answer search time is 405s. But with the assistant of *DeveloperBot*, the average answer search time of the Google + *DeveloperBot* group in task 5 is 258s.

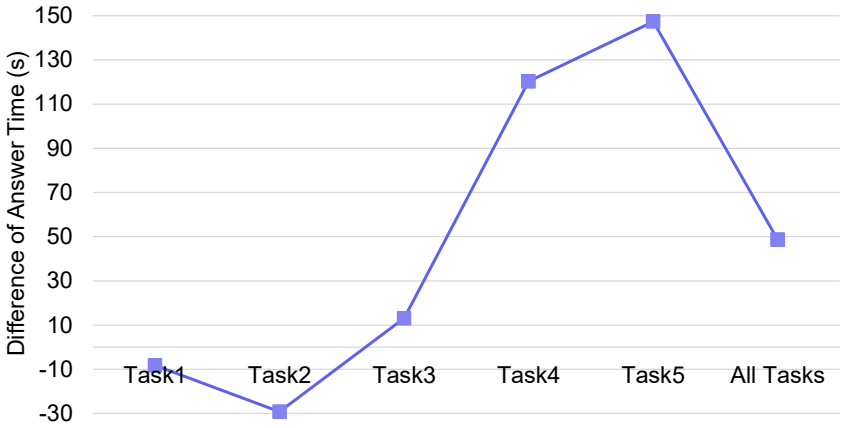


Figure 4.25: The Trend of the Difference of Answer Search Time with Increasing Task Complexity

Fig.4.25 shows a significant trend that with the assist of *DeveloperBot*, the participants can save more time in higher complexity answer search tasks. The group using Google +

DeveloperBot can save 48.5s on average for each answer search task. For task 5, the time saving even can reaches 147s. These show that using *DeveloperBot* as search engine assistant can significantly reduce the participants' answer search time especially for the complex answer search tasks which take a long time to solve with only Google. The open discussion also reflects consistent results for the above conclusion. The special performance of Google group at task 3 also illustrates that direct answers and the explanation(even text) can significantly reduce the search time and improve the answer confidence at the same time.

4.4.4.4 Result: comments to explanations and behaviour changes (RQ4)

We will evaluate the explanation and answer the RQ4 from three ways: first is the quantitative scoring (0-5) of participants in the questionnaires, second is the information obtained from open discussions, and the third is our observation of screen recording.

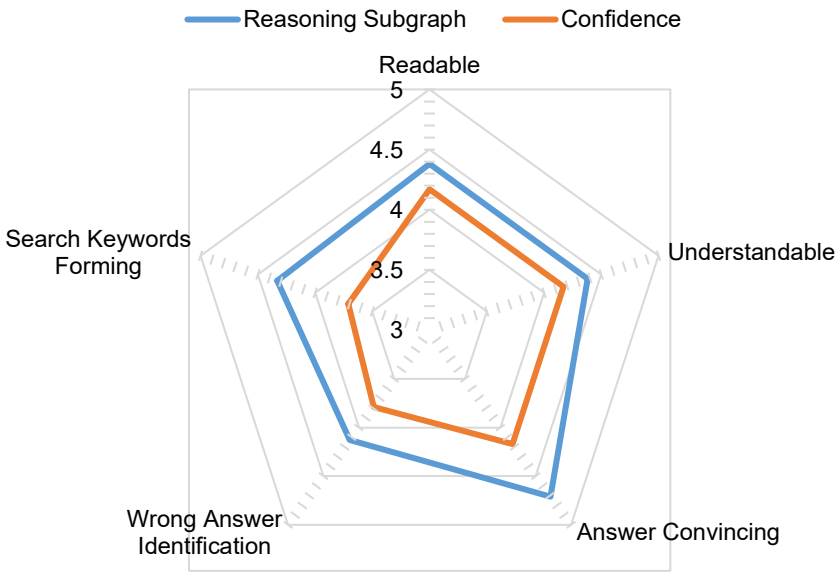


Figure 4.26: Quantitative Scoring of the Explanations

The results of the quantitative scoring of explanation are shown in radar graph Fig. 4.26, and the detailed description is provided in Tab. 4.8. Two schemes are used in *DeveloperBot* as the explanation of direct answers. One is the reasoning subgraph which indicates the reasoning paths from query to direct answers, another is the confidence of current direct answer. Some criteria define the quality of explanation and functions an explanation needs to fulfill [35]. Referring to

these criteria and functions, five indexes are adopted in this experiment to evaluate to quality of the explanation. They are Readable (R), Understandable (Ud), Answer Convincing (AC), Wrong Answer Identification (WAI) and Search Keywords Forming (SKF) Where Readable and Understandable mean the explanation is legible, easy to read and understand. Answer Convincing means this explanation can make the direct answer more convincing. Wrong Answer Identification is the index that evaluates if the explanation can help identify the wrong answer. Search Keywords Forming shows the capacity of the explanation to help form better search keyword.

Table 4.8: Detailed Description of the Quantitative Scoring of the Explanations

No.	Explanation	R	Ud	AC	WAI	SKF
1	Reasoning Subgraph	4.38	4.38	4.71	4.13	4.33
2	Confidence	4.17	4.17	4.17	3.79	3.71

As shown in Fig. 4.26 and Tab. 4.8, the overall scores of reasoning subgraph reach 4.13-4.71. Compared with the other four indexes, the Answer Convincing achieves the highest score 4.71. The ranking of indexes of Readable and Understandable are second and third, with average values of 4.38. Even the Wrong Answer Identification achieves the lowest score, 4.13 is still a high score. The results demonstrate that the reasoning subgraph is a very reasonable and high-quality explanation of direct answer and meet the desired design goals.

In addition, the Fig. 4.26 and Tab. 4.8 also shows the overall scores of using confidence as a explanation. Compared with the reasoning subgraph, the overall scores of confidence are relatively lower than reasoning subgraph (3.71-4.17). The indexes Readable, Understandable and Answer Convincing achieve highest score 4.17. The scores of Wrong Answer Identification and Search Keywords Forming is 3.79 and 3.71, respectively. Even the overall scores of confidence are lower that reasoning subgraph, especially the WAI and SKF. But the result of the indexes R, Ud and AC still can show that it is a good explanation for the direct answer.

The explanations provided by *DeveloperBot* show the reasoning process and the confidence of the answers to the participants, which significantly improve their confidence in the answers provided by *DeveloperBot*. Sometimes, this explanation can even help the participants rule out incorrect answers caused by inaccurate information needs description.

4.5 Summary

The developers have a need for complex information search that provide specific direct answers. The current search engines are weak at the reasoning capacity for complex closed-end questions. In order to solve this issue, in this chapter, a tool called *DeveloperBot* including a series of implementations of the modules of *XBot* framework was presented. *DeveloperBot* can be used as a search engine assistant to assist the answer search of the complex closed-end questions. It is customized by loading the knowledge graph of the software engineering domain into the knowledge base of the *XBot* framework.

Quantitative evaluation and a user study were conducted to evaluate the performance of *DeveloperBot*. The result of quantitative evaluation shows that, with topological structure, predicate similarity and other novel features of the subgraph, the Bayesian decision theory and DNN algorithm can extract the correct answers from the candidate answers with high accuracy and low confidences MSE. The participants of the user study express that *DeveloperBot* has the capacity to model and reason complex problems. They think that *DeveloperBot* is a good supplement to the search engine. The experimental data also shows similar results: compared with just using Google, with the assist of *DeveloperBot*, users can find answers faster and with more accuracy. In addition, using the reasoning subgraph and answer confidence as the explanation of the direct answers can significantly improve the developers' trust and adoption to the answers. These explanations also assist the developers to understand the answers more deeply, improve the answer accuracy and form better search keywords. Furthermore, the more complex the question is, the more effective the *DeveloperBot* can achieve.

Chapter 5

HDSKG: Mining Domain-specific Knowledge Graph From Unstructured Text Resources (*BotLearning*)

In the previous chapters, a brain-inspired cognitive framework of Q&A process named *XBot* and a tool called *DeveloperBot* including a series of implementations of the modules of *XBot* framework were presented. In this chapter[1], an implementation of the *BotLearning* algorithms of *XBot* called *HDSKG* will be presented.

The knowledge graph is useful for many different domains like search result ranking, recommendation, exploratory search, etc. It integrates structural information of concepts across multiple information sources, and links these concepts together. The extraction of domain-specific relation triples (subject, verb phrase, object) is an important technique for domain-specific knowledge graph construction. In this chapter, an automatic method named *HDSKG* is proposed to discover domain-specific concepts and their relation triples from the content of webpages. We incorporate a dependency parser with a rule-based method to chunk the relations triples candidates, then we extract advanced features of these candidate relation triples to estimate the domain relevance by a machine learning algorithm. For the evaluation of our method, we apply *HDSKG* to Stack Overflow (a Q&A website about computer programming). As a result, a knowledge graph of the software engineering domain is constructed. The experimental results show that both the precision and recall of *HDSKG* achieve new state-of-the-art. The performance is particularly efficient in the case of complex sentences. Furthermore, with the self-training

[1]The work in this chapter has been published in [2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER). (pp. 56-67), IEEE]

technique we used in the classifier, *HDSKG* can be applied to other domain easily with less training data.

The rest of this chapter is organized as follows: In the Section 5.1, we introduce the background and motivation. In the Section 5.2, we first introduce the architecture of *HDSKG* in Section 5.2.1. Next, we show the data pre-processing in the Section 5.2.2 and Section 5.2.3. Then, we introduce how to extract candidate relation triples by *HDSKG Chunking* module in the Section 5.2.4. The domain estimation algorithm called *HDSKG Domain Relevance Estimation* is shown in the Section 5.2.5. We evaluate the performance of *HDSKG* in the Section 5.3.

5.1 Background and Motivation

Recently, the trends and development of information retrieval and mining have transferred from the document-centric to the entity-centric [136]. So knowledge graph which integrates the structural information of concepts across multiple information sources, and links these concepts together [160] is useful for many different domains, such as search result ranking, recommendation, exploratory search, etc [80, 92, 88, 94]. The webpages such as Wikipedia, Quora are potential repositories for building up knowledge graph. Thus, automatic knowledge graph construction techniques for the content of webpages become a wide research topic. [92, 161, 162].

Open information extraction refers to the extraction of relation triples without specifying the fixed relation schema, typically the relation is a triples with (subject, verb phrase, object) structure [163, 164] (Subjects and objects are collectively called concepts). The relation triples are one of the important components of the knowledge graph [165]. There are many researches on open information extraction including NELL [47], OpenIE [48], and Google [49]. These existing open information extraction methods fall short of the precision and completeness of the textual knowledge extraction. And can not extract the knowledge graph of the specified domain from the text materials as well. As a result, the scale of the extracted knowledge graph is very large and contains a lot of redundant information.

Knowledge Graph is also popularly used in software engineering domain [166, 167, 94, 95] to solve various problems such as software requirements analysis and design [168], coding support, documentation [94], maintenance and testing [169]. But there are few studies focus on

structural knowledge-base construction in software engineering domain. Large scale knowledge-base construction (KBC) in other domains were studied intensely over the last decade [170, 171, 47, 172, 173, 174, 175]. Such as relational database which allows users to semantically query relationships and properties of natural language resources, including links to other related datasets [161]. It can be used to straightforward search engine algorithms or other search operations and improve the accuracy of information retrieval. Even though some research works are focusing on ontology construction in software engineering domain [176, 122]. They only use the domain key words as the concepts and explore the taxonomic relations of the key words. However, there are still many domain concepts indicating richer semantic and more useful relations between the concepts needed to be mined.

Stack Overflow is the most popular Q&A website about computer programming [177, 122], every question requires the asker to give 1-5 tags. The tag wiki is the content used to describe the definition and some related resource of every tag in Stack Overflow. Such a knowledge repository contains a huge number of sentences with affluent concepts and concepts relations descriptions of software engineer domain. These relations are essential for the construction of software engineer domain knowledge graph. There are a substantial amount of works that explore the knowledge of *tag wiki* [136, 178], but none of them focuses on the extraction of the relation triples.

In this chapter, an automatic method named *HDSKG* (Harvesting domain-specific Knowledge Graph) is proposed to discover domain-specific concepts and their relation triples from the domain-specific webpages. We incorporate a dependency parser with a rule-based method to chunk the relations triples candidates, then it extracts novel features of those triples to estimate domain relevance by self-training SVM (Support Vector Machine) classifier.

For the evaluation, a case study is conducted to extract the knowledge graph from Stack Overflow. Our experimental results show that both the precision and recall of *HDSKG* (78% and 70%, respectively) is much higher than the openIE (11% and 60%, respectively) – the state-of-the-art tool for relation triples extraction. Furthermore, self-training SVM classifier makes *HDSKG* easy to be applied to other domains with less training data.

This chapter makes following three major contributions:

- A novel algorithm called *HDSKG* is proposed which is an implementation of the *BotLearning* algorithms of *XBot*. It extracts the dependencies of the NP (Noun Phrases) and

VP (Verb Phrases) from sentences to generate relation triples. It explicitly incorporates the advantages of both dependency parser and rule-based chunking and improves the efficiency of relation triples extraction.

- The features are further extracted from the relation triples, then *HDSKG* leverages a self-training SVM classifier and domain lexicon to estimate the domain relevance of the relation triples with a small number of training data. This remarkably reduces the number of redundant relation triples and improves the extracted precision.

- *HDSKG* is applied to Stack Overflow tag wiki, and harvests a knowledge graph of software engineering domain with 44800 concepts, 9660 unique verb phrase and 35279 relation triples, and it is available online [1].

5.2 The Approach

In this section, we describe the approach of *HDSKG* to show how *HDSKG* harvest domain-specific knowledge graph from content of webpages. In particular, we present how *HDSKG* works on natural language texts sources to extract relation triples of domain-specific knowledge graph.

5.2.1 Architecture of HDSKG

Fig. 5.1 is the framework of *HDSKG*. The input of *HDSKG* is the natural language text materials from content of webpages (e.g., healthcare webpages, computer programming webpages, etc.). The output of *HDSKG* is the knowledge graph of target domain .

