Brain-Inspired Search Engine Assistant Based on Knowledge Graph

Xuejiao Zhao†, Huanhuan Chen‡, Senior Member, IEEE, Zhenchang Xing§, and Chunyan Miao¶, Senior Member, IEEE

Abstract—Search engines can quickly respond to a hyperlink list according to query keywords. However, when a query is complex, developers need to repeatedly refine search keywords and open a large number of web pages to find and summarize answers. Many research works of question and answering (Q&A) system attempt to assist search engines by providing simple, accurate, and detailed answers. However, without original semantic contexts, these answers lack explainability, making them difficult for users to trust and adopt. In this article, a brain-inspired search engine assistant named DeveloperBot based on knowledge graph is proposed, which aligns to the cognitive process of humans and has the capacity to answer complex queries with explainability. Specifically, DeveloperBot first constructs a multilayer query graph by splitting a complex multiconstraint query into several ordered constraints. Then, it models a constraint reasoning process as a subgraph search process inspired by a spreading activation model of cognitive science. In the end, novel features of the subgraph are extracted for decision-making. The corresponding reasoning subgraph and answer confidence are derived as explanations. The results of the decision-making demonstrate that DeveloperBot can estimate answers and answer confidences with high accuracy. We implement a prototype and conduct a user study to evaluate whether and how the direct answers and the explanations provided by DeveloperBot can assist developers’ information needs.

Index Terms—Brain-inspired system, cognitive process, knowledge graph, question and answering (Q&A) system.

I. INTRODUCTION

TRADITIONAL search engines (e.g., Google and Bing) provide a list of websites in which the needed information may be found for developers. Recently, the community is increasingly recognizing that traditional retrieval models are insufficient to satisfy complex information needs and propose to provide additional enhanced services for complex search tasks depending on their characteristics [1]–[3]. This is a consensus for major search engine companies (e.g., Google, Bing, and Yahoo): automatic question and answering (Q&A) system (also known as a direct answer search engine, or natural language search engine and so on) is a more advanced next-generation search engine that returns simple, direct and real-time answers instead of a ranked list of hyperlinks [4]–[8]. However, without the original semantic context, the answers generated by existing Q&A systems lack explainability that makes the answers difficult for users to trust and adopt [9]–[12].

Knowledge graphs (KGs) are semantic networks that contain a large number of concepts and relations, which makes explainable Q&A system possible [13], [14]. To further provide a human-centered explanation, artificial intelligence (AI) system should align with the cognitive model of humans and explain within the basic framework of human cognition [15]–[20]. However, the existing Q&A systems lack a unified framework of the Q&A cognitive process based on knowledge graphs. To bridge this gap and equip Q&A system with human-like cognitive capabilities, the contents of cognitive science related to Q&A are explored [21]–[24].

These research works show that the cognitive process of Q&A consists of the following five steps including perception, planning, reasoning, response, and learning. Depending on these inspirations, a brain-inspired search engine assistant called DeveloperBot is presented in this article. DeveloperBot contains five modules aligned with the human cognitive process of Q&A. Its framework can be used as a basis for designing a knowledge graph-based Q&A system that can understand answer questions and provide a human-centered explanation.

In order to understand the syntax of the technical questions of developers, a rough observation and analysis are made on the closed-ended questions of Stack Overflow. Closed-ended questions refer to questions that could be answered with a simple response, e.g., one-word answer. The results show

---

[1] H. Chen, “A survey on natural language processing,” in Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min., 2011, pp. 1–9.
[2] S. Aggarwal and V. R. Lesser, “A survey on natural language processing,” in Proc. IEEE Int. Conf. Comput. Speech Lang. Process., 2012, pp. 222–236.
[3] M. J. Dredze, R. Socher, and A.PT. Fergus, “A survey on natural language processing,” in Proc. Int. Conf. Mach. Learn., 2014, pp. 1–9.
[4] D. R. Koller and M. J. Hebert, “A survey on knowledge graphs,” in Proc. Int. Conf. Mach. Learn., 2014, pp. 1–9.
[5] J. C. Platt, “A survey on knowledge graphs,” in Proc. Int. Conf. Mach. Learn., 2014, pp. 1–9.
[6] Y. Bengio, “A survey on knowledge graphs,” in Proc. Int. Conf. Mach. Learn., 2014, pp. 1–9.
[7] Y. Bengio, “A survey on knowledge graphs,” in Proc. Int. Conf. Mach. Learn., 2014, pp. 1–9.
[8] Y. Bengio, “A survey on knowledge graphs,” in Proc. Int. Conf. Mach. Learn., 2014, pp. 1–9.