The *HDSKG* contains two main parts as shown in Fig. 5.1 named *HDSKG Chunking* and *HDSKG Domain Relevance Estimation*. *HDSKG Chunking* incorporates the rule-based method with dependency parser to chunk candidate relation triples. This part comprises five main steps, namely pre-process text, split sentences, add NLP (Natural Language Processing) markup, chunk NP and VP, and chunk relation triples. The second part is used to estimate domain relevance of the candidate relation triples, which comprises two main components, namely extract features

[1] http://neo4j.tuntunkun.org/xuejiao/

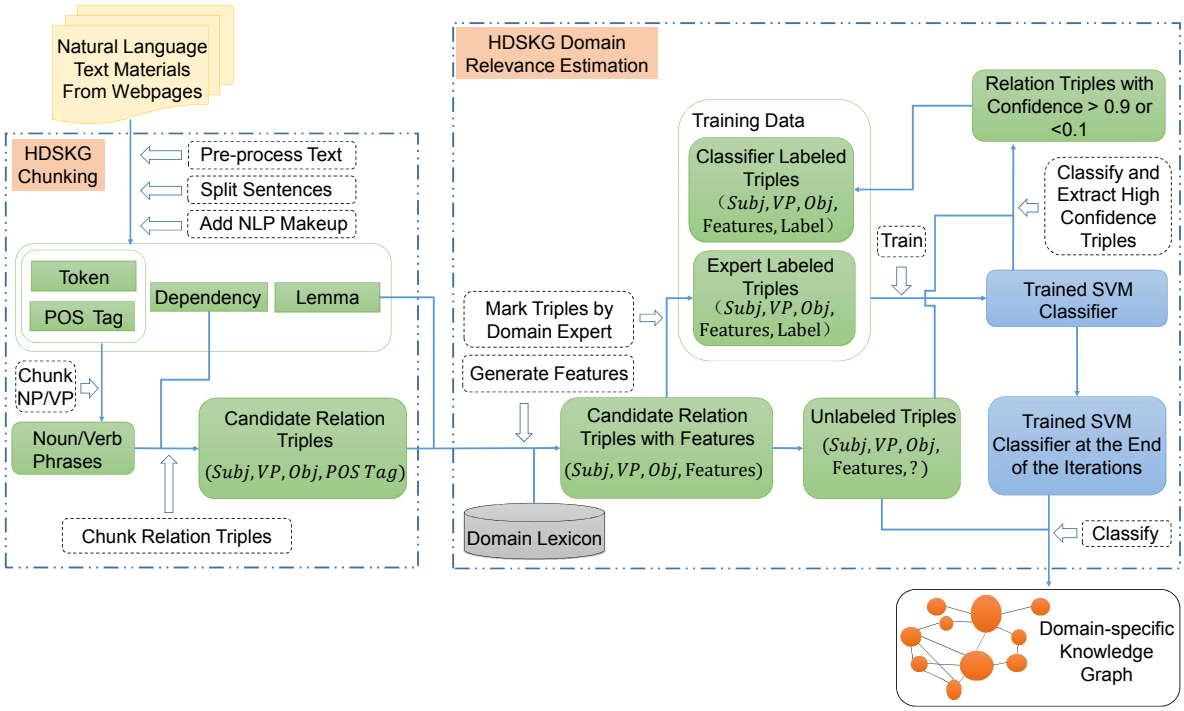


Figure 5.1: The Framework of *HDSKG*

of candidate relation triples, estimate domain relevance of each relation triples by self-training SVM classifier [1].

5.2.2 Pre-process Text

In this step, we extract all the text content from the html files of webpages and filter some useless information which do not contain concepts and relations like hyperlink lists, code snippets, etc. Then we split the text content into sentences.

After that, we enrich the sentences by two steps:

- Recording the source information for every sentence, e.g., the title of the source webpage from which the sentence are extracted, and the sequence of sentence in the source webpage.
- Making the sentences readable without the context: replacing the pronouns at beginning of the sentences (e.g., it, she, he) with the title of source webpage [163].

[1] Actually, the self-training model can choose any classifier. The SVM is chosen just because it performs best in the experiment.

Take the sentence *"It is written in C++, and is ultimately derived from the Borland InterBase 6.0 source code."* for example. It is the third sentence in the tag wiki of *Firebird*, so we record the title of the source webpage – *Firebird* and the sentence sequence – three. Then we replace the *"it"* to *"Firebird"*, so the sentence become *"Firebird-3, Firebird is written in C++, and is ultimately derived from the Borland InterBase 6.0 source code."*

5.2.3 Add NLP Markup

NLP markup is widely used for natural language pre-processing and semantic analysis [137, 138, 139]. The NLP markup component in *HDSKG* system adopts the tool named *coreNLP* of Stanford for Tokenization, POS (Part of Speech) Tagging, Dependency Parsing and Lemmatization [132, 133, 134, 135]. Tokenization is used to break the text into words, phrases, or symbols.

The POS is the category of words (or, more generally, of lexical items) which has similar grammatical properties. e.g., NNP:Proper noun, singular, VBZ:Verb, third person singular present, RB:Adverb. Fig. 5.2 shows the POS tagging result of definition sentence of *Firebird* generated by *coreNLP*.

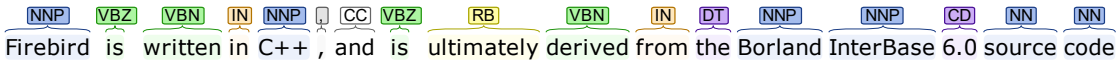


Figure 5.2: POS tagging results of the definition sentence of the Firebird

The goal of lemmatization is to reduce inflectional forms and derivational related forms of a word to a common base form.

5.2.4 Chunk Candidate Relations Triples

There are two times chunking to get the candidate relation triples. Firstly, *HDSKG Chunking* chunks the tokens in the sentence to NP and VP according to the POS, then utilizes the dependency of the tokens to chunk the VP and NP to relation triples.

5.2.4.1 Chunk NP and VP by Rule-Based Chunking

In this step, we adapt a fulltext parsing techniques called "rule-based chunking" to extract NP and VP [142, 143].

Table 5.1: Regular Expression of Different Chunks

Name	Regular Expression
VVP	(MD)*(VB.*)+(JJ)*(RB)*(JJ)*(VB.*)?(DT)?(TO*)+(VB)+ (MD)*(VB.*)+(JJ)*(RB)*(JJ)*(VB.*)?(DT)?(IN*)+(VBG)+
VP	(MD)*(VB.*)+(CD)*(JJ)*(RB)*(JJ)*(VB.*)?(DT)?(IN*|TO*)+ (MD)*(VB.*)+(JJ)*(RB)*(JJ)*(VB.*)?(DT)?(IN*|TO*)+ (MD)*(VB.*)+(JJ)*(RB)*(JJ)*(VB.*)+ (MD)*(VB.*)+
NP	(CD)*(DT)?(CD)*(JJ)*(CD)*(VBD|VBG)*(NN.*)*- (POS)*(CD)*(VBD|VBG)*(NN.*)*- (VBD|VBG)*(NN.*)*(POS)*(CD)*(NN.*)+

In our system, the NP and VP are identified by the regular expressions as shown in Tab. 5.1. Where (MD) is modal; (VB.) stands for different categories of verb such as VB - verb base form, VBG - verb gerund or present participle, VBN - verb past participle, VBP - verb non-3rd person singular present, VBZ - verb 3rd person singular present. (NN.*) stands for different categories of noun such as NN - singular or mass noun, NNS - plural noun, NNP - singular proper noun, NNPS - plural proper noun. (JJ) represents an adjective; (RB) is adverb; (DT) presents an article; and (IN*) means any preposition or subordinating conjunction. The "VVP" is the VP with open clausal complement.

In the regular expressions, "?" stands for whether or not there is such a determinant; "*" means zero or more determinant; "+" means must have such a determinant; "-" means continue to next row.

Using the rule-based chunking, we chunk the tokens of sentence to VP and NP. For example, the sentence *"PyTables is built on top of the HDF5 library, using the Python language and the NumPy package."*, we chunk VP *"is built on"* and *"using"*; NP *"PyTables"*, *"HDF5 library"*, *"Python language"*, and *"NumPy package"* as shown in Fig. 5.3.

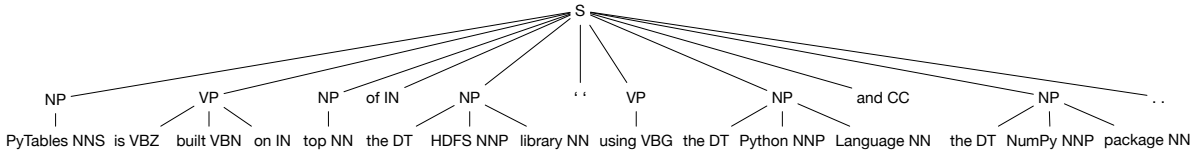


Figure 5.3: Result of NP and VP Chunking

5.2.4.2 Dependency Parsing

Dependency Parsing extracts the grammatical structure of the sentence, then derive relationships between "head" words (also known as Governor) and words which modify the heads (also known as dependent). Fig. 5.4 shows a directed graph representation of the dependencies for the sentence: *"PyTables is built on top of the HDF5 library, using the Python language and the NumPy package."*. The nodes of Fig. 5.4 are the words of the sentence, and the edges in Fig. 5.4 are grammatical relations.

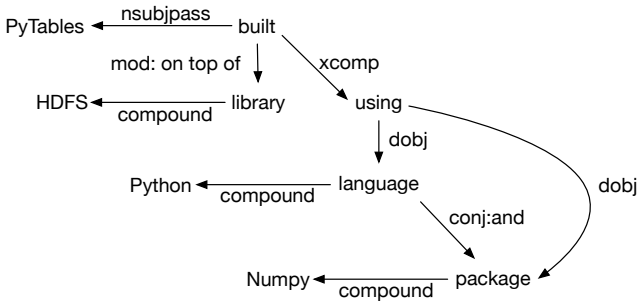


Figure 5.4: Graphical representation of the Dependencies for the sentence: *"PyTables is built on top of the HDF5 library, using the Python language and the NumPy package."*

Tab. 5.2 presents some important dependencies of sentence *"PyTables is built on top of the HDF5 library, using the Python language and the NumPy package."* used in our system.

Table 5.2: Dependencies Description for sentence: *"PyTables is built on top of the HDF5 library, using the Python language and the NumPy package."*

Dependency	Dependent	Governor	Semantic relationship between the words depicted by denpendency
nsubjpass	PyTables-1	built-3	"PyTables-1" is passive nominal subject of "built-3"
auxpass	is-2	built-3	"is-2" is passive auxiliary of built-3
det	the-7	library-9	"the-7" is determiner of "library-9"
nmod	library-9	built-3	"built-3" is nominal modifiers of "library-9"
xcomp	using-11	built-3	"using-11" is open clausal complement of "built-3"
dobj	language-14 package-18	using-11	"language-14" and "package-18" is direct object of "using-11"

As Tab. 5.2 shows, "nsubjpass" is a NP and the syntactic subject of a passive clause [140]. For our illustrative sentence, nsubjpass(PyTables-1, built-3) means that "PyTables-1" is the syntactic subject of the verb "built-3".

The "nmod" used between two content words, as the preposition of one content word is now viewed as a case depending on its complement. In general, the "nmod" indicates some further adjunct relation specified by the case [141]. The nmod (library-9, built-3) means that "built-3" is nominal modifiers of "library-9". The "library-9" expresses further "on top of" relation to "built-3". Notice that some times there will appear a preposition after "nmod" like "nmod:at". This "at" means the complement use the preposition "at" to modify the Dependent.

The "xcomp" means open clausal complement of a verb or an adjective. These complements are always non-finite rather than adjuncts/modifiers. It is a predicative or clausal complement without its own subject. So the "xcomp" always expresses a new meaning or new relation. The xcomp (using-11, built-3) means that "using-11" is open clausal complement of "built-3".

The "dobj" is the direct object of a VP, and is also a NP which is the accusative object of the verb.

5.2.4.3 Chunk Relation Triples

After the VP and NP chunking, we can chunk semantic relation like (concept, relation verb phrase, concept). Traditional rule-based chunking [179, 180] detects the chunk with morphological structure like (NP, VP, NP). This method not only misses many information, but also adds noise to the relation triples. For example, from sentence *"Firebird is written in C++, and is ultimately derived from the Borland InterBase 6.0 source code."*, traditional rule-based chunking can only extract relation triples (Firebird; is_written_in; C++). From sentence *Eclipse on your system can be used as a Java editor and a C++ editor*, relation triples (system; can_be_used_as; Java_editor) are extracted by traditional rule-based chunking using the morphological structure (NP, VP, NP). However, it is semantically wrong.

To tackle the above problems, we propose a method which incorporates a dependency parser with a rule-based chunking. By analyzing the dependency generated by the dependency parse technique, the real subject of the verb in a complex sentence is determined. We assuming having a sentence S. Firstly we use the rule-based chunking to get all the VP and NP as below:

$$S = (NP_1, NP_2, ..., NP_i, ..., NP_m, VP_1, VP_2, ..., VP_j, ..., VP_n), \tag{5.1}$$

where NP_i is the ith NP in S, VP_j is the jth VP in S, there are m NP and n VP in S.

$$NP_j = (n_{j1}, n_{j2}, ..., n_{jr}, ..., n_{jq}), \tag{5.2}$$

$$VP_i = (v_{i1}, v_{i2}, ..., v_{ik}, ..., v_{ip}), \tag{5.3}$$

where n_{jr} is the rth term of NP_j, v_{ik} is the kth term of VP_i, we assume that NP_j contains q terms and VP_i contains p terms.

Secondly we use dependency parsing to get the dependency of all terms v and n. Due to the dynamic of natural language, there are too many different expressions for the same meaning. After we analyzing the dataset deeply, we propose six scenarios for *HDSKG Chunking* to chunk the candidate relation triples from the sentences.

Scenario 1: If we extract the nsubjpass (n_{12}, v_{13}), dobj(v_{13}, n_{22}) from S by dependency parsing, we can chunk (NP_1, VP_1, NP_2) as a relation triples. For example, in the sentence *"OpenRPT provides WYSIWYG editor"*, the "openRPT" is NP_1 and n_{11} , the "WYSIWYG editor" is NP_2 and "editor" is n_{22}, the "provides" is VP_1 and v_{11}. From dependency parsing we get nsubjpass (openRPT-1, provides-2) and dobj (provides-2, editor-4), so we can chunk relation triples (OpenRPT; provides; WYSIWYG_editor).

Scenario 2: If we extract the dependency like nsubjpass (n_{11}, v_{12}), nmod (v_{12}, n_{23}) in sentence *"HSQLDB is supported by many Java frameworks."*, where n_{11} is "HSQLDB", v_{12} is "supported", n_{23} is "frameworks". We should chunk relation triples (HSQLDB; is supported by; many_Java_frameworks). Here we assume adjective "many" can not be ignored like DT (article), because "many Java frameworks" is not equal to "all Java frameworks".