See https://www.ieee.org/publications/rights/index.html for more information.
that these questions are often complex and diverse, and the constraints of a query are often scattered in various grammars, such as attributive clauses and coordinate clauses. Here, the constraint consists of three basic elements represented as category constraint, predicate relation of category constraint and property constraint, and property constraint, and it describes the information, such as category and property, of an answer. However, existing query representation algorithms are insufficient for complex multiconstraint query\(^1\) representation and solving, and the features, such as graph topological structure and indirect relations, are not fully utilized in answer reasoning.

To address these issues, two modules of the DeveloperBot called BotPerception and BotPlanning implement a novel query representation algorithm. Specifically, the BotPerception module incorporates the dependency parsing into the tree parsing [25] to maximize the completeness of the entities and relations extraction. Meanwhile, the BotPlanning module splits a multiconstraint query into several simple constraints and determines their solving order and then constructs these ordered constraints into a multilayer query graph for further usage.

Then, a module of the DeveloperBot called BotReasoning is further proposed to address the complex query solving problem. This algorithm uses the query graph to search for a candidate subgraph from the knowledge graph by spreading activation algorithm inspired by cognitive science [24]. Then, the BotReasoning extracts the candidate answers and corresponding features according to direct relation, indirect relation, topological structure of candidate subgraph, and so on. Next, these features are integrated into the decision-making algorithms such as deep neural network (DNN) to determine the final answers. In the end, a reasoning subgraph and an answer confidence are extracted following the cognitive process as qualitative and quantitative explanations to explain “why,” “how,” and “how confident” an answer is being presented.

The experimental results show that, with the novel features of the subgraph, the DNN can extract answers with higher accuracy and estimate answer confidences with lower mean square error (MSE). We also implement a prototype of DeveloperBot and customize it by loading a knowledge graph of the software engineering domain into its knowledge base. The results of the user study involving 24 participants 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 explanations 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, for relatively complex queries, with the assistance of DeveloperBot, the search performance improvement of the developers is more significant.

This article makes the following four major contributions.

1) A novel brain-inspired search engine assistant named DeveloperBot is proposed, which aligns with the cognitive process of human and has the capacity to answer complex queries with human-centered explainability based on knowledge graph.

2) A query representation algorithm implemented by the BotPerception and the BotPlanning modules is proposed, which incorporates advantages of both the dependency parsing and the tree parsing to maximize the completeness of constraint extraction and determine their solving order, which enhances the representation capacity of the complex multiconstraint query of Q&A system.

3) An algorithm named BotReasoning is further proposed for answer reasoning and explanations generation. It inspires from the spreading activation model and models the constraint reasoning process as the candidate subgraph search and decision-making process. Based on 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 the software engineering domain as a proof of concept. A user study is also conducted to evaluate its practical values.

II. RELATED WORKS

KG has been widely used in Q&A system, recommendation system, and search engines in recent years due to its excellent knowledge representation capacity. Many research works on recommendation system have taken advantage of the explainability of knowledge graph to improve the users’ trust and adoption prominently [26]–[29]. Some of these techniques produce a ranked list of entities using a personalized PageRank procedure as explanations [30]. Some others of them provide human-readable explanations by summarizing common multi-hop relational patterns of knowledge graph for inferring different item associations [29]. Zhang and Chen [27] summarized explainable recommendation applications in different domains, i.e., e-commerce, point of interest, social, and multimedia. Ai et al. [31] presented an Indri system that is an explainable recommendation Q&A system to support modern language technologies. Catherine, Rose et al. [30] and Yu et al. [32] proposed 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 explainability of the knowledge graph. The content of the explanations of this article is partially inspired by existing research works on the recommendation system.

The recent research works of Wang et al. [33] and Zhu et al. [34] are the two closest works to ours. Both their works and ours propose to assist the information needs of

\(^1\) Here is an example of a complex multiconstraint query: “which graph databases are compatible with Linux and support Python?” in which the “graph_database” is a subject constraint indicated the category of the answers. The “Linux” and “python” are object constraints that indicated the property of the answers. The predicate “are_compatible_with” and “support” indicates the relation of the subject constraint and the object constraints. We define a question that requires multiple reasoning to solve multiple constraints as a complex multiconstraint query.
users expressed in natural language by knowledge graph-based Q&A system. However, their works extract the entities of a query by simple token matching [35] or entity linking [33], [34]. Lan and Jiang [36] proposed to incorporate constraints and extend relation paths to handle questions with constraints and questions with multiple hops of relations at the same time. Lee et al. [37] solved the customer complaint handling query by a basic ontology-based searching method. Xie [38] represented the knowledge graph extracted from Chinese medical records by the Node2vec algorithm and then searches answers by k-nearest neighbors (KNNs). In contrast, our model analyzes the expression pattern of natural language deeply and constructs a query into a multilayer query graph for further solving, which enhances the representation capacity of the complex multiconstraint query. Next, although all of these works extract an inference subgraph for further reasoning, this article extracts the subgraph by spread activation model inspired from cognitive science [24], which can extract the candidate answers and corresponding subgraph more comprehensively. This article also considers more comprehensive features of the subgraph, such as indirect relation, topological structure, predicate, and similarity, and some of the features are inspired by previous works. Furthermore, this article integrates all the features into a decision-making algorithm to extract correct answers with high accuracy. This also allows our work to quantify the confidence of correct answers. In addition, our work presents the reasoning subgraph and the confidence as explanations of correct answers, and according to the results of the experiments, this significantly improves answer adoption and user confidence. As far as we know, using a Q&A system with explainability based on knowledge graph as a search engine assistant has not been attempted before.