Scenario 3: In some cases, for example "webkit is developed by Intel at the Intel Technology Center.", we get dependency like nsubjpass (webkit-1, developed-3), nmod:agent (developed-3, Intel-5) and nmod: at (developed-3, Center-10). Here we chunk two relation triples (webkit; is_developed_by; Intel) and (webkit; is_developed_at; Intel_Technology_Center). Notice that with the preposition of dependency "nmod:at", the preposition of VP_1 "is_developed_by" can be changed to "at" while chunking the second relation triples.

Scenario 4: If there is an "and" between two NPs, we will chunk 2 relation triples with different subjects or objects. For example, from sentence "flashcanvas renders shapes and images.", we will chunk relation triples as:

- (flashcanvas; renders; shapes)
- (flashcanvas; renders; images)

Scenario 5: If there is a dependency "xcomp" extracted from S, we will give an additional dependency "nsubjpass" to the dependencies of the sentence. For example, the sentence *"PyTables is built on top of the HDF5 library, using the Python language and the NumPy package."* as shown in Section 5.2.4.2, we extract two dependencies – xcomp(using-11, built-3) and nsubjpass(PyTables-1, built-3) from this sentence, we will add an additional dependency nsubjpass(PyTables-1, using-11) to this sentence. It because that as a open clausal complement of a verb or an adjective, the "xcomp" always expresses a new meaning or new relation. Thereby we can chunk relation triples as:

- (PyTables; is_built_on_top_of; HDF5_library)
- (PyTables; using; Python_language)
- (PyTables; using; NumPy_package)

Scenario 6: If S contains VVP as mentioned in Section 5.2.4.1, we do not chunk two relation triples like **Scenarios 5** even there is a "xcomp" extracted from S. This is because the two verbs of VVP share the same object. Instead, we chunk only one relation triples and leave the two verbs together in VVP as the relation verb of the relation triples. For example "Joone is used to build neural networks.", we will chunk relation triples as:

- (Joone; is_used_to_build; neural_networks)

By the above method, from "Firebird is written in C++, and is ultimately derived from the Borland InterBase 6.0 source code.", we can get dependency nsubjpass(Firebird-1, derived-10), so we will chunk relation triples as:

- (Firebird; is_ultimately_derived_from;

  Borland_InterBase_6.0_source_code)
- (Firebird; is_written_in; C++)

From sentence *"Eclipse on your system can be used as a Java editor."* we can get nsubjpass(Eclipse-1, used-7). Thus we can chunk relation triples as:

- (Eclipse; can_be_used_as; Java_editor)
- (Eclipse; can_be_used_as; C++_editor)

5.2.5 Domain Relevance Estimation of Candidate Relations Triples

From *HDSKG Chunking* we present in Section 5.2.4, we harvest many candidate relation triples (subject, verb phrase, object). But some of the relation triples are not related to our target domain. For example, the relation triples (Values; may be enclosed in; quotes), this is a correct chunking but meaningless for the software engineering domain. In this section, we use a machine learning method with advanced features to estimate the domain relevance of the candidate relation triples generated by *HDSKG Chunking*.

5.2.5.1 Feature Engineering

There are many properties beyond simple term statistics which can be extracted from the candidate relation triples to estimate their domain relevance. We use the features list in Tab. 5.3 to represent each candidate relation triples.

For the complex features we provide a brief description below.

Text Features: This feature type regards the basic text properties of the cr (candidate relation triples) at the term level. The POS fraction features capture the distribution of POS tags in the cr and uses following definition:

- noun : the tokens which POS tags are NN, NNS, NNP, or NNPS.
- verb : the tokens which POS tags are VB, VBD, VBG, VBN, VBP, or VBZ.
- adj : the tokens which POS tags are JJ, JJR, or JJS.

Corpus Features: These features present the statistic relation between current cr and whole cr corpus. If the cr is extracted from source webpages many times, this cr appear to be relevant to target domain with higher potential. The $conf_{subj}$ denotes the percentage of the occurrence of object when subject occurs, the $conf_{obj}$ denotes the percentage of the occurrence of subject when object occurs. And the support indicates the percentage that subject and object occur simultaneously in the corpus. These association rule features present relation between the subject and object. If these two concepts can be found frequently from stm (source text material) simultaneously, the relation triples of these two concepts are high potential relevant to target domain.

Table 5.3: The Features for Candidate Triples Classification

#	Name	Gloss
		Text features
1	#terms_cr	Number of,terms of cr (candidate relation triples)
2-5	POS fractions_subj	Fraction of verbs, nouns, adjectives, others,of subject
6-9	POS fractions_obj	Fraction of verbs, nouns, adjectives, others,of object
10-13	POS fractions_vp	Fraction of verbs, nouns, adjectives, others,of verb phrase
14-17	POS fractions_cr	Fraction of verbs, nouns, adjectives, others,of cr
		Corpus features
18	#mention_subj	Number of subject mentioned in whole cr corpus
19	#mention_obj	Number of object mentioned in whole cr,corpus
20	#mention_vp	Number of verb phrase mentioned in whole cr corpus
21	#mention_cr	Number of cr mentioned in whole cr corpus
22	suport_subj_obj	The proportion that subject and object occur simultaneously
23	conf_subj	The proportion of the occurrence of object
24	conf_obj	The proportion of the occurrence of subject
		Concept features
25	subj_tfidf,	TF-IDF of subject in tag wiki,in whole text materials
26	obj_tfidf	TF-IDF of object in tag wiki,in whole text materials
27	sum_tfidf	Sum of TF-IDF of subject and object in whole text materials
28	average_tfidf	Average TF-IDF of subject,and object in whole text materials
29	%domain_keyword_subj	The proportion,of number of domain keywords in subject
30	%domain_keyword_obj	The proportion of number of domain keywords in object
31	%domain_keyword_subj_obj	The proportion of domain keywords in subject and object
		Source features
32	ss_position	Position of ss (source sentence) in stm (source text material)
33	cr_position	Position of cr in ss
34	From_subj_obj_stm	If this cr extract from stm
35	#terms_ss	Number of terms in ss which cr extracted from
36	start_index_cr	Start index of the first term of cr in ss
37	end_index_cr	End index of the first term of cr in ss

Concept Features: These features regard the quality of the concepts. TF-IDF is a numerical statistic approach which can determine the words relevance in a corpus of documents [181, 182]. TF (Term Frequency) means the raw frequency of a term in a document. IDF (Inverse document frequency) is the logarithmically scaled inverse fraction of the documents that contain the word, obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient. Here is the formula for unnormalized weight of $term_i$ in $document_j$ in a corpus of D (documents):

$$TF - IDF_{weight_{i,j}} = frequency_{i,j} * log_2(D/document_{freq_i}), \tag{5.4}$$

Thus the features $subj\_tfidf$, obj_tfidf, sum_tfidf and $average\_tfidf$ can indicate the domain relevance of subjects and objects. If most of the terms in a subject are very general like "values", "process", etc., the value of the $subj\_tfidf$ will be very low. Otherwise if the subject contains unique and distinctive terms like "c++ library", "MySQL relational database", the $subj\_tfidf$ value will be high.

We also prepare the domain lexicon which contains the specialized vocabularies of the target domain for estimating the domain relevance of the candidate relation triples. So the candidate relation triples which contain more specialized vocabularies of the target domain show higher domain relevance [183].

Source Features: Here, we refer to features of the cr depended on the stm (source text material) and ss (source sentence) [184]. For the $crposition$ and $ss_{position}$, these two features mean that the cr extracted from the front part of the stm and the front part of the ss is high relevance to target domain. The position refer to the sequence of a cr in the stm or the ss. On the other hand, if a cr is extracted from the webpage in which the topic contains the terms of concepts, it shows high domain relevance. For the long sentence, $cr\_position$, $start\_index_{cr}$ and end_index_{cr} indicate that the cr extracted from the front part of a long ss shows high domain relevance.

5.2.5.2 Label Data

We design the estimation of domain relevance as a binary-class classification problem to solve. The supervised learning of classification requires the labeled data for training. We manually mark candidate relation triples by domain experts as training data to boost the learning process. Tab. 5.4 shows a example about how to label the positive and negative relation triples.

5.2.5.3 Semi-Supervised SVM Classifier Learning

The workload while using a system to a new domain determines the applicability and the portability of a system. However, labeling new training data is very labor consuming. To reduce the labor force and improve the prediction accuracy, we select the best single classifer – SVM classifer with a self learning structure to do the classification [122]. In the first iteration, we

Table 5.4: Example of Relation Triples Label

Candidate Relation Triples	Label
(Java; supports; features)	False
(row; is_represented_as; list)	False
(Values; may_be_enclosed_in; quotes)	False
(Java; was_originally_developed_by; James_Gosling)	True
(Java; was_originally_developed_at; Sun_Microsystems)	True
(Eclipse; is; open-source_ide_platform)	True

use the expert labeled relation triples as the training data and get a trained SVM classifier. Then, we input the unlabel relation triples to the trained SVM classifier and get the labels (or classes) of them. The classifier not only classify the relation triples, but also provide a confidence level to every classified relation triples to show the probability of current classification. From the second iteration, self-training enrich training data with the labeled relation triples which $confidence > 0.9$ or $confidence < 0.1$ to train a new classifier for the next iteration. The iteration will terminate if the difference of accuracy lower than a threshold or achieved the preseted iteration time. Then we use the trained SVM classifier at the end of the iterations to classify all the unlabeled relation triples, and use the relation triples in the positive classes to construct the knowledge graph of target domain.

5.3 Experiment

In this section, a case study is proposed to illustrate the *HDSKG* system process for acquiring the domain-specific relation triples and constructing a knowledge graph. Tag wiki of Stack Overflow[1] is used to extract the knowledge graph and some tag wiki documents are chose to evaluate our system.

[1]https://stackoverflow.com/tags

5.3.1 Experiment Setup

5.3.1.1 Data Acquisition

We download the data dump of Stack Overflow until Mar, 2015 from Stack Overflow official data dump[1].

Source Text Materials Acquisition: When asking questions on Stack Overflow, the users will be asked to put 1-5 tags as a short description of this question, so that the Stack Overflow can easy to classify the questions. Tag is a technical term (e.g., windows, graph database, javascript, etc.) involving a wide range of content from the operating system, programming languages, libraries, to specific APIs [136]. The tag wiki is a general introduction and definition of a tag. A sample tag wiki of tag "Firebird" [2] is shown in Fig. 5.5.

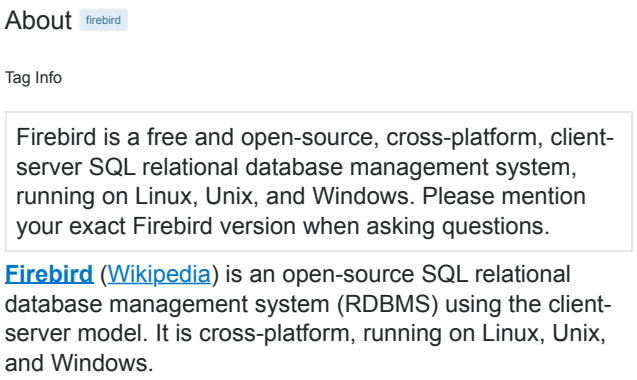


Figure 5.5: Tag Wiki of Tag "firebird"

Stack Overflow is strict to create new tag[3] and edit the tag wiki[4]. The community also pays a lot of effort to maintain and organize the tag wikis. As a result, the tag wikis of Stack Overflow tags have the advantages of accurate tag introduction, comprehensive technical coverage and standard grammatical structure for constructing the knowledge graph of software engineering domain.

In this chapter, the tag wiki of Stack Overflow provided in a html format is used as source text materials to construct the knowledge graph. Some content which contain few relation triples

[1] https://archive.org/details/stackexchange
[2] http://stackoverflow.com/tags/firebird/info
[3] http://stackoverflow.com/help/privileges/create-tags
[4] https://stackoverflow.com/help/privileges/approve-tag-wiki-edits

such as “frequent question”, “Useful links”, etc. are deleted. As a result, we acquire 20534 documents and split 97454 sentences from the documents.

Domain Lexicon: We use the tags of Stack Overflow as the domain lexicon. Finally, 27620 tags is acquired, these tags cover the popular programming languages(e.g., Java, Python, PHP), frameworks(e.g., GWTP, Pallet, AnyEvent), libraries(e.g., OmniFaces, PHPPowerPoint), tools(e.g., opengl, SubGit, MvcSiteMapProvider) and some frequently used terminology(e.g., cookies, ip) of software engineering domain.

Ground Truth of the Relation Triples: From the tag wiki of Stack Overflow, we choose 47 tag wikis by random as our evaluation materials. Four experts are asked to extract the relation triples from the webpages of the 47 tag wikis. The four experts have the background in software engineering and three of them have ever published knowledge graph or ontology construction related papers. To ensure the precision, we give the hyperlinks to the experts rather than the pre-processed documents. Only the relation triples which extract by at least 2 experts are involved in our ground truth. Finally we get 96 relation triples from the webpages of tag wiki, but we ignore 5 of them due to their source sentences from webpages are different from the source sentences from data dump. We use the rest 91 relation triples as the ground truth to evaluate the performance of *HDSKG*.

5.3.1.2 Involved Tools

The *Stanford CoreNLP* is used to do the NLP make up and dependency parse [154]. The tf-idf library of *Gensim* is used to compute tf-idf [155]. The *nltk* is used to do the VP, NP chunking [156]. *neo4j* is used to store the finally relation triples for supporting the knowledge graph searching and result visualization [157].

5.3.2 Comparison Method *openIE*

We leverage the popular open information extraction tool - open IE of Stanford as the comparison method [164, 48] as it is similar to *HDSKG*. On the other hand, openIE was the most state of art tool for open information extraction and provided opensource code in 2016. Our research work was almost the first work proposed the domain relevance estimation function for open information knowledge graph construction. Therefore, we chose to compare with OpenIE when we conducted this research work.

But there are also some differences between our tool and openIE. First is the method to extract relation triples from stm. openIE only focus on the extraction of relation triples, but ignore the subject and object should be entities. For example, from sentence "Born in a small town, she took the midnight train going anywhere.", openIE will extract relation triples (she; took; midnight_train). Here "she" is not an entity. *HDSKG* identifies the pronouns in a sentence and replace by suitable nouns. If some pronouns can not be replaced, we will not extract the corresponding relations, because the relation triples between the pronouns and the nouns are useless to the knowledge graph. The other diffidence is that openIE only extract the relation triples but can not estimate the domain reverence of them. So there are many redundancy relation triples will be extracted by openIE.