III. COGNITIVE FRAMEWORK OF DEVELOPERBOT

In the human brain, there are various memory types (e.g., sensory memory and episodic memory) that store different contents. One of them called semantic memory stores the general knowledge of the world, concepts, and rules in the form of connected concept nodes. As shown on the left of Fig. 1 [39], there are two kinds of relations. One is hierarchical relations, which indicates the concepts organizing from higher to lower order categories such as “Animal” and “Bird.” Another relation is called concept–property relation, which stores the properties of concepts such as “Salmon” and “lives in the ocean” [39].

Actually, the knowledge stored structure of semantic memory is similar to the structure of the knowledge graph as shown on the right of Fig. 1 [40]–[42]. The concepts are represented as entities, and the hierarchical relation is the relation between two entities that connect by a predicate phrase “is_a.” The concept–property relation is represented as relation triples (subject, predicate, and object), where subject and object are entities, and the predicate is the edge from subject to object [43].

Question reasoning is an advanced cognitive process of accessing the semantic memory to look for answers according to some premises [21], [44]–[46], as shown in the upper half of Fig. 2. Assuming a query as a stimulus from the external environment, we summarize several key steps related to the cognitive process of query reasoning as follows: 1) perception: to encode and explain external stimuli (query) as signals that the brain can recognize [22]; 2) planning: to divide a task into smaller, more manageable parts and decide the right executing order [23]; 3) reasoning: to access the semantic memory for the answer [24]; 4) response: to output the answers and explanations from the mouth, expression, or body language [47]; and 5) learning: to learn new knowledge by cognitive activities.

The lower half in Fig. 2 is the cognitive framework of DeveloperBot. It shows that how DeveloperBot emulates and aligns the cognitive process of the human brain. The BotPerception emulates the perception process for representing the stimuli of the human brain. The BotPlanning algorithm splits a query into smaller constraints and decides their solving order which is similar to the planning of the cognitive process. The BotReasoning is an answer reasoning process based on knowledge graph mimicking human reasoning [24]. The user interface (UI) called BotResponse is similar to the response of the cognitive process. BotLearning is a knowledge graph construction algorithm. Based on this brain-inspired framework, human-centered explanations can be generated to show how a query is represented, planned, and reasoned.

IV. APPROACH

A. Overview of DeveloperBot

DeveloperBot contains four main parts as shown in the overview of Fig. 3. The input is the query text from...
developers (e.g., “Who develops java?”), and there are three outputs.

1) **Direct Answers:** The direct answers derived by our system.

2) **Reasoning Subgraph:** A reasoning subgraph to explain the reasons to present each direct answer.

3) **Confidence:** The confidence of every direct answer.

In DeveloperBot, the BotResponse module is a UI that consists of a search box to input query text and an area to visualize direct answers and explanations. BotPerception and BotPlanning represent a multiconstraint query as a multilayer query graph. These two modules receive the query text from the UI, preprocesses the query text, add natural language processing (NLP) markup such as token, part of speech (POS) tag, and dependency, chunk NP and VP, chunk query constraint triples, determine the solving order, and construct a multilayer query graph.

Then, the BotReasoning module is responsible for reasoning answers of constraint quads of the query graph from outer layer to inner layers. BotReasoning repeats the iteration operation until the last constraint quads are solved and then extracts the direct answers and their explanations to the UI.

BotLearning constructs a knowledge graph of software engineering domain offline that combines the knowledge graph from an automatical knowledge graph construction algorithm called HDSKG and expertise of the experts in software engineering domain, and the details can be found in [48].

**B. BotPerception and BotPlanning**

In this section, we elaborate on how BotPerception and BotPlanning combine dependency into constituency-based parse trees to extract, order the constraints in the query, and construct them into a multilayer query graph.

1) **Annotate Query With Multidimensional NLP Markup:** DeveloperBot incorporates the tool named coreNLP to do the tokenization, POS tagging, dependency parsing, and lemmatization to the query [49]–[52]. The task of tokenization is to break the sentence into small pieces, called tokens, and drop some characters such as punctuation. POS tagging is the process of labeling words with their grammatical properties, such as NN: noun, singular or mass, WP: Whpronoun, VBZ: verb, and third-person singular present. Lemmatization can reduce inflectional forms and derivative related forms of a word to a common base form.