5.3.3 Evaluation and Results Analysis

5.3.3.1 Classifier Comparison and Accuracy Evaluation of Iterations

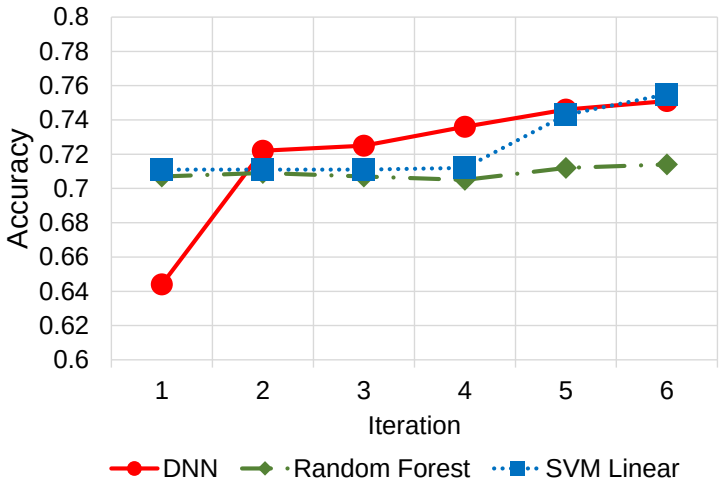


Figure 5.6: Accuracy of Classifiers in each Iteration

We do 10-fold cross-validation to evaluation the accuracy in each iteration. The 1000 labeled data (contain 467 positive data and 523 negative data) are separated in two parts, 900 of them are used as training data and 100 of them are used as testing data. In every cross-validation, the prediction confidences of every prediction data are recorded. After 10 times cross-validation, only the prediction data which $confidence > 0.9$ or $confidence < 0.1$ in all the 10 times

cross-validation are chose to append to training data for the next iteration. Three classifiers are chose to do the comparison including DNN (Deep Neural Networks), Random Forest and SVM Linear. Fig.5.6 is the accuracy of different classifiers in each iteration. It shows the performance of the DNN is the most sensitive classifier to the number of training data, so if we can get large number of training data, the best classifier should be DNN. SVM linear classifier achieves the highest accuracy with the 6 times self-training iterations. The accuracy increase from 0.71 to 0.76, which better than random forest (estimators number = 500) and 4 layers DNN with the $hidden\_units = [200, 400, 100, 50]$.

5.3.3.2 Features Contribution Analysis

For exploration the effect of different features set, we train 5 SVM classifiers using different features groups. Each classifier only uses text features, corpus features, concept features, source features and all features, respectively. As shown in Fig.5.7, the classifier with all features performs best. From Fig.5.7 we also can infer that all these features we design are useful in estimation the domain relevance of relation triples.

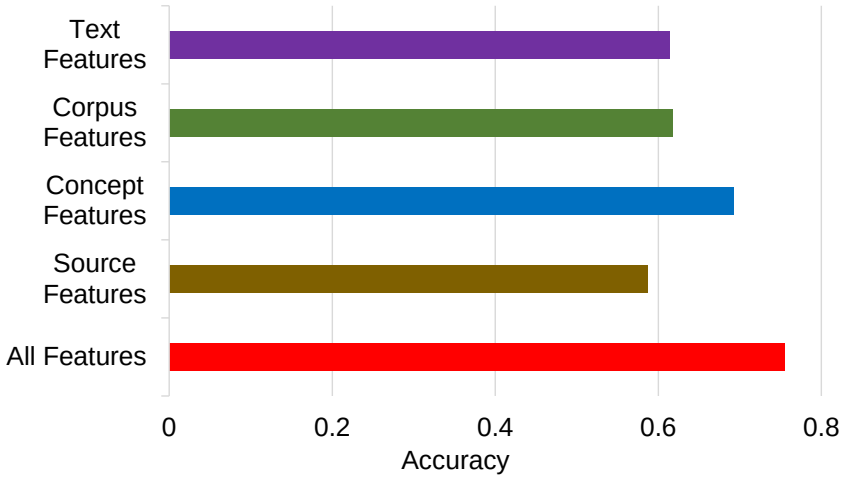


Figure 5.7: Accuracy of Classifiers with Different Features

5.3.3.3 Performance Comparison

We further explore the performance of *HDSKG*. Tab. 5.5 shows the extractions of traditional Rule Based Chunking, openIE, *HDSKG Chunking* and *HDSKG Domain Relevance Estimation*.

Table 5.5: Relation Triples Extracted From Different Extraction Methods

Extraction Method \ Content of Sentence	PyTables is built on top of the HDF5 library, using the Python language and the NumPy package.	Jansi is a small java library that allows you to use ANSI escape codes to format your console output which works even on windows.
Rule Based Chunking Extraction	(PyTables; is_built_on_top_of; HDF5_library)	(Jansi; is; small_java_library) (ANSI; escape; codes)
openIE Extraction	(PyTables; using;,Python_language) (PyTables; is; built) (PyTables; is_built_on_top_of; HDF5_library)	(Jansi; is; small) (ANSI; escape; codes)
HDSKG Chunking Extraction	(PyTables; using; Python_language) (PyTables; using; NumPy_package) (PyTables; is_built_on_top_of; HDF5_library)	(Jansi; is; small_java_library) (ANSI; escape; codes)
HDSKG Domain Relevance Estimation Extraction	(PyTables; using; Python_language) (PyTables; using; NumPy_package) (PyTables; is_built_on_top_of; HDF5_library)	(Jansi; is; small_java_library)
Ground Truth	(PyTables; using; Python_language) (PyTables; using; NumPy_package) (PyTables; is_built_on_top_of; HDF5_library)	(Jansi; is; small_java_library)

This example indicates that compare with traditional Rule Based Chunking, *HDSKG Chunking* can get more relation triples which can not extract by simple morphology matching. The traditional Rule Based Chunking can only chunk the relation triples (PyTables; is_built_on_top_of; HDF5_library) from sentence *"PyTables is built on top of the HDF5 library, using the Python language and the NumPy package."*, but can not chunk (PyTables; using; NumPy_package) because the morphology of this relation triples is not continuous (NP,VP,NP). With the using of dependency parsing, we can get the xcomp (using-11, built-3), dobj (using-11, language-14), and dobj (using-11, package-18). These dependencies mean that "using-11" is open clausal complement of "built-3", and the object of "using-1" is "language-14" and "package-18". From scenario 5 of *HDSKG Chunking*, we can chunk (PyTables; using; NumPy_package) and (PyTables; using; Python_language).

openIE derives the latent relation triples (PyTables; using; Python_language), but ignores the paralleled relation triples (PyTables; using; NumPy_package). But this relation triples captures by *HDSKG* accurately by the scenario 4 designed in our system. In addition to above shortcoming, openIE also generates many redundant relation triples like (PyTables; is; built). *HDSKG* does not make this mistake due to our VP and NP chunking can identify the verb phrase "is built on top of".

For the sentence "Jansi is a small java library that allows you to use ANSI escape codes to format your console output which works even on windows.", all the above tree methods

extract the wrong relation triples (ANSI; escape; codes). Then we can use the *HDSKG Domain Relevance Estimation* to filter this irrelevant relation triples. From the Tab. 5.5 we can see the *HDSKG Domain Relevance Estimation* can reserve the correct relation triples and filter out the irrelevant relation triples accurately.

Compare with the above 2 methods, *HDSKG Chunking* extracts both apparent and latent relation triples, and the domain relevance estimation of *HDSKG* filters the relation triples which less relevant to the target domain.

Precision, recall and f-measure are used as evaluation metrics of the extractions of *HDSKG*. Precision is the ratio of the correct relation triples to all extractions, this metric is seen as a measure of quality. Recall is the ratio of the correct relation triples to all ground truth relation triples, which is a measure of completeness. F-measure is computed by:

$$F = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.5}$$

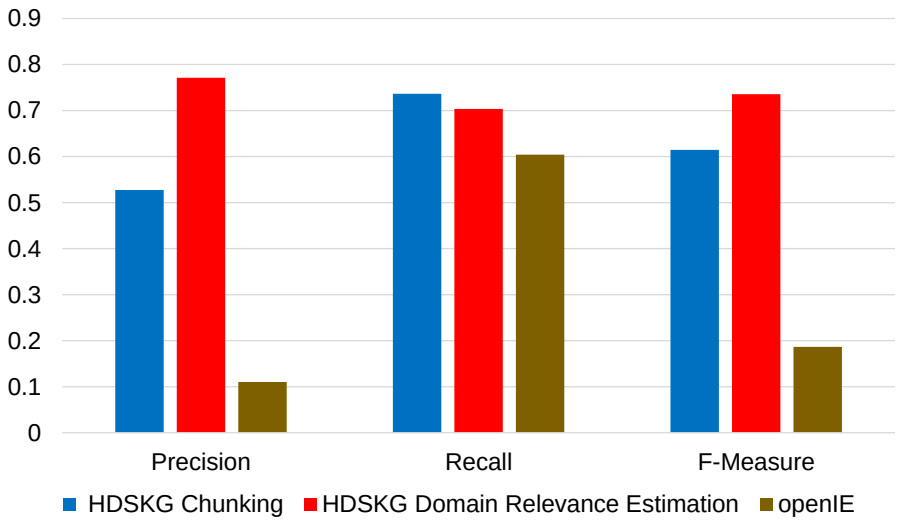


Figure 5.8: Performance of different Extraction Methods

Fig. 5.8 shows the performance of different extracting methods, compare with openIE, *HDSKG domain relevance estimation* achieves the highest precision 0.77 and *HDSKG Chunking* get the highest recall 0.74. According to the result, even we lose 0.04 recall by estimation of domain relevance, the precision is be improved 0.25. This illustrates that *HDSKG Domain Relevance Estimation* estimates the domain relevance with small loss of recall but improves

the precision conspicuously. The recall performance of the openIE is 0.6, which is close to the recall of *HDSKG Domain Relevance Estimation*. But this recall of openIE is at the expense of reduction of the precision by generating more irrelevance relation triples.

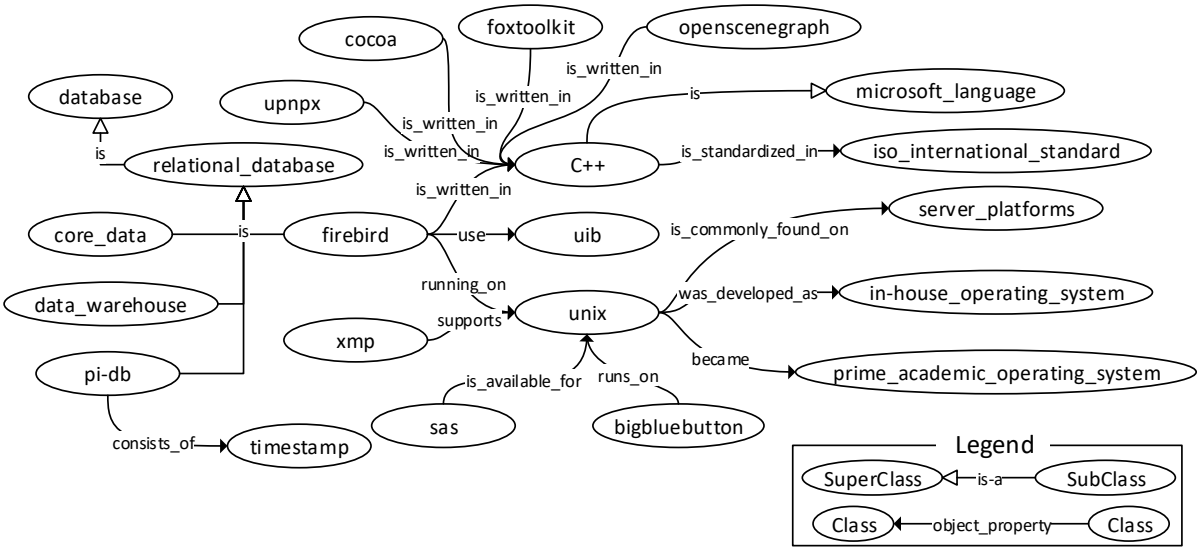


Figure 5.9: An Example of Knowledge Graph Generated by *HDSKG*

The precision value between openIE and *HDSKG* is different. It is because the following reasons:

- *HDSKG* combines more advanced features of the sentence, which can extracted the relation triples more comprehensive.
- Our definitions about "correct" relation triples are different. The experts who make the baseline reach a consensus that some relation triples extracted by openIE can not be regarded as a correct relation triples. For example, for the openIE, (I, like, JavaScript) is regarded as correct relation triples. But the subject ŚIŠ is not a concept or entity. This relation triples is meaningless to add in a knowledge graph and will affect the quality of the knowledge graph. So we made a more strict standard to evaluated the relation triples.

So, comparing with openIE, *HDSKG* can produce less relation triples with higher precision.

Fig. 5.9 is part of the knowledge graph generate by *HDSKG* in the software engineering domain.

5.4 Discussion

So far, we have extracted a knowledge graph of software engineering domain from the tag wiki of Stack Overflow. In this section, we discuss the implications of our research for knowledge graph extraction, information retrieval of software engineering domain, and knowledge representation.

5.4.1 Implications for Knowledge Graph Extraction

As shown in Section 5.2.4, *HDSKG Chunking* incorporates a dependency parser with a rule-based method and extracts the candidate relation triples in the natural language materials with high precision and low recall. The 6 scenarios we present in Section 5.2.4.3 cover most of the occasions in natural language expressions. It ensures the *HDSKG* can easy to be used in other natural language materials, e.g., healthcare, industrial design, etc.

For the domain relevance estimation part of *HDSKG*, the designed features can distinguish domain relevant relation triples from the candidate relation triples efficiently as shown in Section 5.3.3.2. *HDSKG* is easy to migrate to other domains by just changing the domain lexicon to the target domain and labelling a small number of training data. With the self-training SVM classifier, *HDSKG* can construct the knowledge graph of of the target domain automatically.

5.4.2 Implications for Information Retrieval in Software Engineering Domain

The knowledge graph of software engineering domain, the traditional way of information retrieval can be changed.