**Dependency**

| Dependency | Governor | Dependent | Semantic relationship between the words depicted by dependency |
|------------|----------|-----------|---------------------------------------------------------------|
| det        | databases-3 | Which-1 | “Which-1” is determiner of “databases-3”                    |
| subj       | support-4  | databases-3 | “databases-3” is nominal subject of “support-4” |
| dobj       | support-4  | Python-5  | “Python-5” is direct object of “support-4”                    |
| nmod       | accessed-9 | languages-14 | “languages-14” is nominal modifier of “accessed-9”         |
| conj       | support-4  | accessed-9 | “support-4” and “accessed-9” are connected by coordinating conjunction “and” |
A higher layer number means that the layer number refers to the solving order of the constraint quads, which consists of four basic elements represented as QG. A QG is composed of the basic units called constituent extraction?"). As shown in Table I, “det” is the abbreviation of determiner. There are three interrogative determiners: what, which, and whose. “nsubj” is an NP and acts as a passive syntactic subject of a clause [53]. For our sample query, nsubj (support-4, databases-3) means that “databases-3” is the syntactic subject of the verb “support-4.” In general, the “nmod” indicates some further adjunct relation specified by the case.

2) Constituency-Based Tree Parsing: In this step, a full-text parsing technique called “tree parsing” is adapted to segment a sentence into its subconstituents [34], [54]–[56]. As shown in Table II, DeveloperBot sets four subconstituents called WHNP, WHVP, NP, and VP. Here, NP and VP are abbreviations of the noun phrase and verb phrase, respectively. Subconstituent WHNP represents a phrase beginning with “which,” “what,” or “whose” followed by a noun phrase (e.g., which, what, or whose). “nsubj” is an NP and acts as a passive subjunctive. There are three interrogative determiners: what, which, and whose. “nsubj” is an NP and acts as a passive subjunctive. Here, “support-4” is vp11, which means the total number of words of them is x, y, u, and v, respectively.

Second, the dependency parsing technique is used to add dependency markups to all words of Q. After in-depth observation and analysis of the coreference extraction patterns, we summarize the query sentence structures as shown in Fig. 4. Therefore, (graph_databases) at the beginning of the query phrase and “graph databases” is accessed-9 languages-14 at the end of the query phrase.

Pattern 1: If in a query Q, m = 1 and n = 0, WHNP is detected at the beginning of Q. The NP following the interrogative word called WHNP is regarded as a category constraint of direct answers. Assume that whnp is the e-th word of WHNP, if the following four dependency pairs are detected simultaneously in the result of dependency parsing of Q, then, a property constraint (VP, NP) is extracted into the QG.

where whnp, whvp, and vpst are the /th, /th, and /th words of WHNP, WHVP, and VP, respectively.

For the sample query, “which graph databases” is a WHNP at the beginning of the query phrase and “graph databases” is the WHNP, as shown in Fig. 4. Therefore, (graph_databases) is extracted as a category constraint of direct answers. Then, two dependency pairs dp[nsubj (support-4, databases-3) and dobj(support-4, Python-5)] and dp[nsubjpass(accessed-9, databases-3) and nmod(accessed-9, languages-14)] are detected. Here, “support-4” is vp11, which means the first word in the first VP of Q. The “databases-3” is whnp 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 innermost layer of QG. Therefore, from the sample query, Pattern 1 can extract two constraint quads: \( (\text{graph_databases}, \text{support}, \text{Python}, 1) \) and \( (\text{graph_databases}, \text{can_be_accessed_through}, \text{RDF_query_languages}, 1) \).

**Pattern 2:** Assuming that the “property constraints” of a constraint quads is \( PC = (p_{c1}, p_{c2}, \ldots, p_{cg}, \ldots, p_{cd}) \), where \( p_{cg} \) is the \( g \)th word of PC, and there are \( d \) words in PC in total.

If a constraint quad has been extracted and its layer number equal to \( c \), then the following four dependency pair are detected.

1. \( dp[\text{subj} (v_{ps1}, p_{cs}), \text{obj} (v_{ps2}, n_{ps2})] \).
2. \( dp[\text{subjpass} (v_{ps1}, p_{cs}), \text{obj} (v_{ps2}, n_{ps2})] \).
3. \( dp[\text{subj} (v_{ps1}, p_{cs}), \text{nmod} (v_{ps2}, n_{ps2})] \).
4. \( dp[\text{subjpass} (v_{ps1}, p_{cs}), \text{nmod} (v_{ps2}, n_{ps2})] \).

A constraint quad (\( PC, VP, NP_{x}, c+1 \)) is extracted into the QG.

For the sample query, a constraint quads (\( \text{graph_databases}, \text{can_be_accessed_through}, \text{RDF_query_languages}, 1 \)) has been extracted, so \( PC = \{\text{RDF query languages}\} \). A dependency pair \( dp[\text{subjpass} (\text{accessed-12}, \text{databases-3}), \text{nmod} (\text{accessed-9}, \text{languages-14})] \) is detected. Here, “accessed-9” is \( v_{ps1} \), “databases-3” is \( p_{cs} \), and “languages-14” is \( n_{ps2} \), as shown in Fig. 4. Thus, the constraint quads (\( \text{RDF_query_languages}, \text{support}, \text{subgraph_extraction}, 2 \)) is extracted into the QG.

**Pattern 3:** In a query \( Q \), if \( \text{whvp1} \) is “who” and \( \text{WHVP}_{VP} \) is not “is,” the “PERSON” is added to QG as a category constraint of direct answers. For example, a query “who created Python?” is constructed as (PERSON, created, Python, 1), where the \( \text{whvp}1 \) equals to “who,” a category constraint “DATE” is added into QG. If \( \text{whvp}1 \) is “what” and \( \text{WHVP}_{VP} \) is not “is,” a category constraint ANYENTITY (a wildcard of any entities) is put into QG.

**Pattern 4:** If in a query \( Q \), \( \text{WHVP} \) is detected at the beginning of \( Q \). The VP following the interrogative word is “is” and the \( \text{whvp}2 \) is not “when” like query “What is Java Servlet?” We regard this question as a definition query and construct the QG as (ANYENTITY, ANYRELATION, Java_Servlet, 1) and (Java_Servlet, ANYRELATION, ANYENTITY, 1), where ANYRELATION represents a wildcard of any relation.

**Pattern 5:** If in a query \( Q \), \( m = 0 \) and \( n = 0 \), the first word of \( Q \) is “List” followed by only an NP, e.g., “list graph database.” We construct the QG as (\( \text{graph_database}, \text{ANYRELATION}, \text{ANYENTITY}, 1 \)).

### C. BotReasoning

As described in Section IV-B, a query graph QG is constructed. In this section, we elaborate on how BotReasoning reasons the answers and generates explanations for QG. Specifically, first, BotReasoning extracts a constraint quads currentCQ from the outer layer of QG and searches a candidate subgraph SG from KG, as shown in Section IV-C1. Second, BotReasoning generates the candidate answers and extracts the novel features of them. Then, a decision-making algorithm (e.g., Bayesian decision theory, DNN) determines the answers AS of currentCQ. These operations are clarified in Section IV-C2. Third, BotReasoning replaces the object of connected inner layer of currentCQ by AS and prunes currentCQ from QG. BotReasoning repeats the above iteration operation until the last constraint quads of QG is solved and then returns the direct answers and their explanations as shown in Section IV-C3.

1) **Candidate Subgraph Search:** Assume that the current constraint quads for candidate subgraph search is currentCQ = \( (c_{subj}, c_{predicate}, c_{obj}, c_{layer}) \), and the knowledge graph KG = \([\text{node}_1, \text{node}_2, \ldots, \text{node}_e, \ldots, \text{node}_n] \), where \( \text{node}_i \) is the \( i \)th node of KG and \( n \) is the number of nodes of KG. The pseudocode of subgraph search is shown in Algorithm 1.

Each node in KG has an initial associated activation value \( a_i \in \mathbb{R} \) and \( 0 < a_i < 1 \). A link \( \text{link}_{ij} \) connects source node \( \text{node}_i \) to target node \( \text{node}_j \) and the weight of link \( \text{link}_{ij} \) is \( w_{ij} \), where \( w_{ij} \in \mathbb{R} \) and \( 0 < w_{ij} < 1 \). The nodes of KG have an active threshold \( AT \), where \( AT \in \mathbb{R} \) and \( 0 < AT < 1 \). There is a decay factor \( DF \), where \( DF \in \mathbb{R} \) and \( 0 < DF < 1 \).

#### Algorithm 1 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, \( CR \): the crossover relation

1. Define a set \( S \) to store the nodes that can spread activation
2. Link \( c_{subj} \) and \( c_{obj} \) to nodes \( \text{node}_{subj} \) and \( \text{node}_{obj} \) in \( KG \)
3. Subroutine \( SpdActi(node) \)
4. Insert \( \text{node} \) into \( S \)
5. while \((ST > 0)\)
6. \( if \ (S \neq \emptyset) \) then
7. Insert \( S \) into \( SG_{temp} \)
8. for (each \( \text{node}_{i} \in S \)) do
9. \( if \ (\text{node}_{i} = \text{node}_{obj} \) and \( \text{node}_{j} \in SG_{subj} \)) then
10. Insert \( (\text{node}_{j}, \text{link}_{ij}, \text{node}_{i}) \) into \( CR \)
11. else
12. Spread activation and adjust value to every neighboring \( \text{node}_{j} \) according to Equation.2 and Equation.3
13. end if
14. end for
15. Replace all the nodes of \( S \) by activated nodes in this round
16. end if
17. \( ST -- \)
18. end while
19. \( bfr \)eturn \( SG_{temp} \)
20. EndSubroutine
21. \( SG_{subj} \leftarrow SpdActi(node_{subj}) \) \( SG_{obj} \leftarrow SpdActi(node_{obj}) \)
22. \( SG \leftarrow SG_{subj} \cup SG_{obj} \)
23. \( bfr \)eturn \( SG_{subj}, SG_{obj}, SG \) and \( CR \)