Firstly, a direct answer search engine can be constructed [185, 186]. As it shown in Fig. 5.9, there are 2 kinds of relations, they are “is a” and “object property”. “is a” is the taxonomic relations which indicate a containment relation between 2 concepts [187]. For example, the taxonomic relation like “pi-db” -> “relational_database” -> “database” in Fig. 5.9. “pi-db” is subclass of “relational_database”, and “relational_database” is subclass of “database”. So we can derive if we have a sentence like "The employees’ information is put in firebird", this sentence "The employees’ information is put in database" is also right. The “object property” is the non-taxonomic relations, these relations present some properties of the concepts. So if

there is a query like: *"Which database can run on Unix and be written in C++"* From the query we can extract the VP "run on" and "be written in", and the concepts "database", "Unix" and "C++". Then the "firebird" will be recommended directly. This direct answer search engine can answer many questions which ask about a specific concept and help developers to find available technologies without reading many webpages.

We can also use this knowledge graph to the semantic search engine because it links the concepts across query key words, documents, databases, webpages, etc. via semantic modeling.. For example, the phrase "firebird is written in C++" and the phrase "C++ is microsoft language" are extracted from different webpages. There is no link between the concept "firebird" and the concept "microsoft language". But in the knowledge graph, these two concepts connect to "C++". So in the search results, the ranking algorithms should concern the weight to "microsoft language" related webpages when search "firebird".

5.4.3 Implications for Knowledge Representation

Our knowledge graph is also a structural knowledge representation which can be used to do data documentation and enrich search result pages. Actually Stack Overflow paid effort on it already. There is a new online function which is still under beta test named documentation[1]. In this part, Stack Overflow organizes the information of each tag structural like "different version", "important functions" and "useful code snippets", etc. Compared to the traditional tag wiki, developer can easy to locate the information they want. But these documentations are contributed by the users of Stack Overflow manually, they are low efficient and subjective. The knowledge graph extracted by *HDSKG* can assist to enrich the documentation. Because it is extracted from the materials contributed by many developers, with the "corpus features" we used in Section 5.2.5.1, the subjective relation triples can be removed by the classifier. For example, there are five relations of "is written in" in Fig. 5.9, these relations can be used to enrich the documentation of tag "C++" by the tools or libraries which are written by "C++".

Our knowledge graph also can be used to enrich the search result. For example, if you search "iso standard", as the result in Fig. 5.9, we will recommend "C++" which is standardized by "iso standard".

[1] http://stackoverflow.com/documentation

5.5 Summary

Knowledge graph integrates structural information of concepts across multiple information sources, and links these concepts together. It is useful in many domains such as search result ranking, recommendation, exploratory search, etc. In this chapter, an implementation of the *BotLearning* module of *XBot* called *HDSKG* was proposed to discover domain-specific concepts and their relation triples from the content of webpages. By incorporation of a dependency parser with a rule-based method, we got the candidate relations triples candidates with high precision and recall. And by utilizing the self-training SVM classifier, we could estimate the domain relevance of the candidate relation triples. We applied *HDSKG* to Stack Overflow (a Q&A website about computer programming) for evaluation. As a result, a knowledge graph of the software engineering domain is constructed. The experimental results show that both the precision and recall of *HDSKG* achieve new state-of-the-art. The performance is particularly efficient in the case of complex sentences. Furthermore, with the self-training technique we used in the classifier, *HDSKG* can be applied to other domain easily with less training data.

Chapter 6

Conclusion and Future Work

This chapter will summarize the work of this thesis and indicate some research directions inspiring from this thesis.

6.1 Conclusion

In this thesis, we conducted a series of research works to realize an explainable Q&A system based on the domain-specific knowledge graph.

First, in order to provide human-readable and understandable explanations, this thesis explored the human Q&A cognitive process [35]. Inspired by the research works of cognitive science, a framework called *XBot* is proposed. This framework consists of five modules: *BotPerception*, *BotPlanning*, *BotReasoning*, *BotResponse* and *BotLearning*. *XBot* is aligned with the cognitive process of human and can serve as the basis for designing a Q&A system based on the knowledge graph with human-like explainability. The experiment results show that compared with the existing Q&A systems, the Q&A systems based on *XBot* framework has the capacity to answer questions with human-like explainability by learned knowledge.

Second, a Q&A system based on *XBot* framework called *DeveloperBot* was presented. *DeveloperBot* proposed the implementations of the four modules of *XBot* including *BotPerception*, *BotPlanning*, *BotReasoning* and *BotResponse*. And further customized by loading the knowledge graph of general knowledge of the software engineering domain. The *BotPerception* and *BotPlanning* modules are implemented by a query graph construction algorithm. This algorithm incorporates the Dependency Parsing into the Tree Parsing to represent the question. Then it splits a complex query into several simple constraints by dependency parsing and integrates

these constraints and their solving order into a multi-layer query graph. The *BotReasoning* module is implemented by a fast graph cyclic pruning reasoning algorithm. This algorithm models the constraints solving process as subgraph search and decision-making process. The reasoning subgraph will be extracted as the qualitative explanations of "why", "how" an answer is be recommended. And the confidence of direct answer will be computed as the quantitative explanation of "how confident" an answer is.

Quantitative evaluation and a user study were conducted to evaluate the performance of *DeveloperBot*. The results of the quantitative evaluation show that, with topological structure, predicate similarity and other novel features of the subgraph, the Bayesian decision theory and DNN algorithm can extract the correct answers of constraints with high accuracy and low confidences MSE. The participants of the user study express that *DeveloperBot* has the capacity to model and reason complex problems. They think that *DeveloperBot* is a good supplement to the search engine. The experimental data also shows similar results: compared with just using Google, with the assist of *DeveloperBot*, the users can find answers faster and with more accuracy. In addition, using the reasoning subgraph and answer confidence as the explanation of the direct answers can significantly improve the developers' trust and adoption to the answers. These explanations also assist the developers to understand the answers deeply and improve the answer accuracy and formed better search keywords. Furthermore, the more complex the question is, the more effective the *DeveloperBot* can achieve.

Third, a implementation of *BotLearning* module of *XBot* called *HDSKG* was proposed to extract domain-specific knowledge graph from the content of webpages. By incorporation of a dependency parser with a rule-based method, *HDSKG* extracts the candidate relation triples with high precision and recall. *HDSKG* further extracts novel features of these candidate relation triples and estimates the domain relevance of them by utilizing a self-training SVM classifier.

We applied *HDSKG* to Stack Overflow (a Q&A website about computer programming) for evaluation. As a result, we constructed a knowledge graph of general knowledge of software engineering domain with 35279 relation triples, 44800 concepts, and 9660 unique verb phrases. The experimental results show that both the precision and recall of *HDSKG* are much higher than the state-of-the-art tool called openIE for relation triples extraction. The performance is particularly efficient in the case of complex sentences. Furthermore, with the self-training technique used in the classifier, *HDSKG* can be applied to other domain easily with less training data.

These results can indicate that the cognitive Q&A framework *XBot* could effectively serve as the basis for the Q&A system. A series of corresponding implementations of this framework proves that *XBot* can be adopted and customized in different domain. With the framework and corresponding solutions, the Q&A system can answer complex questions with explainability based on the knowledge graph. And such a Q&A system can effectively improve the efficiency of closed-end question search. This research work has the potential to extend to meet broader requirements like complex text-based software engineering artifacts.

6.2 Future Work

6.2.1 *DoctorBot*

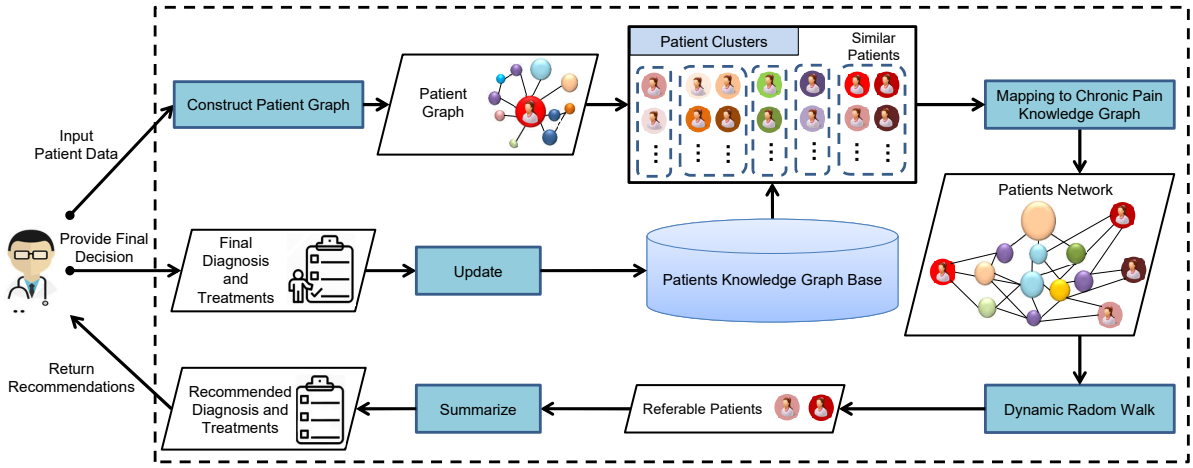


Figure 6.1: Framework of *DoctorBot*

The studies conducted in this thesis could be extended in various directions, and an example of these possible research directions will be discussed in this section. As mentioned before, *XBot* can be customized in different domains which just like humans can learn different knowledge and become experts of different domains. We already showed a customization example in Chapter 1 which customized *XBot* as a search engine assistant for developers called *DeveloperBot*. In this section, we will present anther customization example called *DoctorBot* which customized

by loading the knowledge graph of chronic pain domain into the knowledge base of the *XBot* framework.

DoctorBot is used as a doctor assistant for chronic pain diagnosis and treatments recommendation. As shown in Fig. 6.1, the patients' knowledge graph base is corresponding to the experience of the doctors in their semantic memory. Every patient in the patients' graph base has their corresponding diagnosis and treatments which extracted from the electronic medical record. The duty of *DoctorBot* is to find the best reference cases and summarize the diagnosis and treatments to recommend to the input patient.

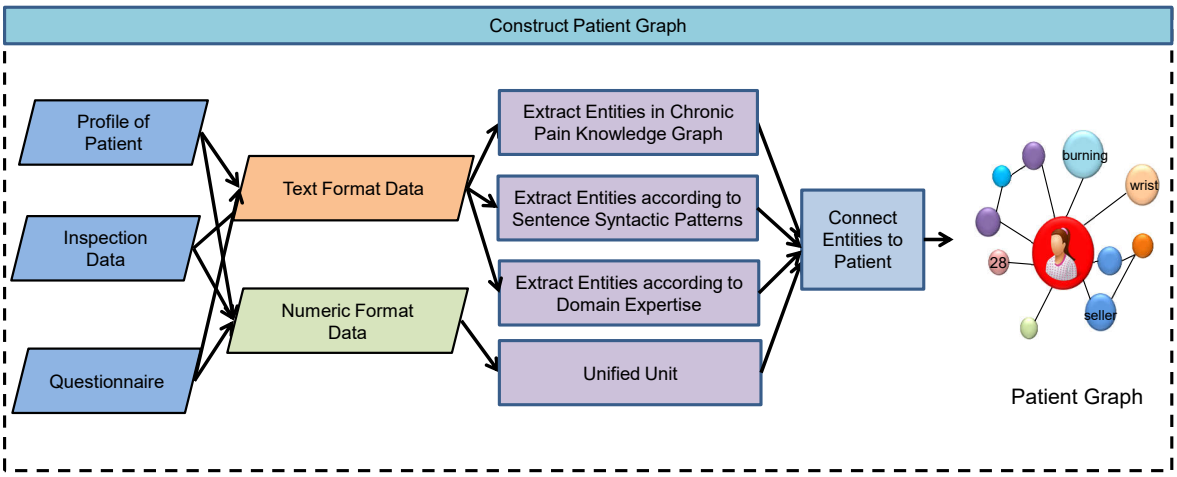


Figure 6.2: *BotPerception*: Patient Graph Construction

DoctorBot input the profile, questionnaire and inspection data of a patient, and construct these data as a patient graph by different entities extraction methods shown in Fig. 6.2. This is implementation of *BotPerception* module of *XBot*.

After that, *DoctorBot* will cluster the input patient graph with the patient graphs in patients graph base and find similar patients. This process like the *BotPlanning* of *XBot* which divide a patient searching task into small tasks limited to a few patients. Then *BotReasoning* algorithm of *DoctorBot* will connect the input patient graph and the similar patients' patient graphs into the chronic pain knowledge graph. A dynamic random walk algorithm will be used to reason the most valuable referring patients. At the end, the *DoctorBot* will learn the new diagnosis and

Figure 6.3: *BotResponse*: the User Interfaces of *DoctorBot*

treatments provide by the doctor and update the patients' graph base. The UI of *BotResponse* is shown in Fig. 6.3.

In addition to *DoctorBot*, *XBot* can be customized in many different domain. For example, *XBot* can be trained by knowledge of music, dance, painting, etc. and become *MusicBot*, *DanceBot*, *PaintingBot*. And just like humans need different strategies to learn different domain knowledge, *XBot* can be implemented in different *BotPerception*, *BotPlanning* and *BotReasoning* methods. The explanation also can be more devise and flexible. Except for the above situations, open question answer based on knowledge graph also an interesting but difficult problem to solve for Q&A system.

6.2.2 True or False Questions Answering

Except for closed-end questions, we can also extend the Q&A to other types of questions like true or false questions. But the human brain has different reasoning logics and strategies for different questions. The future work mainly focuses on the reasoning logic of closed-end question, true or false question-categorization task and true or false question-function validation task as shown in Tab. 6.1. In order to find the right answer, decide whether a concept belongs

Table 6.1: Types of Relations Found in SG that Constitute Positive Or Negative Evidences

No.	Positive Evidence	Negative Evidence
	True or False Question-Categorization Task	
1	The subject and object have positive connection of subclass to superclass	The subject and object have negative connection of subclass to superclass
2	The subject have common property to the subclasses of object	The subject have distinguishing property to the subclasses of object
3	The subject have have crucial common property to the subclasses of object	The subject have crucial distinguishing property to the subclasses of object
	True or False Question-Property Validation Task	
1	The subject has positive $R1$	The subject has negative $R1$
2	The subject has positive $R2$	The subject has negative $R2$
3	The subject has positive $R3$	The subject has negative $R3$
4	The subject has positive $R4$	The subject has negative $R4$

to a category or whether a function is supported, sufficient evidence needs to be gathered to determine whether it surpasses the positive or negative criteria. Tab. 6.1 summaries the positive and negative evidence that can be used to make different decisions. In general, when property comparison, the weights of positive and negative evidence are not equal. A little negative evidence may lead to negative decisions, whereas much positive evidence is needy in order to make a positive decision.