In the beginning of the subgraph search, BotReasoning links \( c_{subj} \) and \( c_{obj} \) to the corresponding \( \text{node}_{subj} \) and \( \text{node}_{obj} \) in the KG.
Therefore, the initial activation values of node subj and node obj are greater than active threshold AT. These two nodes are activated and initiate to spread the activation to all the neighboring nodes parallelly, as shown in Fig. 5.

Assume that link $ij$ connects the source node node $i$ to the target node node $j$. While the activation spreads from node $i$ to node $j$, node $i$ can compute $a_{j\text{.temp}}$ as

$$a_{j\text{.temp}} = a_j + (a_i \ast w_{ij} \ast DF). \tag{2}$$

Then, $a_j$ adjusts its value to $a'_j$ according to $a_{j\text{.temp}}$ as follows:

$$a'_j = \begin{cases} 
1, & \text{if } a_{j\text{.temp}} \geq 1 \\
 a_{j\text{.temp}}, & \text{if } AT \leq a_{j\text{.temp}} < 1 \\
 a_j, & \text{if } a_{j\text{.temp}} < AT.
\end{cases} \tag{3}$$

Once node $j$ is activated, it will initiate the next spreading activation cycle to its neighboring nodes. The spreading is terminated under the following three situations.

1) The spreading arrives at the endmost nodes of the KG or exceeds the preset upper bound ST.
2) All the nodes reach AT.
3) During searching node obj, if the neighboring node node $j$ of node $i$ belongs to SG subj. The spreading is terminated and $(node_j, link_{ij}, node_i)$ is inserted into CR, where CR are the crossover relations indicating the relations of $c_{subj}$ and $c_{obj}$.

Fig. 5 shows an example to show how spread activation works from one node to others. The gradually varied blue of the nodes means the decay of the activated values of the nodes. In this example, the initial activate value of the first node is 1, and the AT is 0.35. This spread activation process terminates after four times spreading.

2) Decision-Making:

a) Topological structure of subgraph SG: Fig. 6 shows an example of SG. The root node and leaf node refer to the most superclass node and most subclass node of SG, respectively. The candidate answers are all the leaf nodes of SG subj.

Assuming that the red focus node in Fig. 6 is a candidate answer for making a decision (to decide whether the candidate answer is a correct answer or not). The following two kinds of potential topological structures of the CR can affect decision-making. $R1$ refers to the candidate answer or any superclasses of the candidate answer connected to $c_{obj}$ or any superclasses of $c_{obj}$ by the predicate. $R2$ indicates that the candidate answer or any superclasses of candidate answer connect to any subclasses of $c_{obj}$ by the predicate.

b) Predicate similarity: There are many different ways to express the same meaning and opposite meaning. Therefore, the predicates of CR also affect the decision-making. To measure the semantic relevance between two predicates, a technique named word2vec [57]–[59] is adopted. It assumes that words appear in similar context may have similar meanings [60] and transforms the words or phrases into the form of continuous vectors [57]. Then, words or phrases can do relevance computation by distance measurement of vector (here, we use cosine similarity).

c) Features generations: In order to estimate the candidate answers, sufficient evidence needs to be gathered to determine whether it surpasses the positive or negative criteria. In general, the weights of positive and negative evidence are not equal. A few negative evidences may lead to negative decisions, whereas many positive evidences are needed to make a positive decision.

We synthesize the topological structure and predicate similarity into four features, as shown in Table III. In these four features, feature #1 represents the influence of $R1$. If $R1$ does not exist, the value of feature #1 takes 0; otherwise, it takes the maximum value of predicates similarity of all $R1$: $p_{r1\_pdictSim} = \max(pdictSim_{R1}i)$, where $i = (1, 2, \ldots, n)$ and pdictSim $R1i$ is the predicates similarity of $i$th $R1$ and $n$ is the number of $R1$. 

TABLE III
FEATURES TO REPRESENT THE CANDIDATE ANSWER

| #  | 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, 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. |

Feature #2 represents the influence of R2. If R2 does not exist, the value takes 0. Otherwise, the more R2 in CR, the higher the probability that the candidate answer meets the property constraint. Therefore, the value of feature #2 is calculated as $p_{r2_pdictSim} = 1 / (mn) \sum_{i=1}^{m} \sum_{j=1}^{n} p_{dictSim_{R2_{ij}}}$, where $p_{dictSim_{R2_{ij}}}$ is the predicates similarity of the $i$th leaf nodes of $c_{uobj}$ connected to the $j$th subclass of $c_{obj}$. $m$ is the number of leaf nodes of $c_{uobj}$ and $n$ is the number of subclasses of $c_{obj}$. Taking Fig. 6 as an example, here, $m = 2$ and $n = 2$, and assume the value of $p_{dictSim_{R2_{12}}}$ is 1 and $p_{dictSim_{R2_{22}}}$ is 0.5 and the value of feature #2 is $(1 + 0.5)/4 = 0.375$. For features #3 and #4, if negative R1 or R2 does not exist, the value takes 0; otherwise, it takes 1.