True or False Question-Categorization Task: true or false question means the answer to the question is to determine whether the question is true or false. Categorization task means the question is to determine the subordination of two concepts. e.g. “If neo4j is a graph database?” In this question, “neo4j” is subject and “graph database” is an object. As shown in Tab. 6.1, there are 3 evidence pairs to evaluate these kinds of question task. If the positive evidence No.1 is founded in SG, the answer will be true with confidence is 1. Otherwise, if the negative evidence No.1 is founded, the answer will be false with confidence is 1. In the case of the positive evidence and negative evidence are founded at the same time, the answer will be false with confidence is 1.

Once the No.1 evidence pair is not founded, the No.2 evidence pair and No.3 evidence pair will be used to do further analysis. The common property in the No.2 positive evidence

means the matching non-taxonomic relations between subject and all subclasses of the object. Otherwise, the distinguishing property means the mismatch non-taxonomic relations between subject and all subclasses of object (predicate express the opposite meaning). These evidences are represented as follows:

$$\begin{aligned} NTScore_{pst} &= \frac{1}{m}\sum_{i=1}^{m} PredicateScore_i, \\ NTScore_{neg} &= \frac{k}{m}, k = (0, 2, ..., m), \end{aligned} \tag{6.1}$$

where m is the number of common properties. $PredicateScore_i$ is the predicate relevance of ith common property. k is the number of distinguishing properties.

Similarly, the crucial property of the object is the non-taxonomic relations which repeat in most of the subclasses of the object. For the "graph database", the crucial property should be "support graph storage". To the contrary, the crucial distinguish property should be "do not support graph storage". These evidences are represented as follows:

$$\begin{aligned} CNTScore_{pst} &= \frac{1}{n}\sum_{j=1}^{n} PredicateScore_j, \\ CNTScore_{neg} &= \frac{p}{n}, p = (0, 2, ..., n), \end{aligned} \tag{6.2}$$

where n is the number of crucial common properties. $PredicateScore_j$ is the predicate relevance of jth crucial common property. p is the number of crucial distinguishing properties.

Note that, the evidence pair No.2 and No.3 are continuous but not independent. So during decision making, $p(\boldsymbol{x}|\omega_i)$ of Equation. 4.16 should be d-dimensional Gaussian probability density as follows:

$$p(\boldsymbol{x}|\omega_i) = \frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma_i}|}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu_i})^{\mathrm{T}}\boldsymbol{\Sigma_i}^{-1}(\boldsymbol{x}-\boldsymbol{\mu_i})\right], \tag{6.3}$$

where $\boldsymbol{\mu_i}$ is the mean vector of $\boldsymbol{x}$ from class ω_i. $\boldsymbol{\Sigma_i}$ is the covariance matrix of $\boldsymbol{x}$ from class ω_i. $\boldsymbol{x} = (x1, x2)^{\mathrm{T}}$, here $x_1 = NTScore_{pst}$ and $x_2 = CNTScore_{pst}$ during compute the posteriori probability of true answer (assume ω_1). During compute the posteriori probability of false answer (assume ω_2), $x_1 = NTScore_{neg}$ and $x_2 = CNTScore_{neg}$.

In the special case of $d = 2$ and 2 classes, Bayesian discriminant function $g(\boldsymbol{x})$ of $\omega_i, i = 1, 2$ is given by the following:

$$g(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}}\eta_1\boldsymbol{x} - \boldsymbol{x}^{\mathrm{T}}\eta_2\boldsymbol{x} + \boldsymbol{\gamma_1}^{\mathrm{T}}\boldsymbol{x} - \boldsymbol{\gamma_2}^{\mathrm{T}}\boldsymbol{x} + \theta_1 - \theta_2, \tag{6.4}$$

where $\eta_i = -\frac{1}{2}\boldsymbol{\Sigma_i}^{-1}$, $\boldsymbol{\gamma_i} = \boldsymbol{\Sigma_i}^{-1}\boldsymbol{\mu_i}$ and $\theta_i = -\frac{1}{2}\boldsymbol{\mu_i}^{\mathrm{T}}\boldsymbol{\Sigma_i}^{-1}\boldsymbol{\mu_i} - \frac{1}{2}ln|\boldsymbol{\Sigma_i}| + lnp(\omega_i)$. Here the decision $g(x) = 0$ is a quadratic hyperplane.

True or False Question-Property Validation Task: Property validation task means the question is to determine whether a subject has a specific property. e.g. "If graph database support rdf query language?" In this question, "graph database" is subject and "rdf query language" is an object. As shown in Tab. 6.1, there are 4 evidence pairs to evaluate these kinds of question task.

Here we assume that if the answer is true and confidence value is 1, which means all the subclasses of the subject have the property to all the subclasses of the object, and the all predicate relevances are 1. For the $R1$, $R2$ and $R3$, we will give the properties of superclasses all the corresponding subclasses.

We synthesize predicate relevances and the ratio of all the subject subclass and object subclass pairs that have the corresponding property to the pairs do not have as follows:

$$\begin{aligned} RelScore_{pst} &= \frac{1}{mn}\sum_{i=1}^{m+n} max(PredicateScore_{ij}), j = (1, 2, ..., p), \\ RelScore_{neg} &= \frac{k}{mn}, k = (0, 2, ..., q), \end{aligned} \tag{6.5}$$

where m and n are the number of leaf subclasses of SG_{subj} and SG_{obj}. $PredicateScore_i$ is the ith subject leaf subclass and object leaf subclass pairs, here we take the highest predicate relevance among this relevance while the leaf nodes contain one more (assume p) relations. k is the subject leaf subclass and object leaf subclass pairs which contain negative relations. If a pair contains both positive relation and negative relation, we regard this node as negative.

The R_1, R_2, R_3 and R_4 is independent, so during decision making, the $RelScore_{pst}$ and $RelScore_{neg}$ will be used as the positive evidence and negative evidence (x in Equation.4.13) to the Bayesian decision theory. For the $\omega_i (i = 1, 2, ..., s)$ in Equation.4.13, the class number is two. ω_1 means the answer is true, and ω_2 means the answer is false.

6.2.3 Evaluate *HDSKG* with Public Datasets

In Section 5.3, we only test *HDSKG* on the dataset of software engineering domain. Test *HDSKG* on the public dataset can make our method more convinced. It is a pity that when I conducted this research work, as best I know, for the techniques which extracted knowledge graph from the text (provided text->relation triples both for open information extraction), there were still do not have a commonly accepted public dataset. At present, the development time of knowledge graph construction technology is relatively long. There are some generally accepted public datasets emerged [188, 189, 190], maybe we can improve our tool and the evaluation in the future.

Publication

Working on this thesis produced the following eight publications:

(1) "HDSKG: Harvesting domain specific knowledge graph from content of webpages."
Xuejiao Zhao, Zhenchang Xing, Muhammad Ashad Kabir, Naoya Sawada, Jing Li, and Shang-Wei Lin. 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 56-67, IEEE, 2017. (This paper relates to Chapter. 5.)

(2) "*DeveloperBot*: an Explainable Q&A System based on Knowledge Graph."
Xuejiao Zhao, Chunyan Miao, Huanhuan Chen, Zhenchang Xing, Naoya Sawada. IEEE Transactions on Neural Networks and Learning Systems (TNNLS, Under Review). (This paper relates to Chapter. 3 and Chapter. 1.)

(3) "Identifying Cognitive Distortion by Convolutional Neural Network based Text Classification."
Xuejiao Zhao, Chunyan Miao, and Zhenchang Xing. International Journal of Information Technology, pp. 1-12, vol. 23, no. 1, 2017.

(4) "HDSO: Harvest Domain Specific Non-taxonomic Relations of Ontology from Internet by Deep Neural Networks (DNN)."
Xuejiao Zhao. Big Software on the Run: Where Software meets Data (BSR Winter School), pp. 74-79, 2017. (This paper relates to Chapter. 5.)

(5) "Improving API Caveats Accessibility by Mining API Caveats Knowledge Graph."
Li, Hongwei, Sirui Li, Jiamou Sun, Zhenchang Xing, Xin Peng, Mingwei Liu, and **Xuejiao Zhao**. 2018 IEEE 34th The International Conference on Software Maintenance and Evolution (ICSME), pp. 183-193, IEEE, 2018. (IEEE TCSE Distinguished Paper Award)

(6) "A comprehensive exploration to the machine learning techniques for diabetes identification." Wei, Sidong, **Xuejiao Zhao**, and Chunyan Miao. In IEEE 4th World Forum on Internet of Things (WF-IoT), pp. 291-295, IEEE, 2018.

(7) "From discussion to wisdom: web resource recommendation for hyperlinks in stack overflow." Li, Jing, Zhenchang Xing, Deheng Ye, and **Xuejiao Zhao**. In Proceedings of the 31st Annual ACM Symposium on Applied Computing (SAC), pp. 1127-1133, ACM, 2016.

(8) "amassist: In-ide ambient search of online programming resources." Li, Hongwei, **Xuejiao Zhao**, Zhenchang Xing, Lingfeng Bao, Xin Peng, Dongjing Gao, and Wenyun Zhao. In IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pp. 390-398, IEEE, 2015.

References

[1] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[2] A. M. Turing, "Computing machinery and intelligence," in *Parsing the turing test*, pp. 23–65, Springer, 2009.

[3] A. Hodges, *Alan Turing: The Enigma: The Enigma*. Random House, 2012.

[4] G. Oppy and D. Dowe, "The turing test," 2003.

[5] A. Murgia, D. Janssens, S. Demeyer, and B. Vasilescu, "Among the machines: Human-bot interaction on social q&a websites," in *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pp. 1272–1279, 2016.

[6] A. Hakkoum and S. Raghay, "Semantic q&a system on the qurŠan," *Arabian Journal for Science and Engineering*, vol. 41, no. 12, pp. 5205–5214, 2016.

[7] B. Hammo, H. Abu-Salem, S. L. Lytinen, and M. Evens, "Qarab: A: Question answering system to support the arabic language," in *Proceedings of the ACL-02 workshop on Computational approaches to semitic languages*, 2002.

[8] N. Alvarado, S. S. Adams, S. Burbeck, and C. Latta, "Beyond the turing test: Performance metrics for evaluating a computer simulation of the human mind," in *Proceedings 2nd International Conference on Development and Learning. ICDL 2002*, pp. 147–152, IEEE, 2002.

[9] H.-C. Tu and J. Hsiang, "An architecture and category knowledge for intelligent information retrieval agents," *Decision Support Systems*, vol. 28, no. 3, pp. 255–268, 2000.

[10] S.-S. Liaw and H.-M. Huang, "An investigation of user attitudes toward search engines as an information retrieval tool," *Computers in human behavior*, vol. 19, no. 6, pp. 751–765, 2003.

[11] M. Wang, Y. Zou, Y. Cao, and B. Xie, "Searching software knowledge graph with question," in *International Conference on Software and Systems Reuse*, pp. 115–131, Springer, 2019.

[12] C. Zhu, K. Ren, X. Liu, H. Wang, Y. Tian, and Y. Yu, "A graph traversal based approach to answer non-aggregation questions over dbpedia," in *Joint International Semantic Technology Conference*, pp. 219–234, Springer, 2015.

[13] R. Hong, M. Wang, G. Li, L. Nie, Z.-J. Zha, and T.-S. Chua, "Multimedia question answering," *IEEE MultiMedia*, no. 4, pp. 72–78, 2012.

[14] M.-D. Olvera-Lobo and J. Gutiérrez-Artacho, "Searching health information in question-answering systems," in *Handbook of Research on ICTs for Human-Centered Healthcare and Social Care Services*, pp. 474–490, IGI Global, 2013.

[15] S. Quarteroni and S. Manandhar, "A chatbot-based interactive question answering system," *Decalog 2007*, vol. 83, 2007.

[16] S. Varges, F. Weng, and H. Pon-Barry, "Interactive question answering and constraint relaxation in spoken dialogue systems," in *Proceedings of the 7th SIGdial Workshop on Discourse and Dialogue*, pp. 28–35, 2006.

[17] G. Maheshwari, P. Trivedi, D. Lukovnikov, N. Chakraborty, A. Fischer, and J. Lehmann, "Learning to rank query graphs for complex question answering over knowledge graphs," in *International Semantic Web Conference*, pp. 487–504, Springer, 2019.

[18] G. Murphy, *The big book of concepts*. MIT press, 2004.

[19] T. Miller, "Explanation in artificial intelligence: Insights from the social sciences," *Artificial Intelligence*, vol. 267, pp. 1–38, 2019.

[20] Y. Zhang and X. Chen, "Explainable recommendation: A survey and new perspectives," *arXiv preprint arXiv:1804.11192*, 2018.

[21] Q. Ai, V. Azizi, X. Chen, and Y. Zhang, "Learning heterogeneous knowledge base embeddings for explainable recommendation," *Algorithms*, vol. 11, no. 9, p. 137, 2018.

[22] R. Catherine, K. Mazaitis, M. Eskenazi, and W. Cohen, "Explainable entity-based recommendations with knowledge graphs," *arXiv preprint arXiv:1707.05254*, 2017.

[23] X. Yu, X. Ren, Y. Sun, Q. Gu, B. Sturt, U. Khandelwal, B. Norick, and J. Han, "Personalized entity recommendation: A heterogeneous information network approach," in *Proceedings of the 7th ACM international conference on Web search and data mining*, pp. 283–292, 2014.

[24] J. Yao and Y. Yao, "Web-based information retrieval support systems: building research tools for scientists in the new information age," in *Proceedings IEEE/WIC International Conference on Web Intelligence (WI 2003)*, pp. 570–573, IEEE, 2003.

[25] H. Xu and A. Li, "Two-level smart search engine using ontology-based semantic reasoning.," in *SEKE*, pp. 648–652, 2014.

[26] W. Ma, M. Zhang, Y. Cao, W. Jin, C. Wang, Y. Liu, S. Ma, and X. Ren, "Jointly learning explainable rules for recommendation with knowledge graph," in *The World Wide Web Conference*, pp. 1210–1221, ACM, 2019.

[27] S. Chari, D. M. Gruen, O. Seneviratne, and D. L. McGuinness, "Foundations of explainable knowledge-enabled systems," *arXiv preprint arXiv:2003.07520*, 2020.