3) Final Answer and Explanation Generation: BotReasoning extracts the layer information of QG and the answer reasoning is performed from the outer layer to the inner layer. The reasoning order of the constraint quads in the same layer is determined by random. Once the answers AS exist, the value takes 0. Otherwise, the more AS in CR, the higher the probability that the candidate answer meets the decision-making constraint. Therefore, the value of feature #2 is calculated as $p_{r2_pdictSim} = 1 / (mn) \sum_{i=1}^{m} \sum_{j=1}^{n} p_{dictSim_{R2_{ij}}}$, where $p_{dictSim_{R2_{ij}}}$ is the predicates similarity of the $i$th leaf nodes of $c_{uobj}$ connected to the $j$th subclass of $c_{obj}$. $m$ is the number of leaf nodes of $c_{uobj}$ and $n$ is the number of subclasses of $c_{obj}$. Taking Fig. 6 as an example, here, $m = 2$ and $n = 2$, and assume the value of $p_{dictSim_{R2_{12}}}$ is 1 and $p_{dictSim_{R2_{22}}}$ is 0.5 and the value of feature #2 is $(1 + 0.5)/4 = 0.375$. For features #3 and #4, if negative R1 or R2 does not exist, the value takes 0; otherwise, it takes 1.

D. BotResponse: UI of DeveloperBot

A proof-of-concept prototype of DeveloperBot is implemented, as shown in Fig. 7. The user can enter their query at the input bar and press the button “Direct Search.” The application can return a list of direct answers, brief introductions, and their corresponding explanations. In the reasoning subgraph, the entities identified by the BotPerception module are represented as red color. The answers reasoning by BotReasoning are presented as graded blue that shows how the BotPlanning orders the constraints.

This reasoning subgraph explains “why” “virtuoso” is the answer: virtuoso is a graph database, virtuoso supports SPARQL, and SPARQL is an RDF query language that supports subgraph extraction. It also explains “how” “virtuoso” is achieved: first, DeveloperBot identifies the four entities “graph database,” “python,” “rdf query language,” and “subgraph extraction.” Second, it gets the answer “SPARQL” of constraint “RDF query language support subgraph extraction.” Finally, DeveloperBot searches the “graph database” supported both python and SPARQL. Also, the confidence of the answer “virtuoso” is 0.9861.

V. EXPERIMENT

In this section, a series of experiments is conducted to answer the following three research questions of DeveloperBot.

1) RQ1: How is the performance of different decision-making algorithms in estimating the correct answers?
2) RQ2: Whether DeveloperBot can improve the performance of developers during answer search? Is the performance different under different task complexity?
A cross-platform application may run on Microsoft Windows, Linux, and macOS. Therefore, we hire some domain experts to augment the knowledge graph by the common sense knowledge of the software engineering domain and do the clearance at the same time.

Third, we use the NER technique of coreNLP to recognize the predefined categories (e.g., person, company names, organizations, locations, and time expressions) from entities of the knowledge graph [49]–[52]. For example, the relation triples (Microsoft, was_founded_by, Bill Gates) are recognized as two named entities are “Microsoft”-ORGANIZATION and “Bill Gates”-PERSON. Thus, two relation triples (Microsoft, is_a, organization) and (Bill Gates, is_a, person) are added to the knowledge graph.

After that, the final knowledge graph acquires 39 022 concepts, 8173 unique predicates, and 35 938 relation triples. The knowledge covers the popular programming languages (e.g., Java, Javascript, and Python), frameworks (e.g., Flex and Django), database (e.g., neo4j and Mysql), and tools (e.g., d3 and MSBuild) of software engineering domain.

B. 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 are initialized to greater than AT. The maximum number of iterations is set to 30. The DNN model used in this experiment is a three-layer classifier whose hidden layer is set to [10, 20, 10]. For the decision-making process of Bayesian decision theory, \( p(o_0 = 0) = 0.15 \) and \( p(o_1 = 1) = 0.85 \).

The Stanford CoreNLP is used for NLP markup, dependency parse [49], and NER. The word2vec library of Gensim is used to compute word embedding [63]. The nltk library is used to do the VP, NP chunking [64].

C. Evaluation

In this section, quantitative evaluation (objective) and a user study (subjective) are conducted to evaluate the performance of DeveloperBot.