[28] X. Yanghua, "Opportunities and challenges of explainable artificial intelligence based on knowledge graph,"

[29] R. High, "The era of cognitive systems: An inside look at ibm watson and how it works," *IBM Corporation, Redbooks*, pp. 1–16, 2012.

[30] Y. Wang, "Cognitive learning methodologies for brain-inspired cognitive robotics," *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI)*, vol. 9, no. 2, pp. 37–54, 2015.

[31] Y. Wang, "Deep reasoning and thinking beyond deep learning by cognitive robots and brain-inspired systems," in *2016 IEEE 15th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)*, pp. 3–3, IEEE, 2016.

[32] A. D. W. Sumari, A. S. Ahmad, A. I. Wuryandari, and J. Sembiring, "Brain-inspired knowledge growing-system: towards a true cognitive agent," *International Journal of Computer Science & Artificial Intelligence (IJCSAI)*, vol. 2, no. 1, pp. 26–36, 2012.

[33] Y. Wang, B. Widrow, L. A. Zadeh, N. Howard, S. Wood, V. C. Bhavsar, G. Budin, C. Chan, R. A. Fiorini, M. L. Gavrilova, *et al.*, "Cognitive intelligence: Deep learning, thinking, and reasoning by brain-inspired systems," *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI)*, vol. 10, no. 4, pp. 1–20, 2016.

[34] Y. Wang and V. Chiew, "On the cognitive process of human problem solving," *Cognitive systems research*, vol. 11, no. 1, pp. 81–92, 2010.

[35] R. Confalonieri, T. R. Besold, T. Weyde, K. Creel, T. Lombrozo, S. T. Mueller, and P. Shafto, "What makes a good explanation? cognitive dimensions of explaining intelligent machines.," in *CogSci*, pp. 25–26, 2019.

[36] R. R. Hoffman, S. T. Mueller, G. Klein, and J. Litman, "Metrics for explainable ai: Challenges and prospects," *arXiv preprint arXiv:1812.04608*, 2018.

[37] S. Bernecker, "Memory knowledge," *The Routledge Companion to Epistemology. New York: Routledge*, pp. 326–34, 2011.

[38] L. R. Squire, "Declarative and nondeclarative memory: Multiple brain systems supporting learning and memory," *Journal of cognitive neuroscience*, vol. 4, no. 3, pp. 232–243, 1992.

[39] M. R. Quillan, "Semantic memory," tech. rep., BOLT BERANEK AND NEWMAN INC CAMBRIDGE MA, 1966.

[40] D. Van Knippenberg, L. Dahlander, M. R. Haas, and G. George, "Information, attention, and decision making," 2015.

[41] J. G. Lynch, "Memory and decision making," 1991.

[42] G. Giguère and B. C. Love, "Limits in decision making arise from limits in memory retrieval," *Proceedings of the National Academy of Sciences*, vol. 110, no. 19, pp. 7613–7618, 2013.

[43] A. M. Owen, "Cognitive planning in humans: neuropsychological, neuroanatomical and neuropharmacological perspectives," *Progress in neurobiology*, vol. 53, no. 4, pp. 431–450, 1997.

[44] A. M. Collins and E. F. Loftus, "A spreading-activation theory of semantic processing.," *Psychological review*, vol. 82, no. 6, p. 407, 1975.

[45] J. Olds, M. Khan, M. Nayebpour, and N. Koizumi, "Explainable ai: A neurally-inspired decision stack framework," *arXiv preprint arXiv:1908.10300*, 2019.

[46] C. Fellbaum, "A semantic network of english: the mother of all wordnets," in *EuroWordNet: A multilingual database with lexical semantic networks*, pp. 137–148, Springer, 1998.

[47] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr, and T. M. Mitchell, "Toward an architecture for never-ending language learning.," in *AAAI*, vol. 5, p. 3, 2010.

[48] G. Angeli, M. J. Premkumar, and C. D. Manning, "Leveraging linguistic structure for open domain information extraction," *Linguistics*, no. 1/24, 2015.

[49] M. Pasca, D. Lin, J. Bigham, A. Lifchits, and A. Jain, "Organizing and searching the world wide web of facts-step one: the one-million fact extraction challenge," in *AAAI*, vol. 6, pp. 1400–1405, 2006.

[50] A. F. Beavers, "Cognitive science: An introduction to the science of the mind," 2013.

[51] M. A. Boden, *The creative mind: Myths and mechanisms*. Routledge, 2004.

[52] B. Subagdja and A.-H. Tan, "Towards a brain inspired model of self-awareness for sociable agents," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[53] M. Negnevitsky, *Artificial intelligence: a guide to intelligent systems*. Pearson education, 2005.

[54] P. Thagard, *Mind: Introduction to cognitive science*, vol. 17. MIT press Cambridge, MA, 2005.

[55] S. Woll, *Everyday thinking: Memory, reasoning, and judgment in the real world*. Psychology Press, 2001.

[56] M. W. Matlin, *Cognition*. Wiley, 2008.

[57] K.-R. Muller, C. W. Anderson, and G. E. Birch, "Linear and nonlinear methods for brain-computer interfaces," *IEEE transactions on neural systems and rehabilitation engineering*, vol. 11, no. 2, pp. 165–169, 2003.

[58] G. S. Dell, "A spreading-activation theory of retrieval in sentence production.," *Psychological review*, vol. 93, no. 3, p. 283, 1986.

[59] M. S. Gazzaniga, R. B. Ivry, and G. Mangun, *Cognitive Neuroscience. The biology of the mind, (2014)*. Norton: New York, 2006.

[60] C. R. Gallistel and A. P. King, *Memory and the computational brain: Why cognitive science will transform neuroscience*, vol. 6. John Wiley & Sons, 2011.

[61] H. Zhu, Y. Zeng, D. Wang, and B. Xu, "Brain knowledge graph analysis based on complex network theory," in *International Conference on Brain Informatics*, pp. 211–220, Springer, 2016.

[62] X. Huang, L. Wang, J. Ruan, S. Wu, and D. Li, "Exploration research of thinking innovation based on pcst theory and knowledge graph," in *2018 Chinese Automation Congress (CAC)*, pp. 1040–1045, IEEE, 2018.

[63] R. C. Atkinson and R. M. Shiffrin, "Human memory: A proposed system and its control processes," in *Psychology of learning and motivation*, vol. 2, pp. 89–195, Elsevier, 1968.

[64] A. Paivio, "Imagery and memory.," 1995.

[65] P. Wang, Q. Wu, C. Shen, A. v. d. Hengel, and A. Dick, "Explicit knowledge-based reasoning for visual question answering," *arXiv preprint arXiv:1511.02570*, 2015.

[66] M. Kavalec, A. Maedche, and V. Svátek, "Discovery of lexical entries for non-taxonomic relations in ontology learning," in *International Conference on Current Trends in Theory and Practice of Computer Science*, pp. 249–256, Springer, 2004.

[67] X. Wang, Y. Chen, J. Yang, L. Wu, Z. Wu, and X. Xie, "A reinforcement learning framework for explainable recommendation," in *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 587–596, IEEE, 2018.

[68] M.-m. Poo, J.-l. Du, N. Y. Ip, Z.-Q. Xiong, B. Xu, and T. Tan, "China brain project: basic neuroscience, brain diseases, and brain-inspired computing," *Neuron*, vol. 92, no. 3, pp. 591–596, 2016.

[69] Y. Lan, S. Wang, and J. Jiang, "Knowledge base question answering with a matching-aggregation model and question-specific contextual relations," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 10, pp. 1629–1638, 2019.

[70] A. Sayed and A. Al Muqrishi, "Ibri-casonto: Ontology-based semantic search engine," *Egyptian Informatics Journal*, vol. 18, no. 3, pp. 181–192, 2017.

[71] J. W. Murdock and G. Tesauro, "Statistical approaches to question answering in watson," *Mathematics Awareness Month Theme Essay*, 2012.

[72] C.-H. Lee, Y.-H. Wang, and A. J. Trappey, "Ontology-based reasoning for the intelligent handling of customer complaints," *Computers & Industrial Engineering*, vol. 84, pp. 144–155, 2015.

[73] Y. Lan and J. Jiang, "Query graph generation for answering multi-hop complex questions from knowledge bases," Association for Computational Linguistics, 2020.

[74] J. Sun, Z. Xing, R. Chu, H. Bai, J. Wang, and X. Peng, "Know-how in programming tasks: From textual tutorials to task-oriented knowledge graph," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 257–268, IEEE, 2019.

[75] B. Xu, Z. Xing, X. Xia, and D. Lo, "Answerbot: Automated generation of answer summary to developers' technical questions," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 706–716, IEEE, 2017.

[76] L. Cai, H. Wang, B. Xu, Q. Huang, X. Xia, D. Lo, and Z. Xing, "Answerbot: an answer summary generation tool based on stack overflow," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1134–1138, 2019.

[77] H. Li, I. Councill, W.-C. Lee, and C. L. Giles, "Citeseerx: an architecture and web service design for an academic document search engine," in *Proceedings of the 15th international conference on World Wide Web*, pp. 883–884, ACM, 2006.

[78] P. Szekely, C. A. Knoblock, J. Slepicka, A. Philpot, A. Singh, C. Yin, D. Kapoor, P. Natarajan, D. Marcu, K. Knight, *et al.*, "Building and using a knowledge graph to combat human trafficking," in *International Semantic Web Conference*, pp. 205–221, Springer, 2015.

[79] J. Pujara, H. Miao, L. Getoor, and W. Cohen, "Knowledge graph identification," in *International Semantic Web Conference*, pp. 542–557, Springer, 2013.

[80] F. Niu, C. Zhang, C. Ré, and J. W. Shavlik, "Deepdive: Web-scale knowledge-base construction using statistical learning and inference.," *VLDS*, vol. 12, pp. 25–28, 2012.

[81] C. Z. Mooney, R. D. Duval, and R. Duval, *Bootstrapping: A nonparametric approach to statistical inference*. No. 94-95, Sage, 1993.

[82] T. H. Nguyen and R. Grishman, "Relation extraction: Perspective from convolutional neural networks," in *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, pp. 39–48, 2015.

[83] M. Miwa and M. Bansal, "End-to-end relation extraction using lstms on sequences and tree structures," *arXiv preprint arXiv:1601.00770*, 2016.

[84] C. Alt, M. Hübner, and L. Hennig, "Improving relation extraction by pre-trained language representations," *arXiv preprint arXiv:1906.03088*, 2019.

[85] H. Yu, Y. Cao, G. Cheng, P. Xie, Y. Yang, and P. Yu, "Relation extraction with bert-based pre-trained model," in *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pp. 1382–1387, IEEE, 2020.

[86] S. L. Abebe and P. Tonella, "Towards the extraction of domain concepts from the identifiers," in *2011 18th Working Conference on Reverse Engineering*, pp. 77–86, IEEE, 2011.

[87] M. Schmitz, R. Bart, S. Soderland, O. Etzioni, *et al.*, "Open language learning for information extraction," in *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 523–534, Association for Computational Linguistics, 2012.

[88] J. Fan, D. Ferrucci, D. Gondek, and A. Kalyanpur, "Prismatic: Inducing knowledge from a large scale lexicalized relation resource," in *Proceedings of the NAACL HLT 2010 first international workshop on formalisms and methodology for learning by reading*, pp. 122–127, Association for Computational Linguistics, 2010.

[89] T. Falke, G. Stanovsky, I. Gurevych, and I. Dagan, "Porting an open information extraction system from english to german," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 892–898, 2016.

[90] M. Cetto, C. Niklaus, A. Freitas, and S. Handschuh, "Graphene: Semantically-linked propositions in open information extraction," *arXiv preprint arXiv:1807.11276*, 2018.

[91] A. Fader, S. Soderland, and O. Etzioni, "Identifying relations for open information extraction," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 1535–1545, Association for Computational Linguistics, 2011.

[92] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: a core of semantic knowledge," in *Proceedings of the 16th international conference on World Wide Web*, pp. 697–706, ACM, 2007.

[93] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum, "Yago2: A spatially and temporally enhanced knowledge base from wikipedia," *Artificial Intelligence*, vol. 194, pp. 28–61, 2013.

[94] R. Padhye, D. Mukherjee, and V. S. Sinha, "Api as a social glue," in *Companion Proceedings of the 36th International Conference on Software Engineering*, pp. 516–519, ACM, 2014.

[95] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live api documentation," in *Proceedings of the 36th International Conference on Software Engineering*, pp. 643–652, ACM, 2014.

[96] M. W. Matlin and H. J. Foley, *Sensation and perception*. Allyn & Bacon, 1992.

[97] J. Lisman, "The challenge of understanding the brain: where we stand in 2015," *Neuron*, vol. 86, no. 4, pp. 864–882, 2015.

[98] S. Chen, S. Zhang, J. Shang, B. Chen, and N. Zheng, "Brain-inspired cognitive model with attention for self-driving cars," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 11, no. 1, pp. 13–25, 2017.

[99] V. Tresp, Y. Ma, and S. Baier, "Learning with knowledge graphs.," in *NeSy*, 2017.

[100] F. I. Craik and J. M. Jennings, "Human memory.," 1992.

[101] G. Sperling, "A model for visual memory tasks," *Human factors*, vol. 5, no. 1, pp. 19–31, 1963.

[102] A. Baddeley, "Working memory," *Science*, vol. 255, no. 5044, pp. 556–559, 1992.

[103] M. Gazzaniga and R. B. Ivry, *Cognitive Neuroscience: The Biology of the Mind: Fourth International Student Edition*. WW Norton, 2013.

[104] A. D. Baddeley, "The influence of acoustic and semantic similarity on long-term memory for word sequences," *The Quarterly journal of experimental psychology*, vol. 18, no. 4, pp. 302–309, 1966.

[105] P. Goelet, V. F. Castellucci, S. Schacher, and E. R. Kandel, "The long and the short of long-term memoryŮa molecular framework," *Nature*, vol. 322, no. 6078, p. 419, 1986.

[106] G. K. Aguirre and M. D'Esposito, "Topographical disorientation: a synthesis and taxonomy," *Brain*, vol. 122, no. 9, pp. 1613–1628, 1999.

[107] R. A. McCarthy, J. J. Evans, and J. R. Hodges, "Topographic amnesia: spatial memory disorder, perceptual dysfunction, or category specific semantic memory impairment?," *Journal of Neurology, Neurosurgery & Psychiatry*, vol. 60, no. 3, pp. 318–325, 1996.

[108] E. Tulving and D. L. Schacter, "Priming and human memory systems," *Science*, vol. 247, no. 4940, pp. 301–306, 1990.

[109] M. Eysenck, *Attention and arousal: Cognition and performance*. Springer Science & Business Media, 2012.

[110] P. Graf and D. L. Schacter, "Implicit and explicit memory for new associations in normal and amnesic subjects.," *Journal of Experimental Psychology: Learning, memory, and cognition*, vol. 11, no. 3, p. 501, 1985.

[111] K. K. Szpunar, "Episodic future thought: An emerging concept," *Perspectives on Psychological Science*, vol. 5, no. 2, pp. 142–162, 2010.

[112] E. Tulving *et al.*, "Episodic and semantic memory," *Organization of memory*, vol. 1, pp. 381–403, 1972.

[113] A. M. Collins and M. R. Quillian, "Retrieval time from semantic memory," *Journal of verbal learning and verbal behavior*, vol. 8, no. 2, pp. 240–247, 1969.

[114] A. Jaya and G. Uma, "A knowledge based approach for semantic reasoning of stories using ontology," in *Thinkquest˜2010*, pp. 310–315, Springer, 2011.

[115] H. Tang and W. Huang, "Brain inspired cognitive system for learning and memory," in *International Conference on Neural Information Processing*, pp. 477–484, Springer, 2011.

[116] G. Singer, U. Norbisrath, and D. Lewandowski, "Ordinary search engine users carrying out complex search tasks," *Journal of Information Science*, vol. 39, no. 3, pp. 346–358, 2013.

[117] J. Cho and S. Roy, "Impact of search engines on page popularity," in *Proceedings of the 13th international conference on World Wide Web*, pp. 20–29, ACM, 2004.

[118] X. Zhao, H. Li, Y. Tang, D. Gao, L. Bao, and C.-H. Lee, "A smart context-aware program assistant based on dynamic programming event modeling," in *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 24–29, IEEE, 2018.

[119] H. Li, X. Zhao, Z. Xing, L. Bao, X. Peng, D. Gao, and W. Zhao, "amassist: In-ide ambient search of online programming resources," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pp. 390–398, IEEE, 2015.

[120] B. Sarrafzadeh, O. Vechtomova, and V. Jokic, "Exploring knowledge graphs for exploratory search," in *Proceedings of the 5th Information Interaction in Context Symposium*, pp. 135–144, ACM, 2014.

[121] C. N. Carlson, "Information overload, retrieval strategies and internet user empowerment," 2003.

[122] J. Zhu, B. Shen, X. Cai, and H. Wang, "Building a large-scale software programming taxonomy from stackoverflow," in *SEKE 2015: 27th International Conference on Software Engineering and Knowledge Engineering*, pp. 391–396.

[123] F. Joel, "Google: Our new search strategy is to compute answers, not links." https://thenextweb.com/google/2011/06/01/google-our-new-search-strategy-is-to-compute-answers-not-links/. Accessed June 1, 2011.

[124] G. Seth, "Breakthrough analysis: Search is not the answer." https://www.informationweek.com/software/information-management/breakthrough-analysis-search-is-not-the-answer/d/d-id/1046118. Accessed August 14, 2006.

[125] G. Liz, "Breakthrough analysis: Search is not the answer." http://allthingsd.com/20130314/how-search-is-evolving-finally-beyond-caveman-queries/. Accessed March 14, 2013.

[126] G. Seth, "How search is evolving ?finally! ?beyond caveman queries." https://techcrunch.com/2011/05/07/search-answers-not-just-links/. Accessed August 14, 2006.

[127] Y. Liu, X. Yi, R. Chen, and Y. Song, "A survey on frameworks and methods of question answering," in *2016 3rd International Conference on Information Science and Control Engineering (ICISCE)*, pp. 115–119, IEEE, 2016.

[128] X. Zhao, Z. Xing, M. A. Kabir, N. Sawada, J. Li, and S.-W. Lin, "Hdskg: Harvesting domain specific knowledge graph from content of webpages," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 56–67, IEEE, 2017.

[129] X. Zhao, "Hdso: Harvest domain specific non-taxonomic relations of ontology from internet by deep neural networks (dnn)," *BSR winter school Big Software on the Run: Where Software meets Data*, p. 74.

[130] J. R. Finkel, T. Grenager, and C. D. Manning, "Incorporating non-local information into information extraction systems by gibbs sampling," in *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACLąŕ05)*, pp. 363–370, 2005.

[131] H. Li, S. Li, J. Sun, Z. Xing, X. Peng, M. Liu, and X. Zhao, "Improving api caveats accessibility by mining api caveats knowledge graph," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 183–193, IEEE, 2018.

[132] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit.," in *ACL (System Demonstrations)*, pp. 55–60, 2014.

[133] K. Toutanova and C. D. Manning, "Enriching the knowledge sources used in a maximum entropy part-of-speech tagger," in *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*, pp. 63–70, Association for Computational Linguistics, 2000.

[134] J. Nivre, M.-C. de Marneffe, F. Ginter, Y. Goldberg, J. Hajic, C. D. Manning, R. McDonald, S. Petrov, S. Pyysalo, N. Silveira, *et al.*, "Universal dependencies v1: A multilingual treebank collection," in *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*, 2016.

[135] S. Schuster and C. D. Manning, "Enhanced english universal dependencies: An improved representation for natural language understanding tasks," in *Proceedings of the 10th International Conference on Language Resources and Evaluation*, 2016.

[136] C. Chen, Z. Xing, and L. Han, "Techland: Assisting technology landscape inquiries with insights from stack overflow," *32nd ICSME. IEEE*, 2016.

[137] J. Hirschberg and C. D. Manning, "Advances in natural language processing," *Science*, vol. 349, no. 6245, pp. 261–266, 2015.

[138] A. M. Rush, S. Chopra, and J. Weston, "A neural attention model for abstractive sentence summarization," *arXiv preprint arXiv:1509.00685*, 2015.

[139] H. He, K. Gimpel, and J. Lin, "Multi-perspective sentence similarity modeling with convolutional neural networks," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1576–1586, 2015.

[140] M.-C. De Marneffe and C. D. Manning, "Stanford typed dependencies manual," tech. rep., Technical report, Stanford University, 2008.

[141] M.-C. De Marneffe, T. Dozat, N. Silveira, K. Haverinen, F. Ginter, J. Nivre, and C. D. Manning, "Universal stanford dependencies: A cross-linguistic typology.," in *LREC*, vol. 14, pp. 4585–92, 2014.

[142] C. Grover and R. Tobin, "Rule-based chunking and reusability," in *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC 2006)*, 2006.

[143] T. Zhang, F. Damerau, and D. Johnson, "Text chunking based on a generalization of winnow," *Journal of Machine Learning Research*, vol. 2, no. Mar, pp. 615–637, 2002.

[144] E. Loper and S. Bird, "Nltk: the natural language toolkit," *arXiv preprint cs/0205028*, 2002.

[145] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li, "Efficient subgraph matching on billion node graphs," *Proceedings of the VLDB Endowment*, vol. 5, no. 9, pp. 788–799, 2012.

[146] B. Marshall, H. Chen, and T. Madhusudan, "Matching knowledge elements in concept maps using a similarity flooding algorithm," *Decision Support Systems*, vol. 42, no. 3, pp. 1290–1306, 2006.

[147] P. M. Fitts, "Cognitive aspects of information processing: Iii. set for speed versus accuracy.," *Journal of experimental psychology*, vol. 71, no. 6, p. 849, 1966.

[148] Z. S. Harris, "Distributional structure," *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.

[149] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[150] X. Zhao, C. Miao, and Z. Xing, "Identifying cognitive distortion by convolutional neural network based text classification," 2017.

[151] B. Xu, D. Ye, Z. Xing, X. Xia, G. Chen, and S. Li, "Predicting semantically linkable knowledge in developer online forums via convolutional neural network," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pp. 51–62, ACM, 2016.

[152] G. Nagy, "Pattern classification and scene analysis-duda, ro and hart, pe," 1973.

[153] S. Rässler, *Statistical matching: A frequentist theory, practical applications, and alternative Bayesian approaches*, vol. 168. Springer Science & Business Media, 2012.

[154] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Association for Computational Linguistics (ACL) System Demonstrations*, pp. 55–60, 2014.

[155] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, (Valletta, Malta), pp. 45–50, ELRA, May 2010.

[156] S. Bird, "Nltk: the natural language toolkit," in *Proceedings of the COLING/ACL on Interactive presentation sessions*, pp. 69–72, Association for Computational Linguistics, 2006.

[157] J. Webber, "A programmatic introduction to neo4j," in *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*, pp. 217–218, ACM, 2012.

[158] M. David, "Powers. 2011. evaluation: From precision, recall and f-score to roc, informedness, markedness & correlation," *Journal of Machine Learning Technologies*, vol. 2, no. 1.

[159] J. Brooke *et al.*, "Sus-a quick and dirty usability scale," *Usability evaluation in industry*, vol. 189, no. 194, pp. 4–7, 1996.

[160] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: a collaboratively created graph database for structuring human knowledge," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1247–1250, ACM, 2008.

[161] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data," in *The semantic web*, pp. 722–735, Springer, 2007.

[162] J. Zhu, Z. Nie, X. Liu, B. Zhang, and J.-R. Wen, "Statsnowball: a statistical approach to extracting entity relationships," in *Proceedings of the 18th international conference on World wide web*, pp. 101–110, ACM, 2009.

[163] F. Wu and D. S. Weld, "Open information extraction using wikipedia," in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pp. 118–127, Association for Computational Linguistics, 2010.

[164] O. Etzioni, M. Banko, S. Soderland, and D. S. Weld, "Open information extraction from the web," *Communications of the ACM*, vol. 51, no. 12, pp. 68–74, 2008.

[165] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion.," in *AAAI*, pp. 2181–2187, 2015.

[166] J. C. de Almeida Biolchini, P. G. Mian, A. C. C. Natali, T. U. Conte, and G. H. Travassos, "Scientific research ontology to support systematic review in software engineering," *Advanced Engineering Informatics*, vol. 21, no. 2, pp. 133–151, 2007.

[167] P. Wongthongtham, E. Chang, T. Dillon, and I. Sommerville, "Development of a software engineering ontology for multisite software development," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 8, pp. 1205–1217, 2009.

[168] H. S. Delugach, "Specifying multiple-viewed software requirements with conceptual graphs," *Journal of Systems and Software*, vol. 19, no. 3, pp. 207–224, 1992.

[169] H.-J. Happel and S. Seedorf, "Applications of ontologies in software engineering," in *Proc. of Workshop on Sematic Web Enabled Software Engineering"(SWESE) on the ISWC*, pp. 5–9, Citeseer, 2006.

[170] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni, "Open information extraction from the web.," in *IJCAI*, vol. 7, pp. 2670–2676, 2007.

[171] J. Betteridge, A. Carlson, S. A. Hong, E. R. Hruschka Jr, E. L. Law, T. M. Mitchell, and S. H. Wang, "Toward never ending language learning.," in *AAAI Spring Symposium: Learning by Reading and Learning to Read*, pp. 1–2, 2009.

[172] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates, "Web-scale information extraction in knowitall:(preliminary results)," in *Proceedings of the 13th international conference on World Wide Web*, pp. 100–110, ACM, 2004.

[173] G. Kasneci, M. Ramanath, F. Suchanek, and G. Weikum, "The yago-naga approach to knowledge discovery," *ACM SIGMOD Record*, vol. 37, no. 4, pp. 41–47, 2009.

[174] R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, S. Vaithyanathan, and H. Zhu, "Systemt: a system for declarative information extraction," *ACM SIGMOD Record*, vol. 37, no. 4, pp. 7–13, 2009.

[175] N. Nakashole, M. Theobald, and G. Weikum, "Scalable knowledge harvesting with high precision and high recall," in *Proceedings of the fourth ACM international conference on Web search and data mining*, pp. 227–236, ACM, 2011.

[176] J. Zhu, H. Wang, and B. Shen, "Software. zhishi. schema: A software programming taxonomy derived from stackoverflow,"

[177] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? an analysis of topics and trends in stack overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.

[178] C. Chen and Z. Xing, "Mining technology landscape from stack overflow," *10th ESEM. IEEE/ACM*, 2016.

[179] X. Chen, C.-H. Chen, K. F. Leong, and X. Jiang, "An ontology learning system for customer needs representation in product development," *The International Journal of Advanced Manufacturing Technology*, vol. 67, no. 1-4, pp. 441–453, 2013.

[180] X. Jiang and A.-H. Tan, "Crctol: A semantic-based domain ontology learning system," *Journal of the American Society for Information Science and Technology*, vol. 61, no. 1, pp. 150–168, 2010.

[181] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of massive datasets*. Cambridge University Press, 2014.

[182] J. Ramos, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*, 2003.

[183] P. Velardi, M. Missikoff, and R. Basili, "Identification of relevant terms to support the construction of domain ontologies," in *Proceedings of the workshop on Human Language Technology and Knowledge Management-Volume 2001*, p. 5, Association for Computational Linguistics, 2001.

[184] N. Voskarides, E. Meij, M. Tsagkias, M. de Rijke, and W. Weerkamp, "Learning to explain entity relationships in knowledge graphs," *In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and The 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing (ACL-IJCNLP 2015)*, p. 11, 2015.

[185] Y. WHAN KIM and J. H. Kim, "A model of knowledge based information retrieval with hierarchical concept graph," *Journal of Documentation*, vol. 46, no. 2, pp. 113–136, 1990.

[186] M. Joshi, U. Sawant, and S. Chakrabarti, "Knowledge graph and corpus driven segmentation and answer inference for telegraphic entity-seeking queries.," in *EMNLP*, pp. 1104–1114, 2014.

[187] M. A. Hearst, "Automatic acquisition of hyponyms from large text corpora," in *Proceedings of the 14th conference on Computational linguistics-Volume 2*, pp. 539–545, Association for Computational Linguistics, 1992.

[188] Y. Yao, D. Ye, P. Li, X. Han, Y. Lin, Z. Liu, Z. Liu, L. Huang, J. Zhou, and M. Sun, "Docred: A large-scale document-level relation extraction dataset," *arXiv preprint arXiv:1906.06127*, 2019.

[189] H. Kim, "Digital humanities research administration of archives and humanities.." http://dh.aks.ac.kr/Encyves/wiki/. 2017.

[190] D. Sorokin and I. Gurevych, "Context-aware representations for knowledge base relation extraction," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 1784–1789, 2017.