1) Quantitative Evaluation: In the quantitative evaluation, we invited five developers to ask some multiconstraint natural language questions they interested. We require that these questions should be related to general knowledge of the software engineering domain and can be answered by direct answers extracted from the content of the tag wiki of Stack Overflow. These questions mainly cover factual questions, such as who, what, which, and list. Finally, the five developers proposed 160 eligible questions. Then, they asked to use the computer to search for their answers to every question. Then, they discuss together the final answers for every question; 51 of the questions were dropped because they had no answer or their answers could not reach a consensus. Thus, the 109 ground truth questions and the 1819 candidate answers were generated.
To evaluate the decision-making process, the balanced accuracy, precision, recall, f1-score, and confidence MSE are used as evaluation metrics [65]. The average result of the ten times cross validation is taken as the final results of different metrics. Here, the MSE measures the average squared difference between the estimated confidence values and the actual confidence values: \( \text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (1 - \text{answerConfidence})^2 \), where \( n \) is the number of answers and 1 is the actual confidence values for correct answers. The balanced accuracy returns the average accuracy per class: 
\[
\text{Balanced Accuracy} = \frac{1}{2} \left( \frac{\text{TP}}{\text{TP} + \text{FP}} + \frac{\text{TN}}{\text{TN} + \text{FN}} \right),
\]
where TP, TN, FP, and FN are true positives, true negatives, false positives, and false negatives, respectively.

2) User Study:

a) Tools for comparison: In this user study, we compare the information search performance of using Google + DeveloperBot with using only Google. There are some direct answer search engines, such as Wolfram Alpha (https://www.wolframalpha.com/) and ASK (https://www.ask.com) that are similar to DeveloperBot. However, these tools can hardly answer specific questions of the software engineer domain, so it makes no sense to compare with them. Furthermore, this article mainly focuses on a general method to answer questions and provide explanations by a knowledge graph. It can be extended to many different areas, situations, and questions coverage, which depends on the knowledge graph used. It is just like us humans, learning different knowledge and becoming experts in different domains. The questions we can answer depend on our knowledge quantity.

b) Participants: In the user study, 24 developers from different backgrounds were recruited. They are made up of employees, undergraduate students, Ph.D. students, professors from different universities, professional developers from IT companies, and so on. Their programming proficiency distributions are 12.5%, 75%, and 12.5% for expert, competent, and beginner, respectively. The participants also have diverse programming background and their programming languages involved Python, Java, Javascript, SQL, C#, and so on. In each proficiency level and background, the developers are halved to G1 and G2 randomly (12 participants per group). To exclude the effects of participants’ programming level and fatigue, G1 and G2 act as experimental group (use Google + DeveloperBot) and control group (use only Google) to finish each task circularly Therefore, every developer has the opportunity to solve the simple or hard task by only Google or Google + DeveloperBot. They can also have a comprehensive comparison to the differences with or without the search engine assistant DeveloperBot.

c) Task: In this study, we have chosen five answer search tasks (represent as Task1 – Task5) from the ground truth questions, as shown in Table IV. The answers of these tasks cover celebrity of software engineering domain, library, operating system, IDE, database, and so on. According to the preliminary test, the difficulty of these five tasks increases in order. 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.

d) Procedure: This user study begins with a brief training of DeveloperBot by the experimenter. After finishing each task, participants need to fill their confidence to the answers they provided (on five-point Likert scale with one being the least confident and five being the most confident). A screen capture software is required to run throughout the whole process that allows us to analyze the participants’ behavior after the experiment. In the end, all the participants are asked to do an open discussion together. As a supplement to the experiment, this opens discussion focusing on their options about the different features of DeveloperBot. They can also give suggestions for tool improvement and some explanations about some exceptional situations in the experiment.

D. Results and Results Analysis

1) Result: Performance of Different Decision-Making Algorithms (RQ1): Here, we present the results of quantitative evaluation and answer the RQ1. Fig. 9 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 four features extracted by our algorithm, any decision-making algorithm can get more than 0.98 balanced accuracy. Compared with other algorithms, DNN achieves the highest precision 0.98, recall 0.98, and f1-score 0.99. From Fig. 9, we can infer that some decision-making algorithms can achieve high precision, but at the cost of a low recall, like SVM Linear. The Bayesian decision theory obtains a high recall, but the precision is low. In the future research, we might investigate the employment of other state-of-the-art
algorithms, such as negative correlation learning [66], [67],
statistical learning for probabilistic outputs [68], [69],
and Bayesian inference [70].

Fig. 10 shows the MSE of confidence for different decision-
making algorithms. It shows that DNN achieves the lowest
MSE for the estimation of answer confidence. The Bayesian
decision theory gets the highest MSE. The estimation of
answer confidence is within the acceptable range for all
decision-making algorithms.

To explore the effects of different features, we train three
DNN models using different features. These three models
use all features, only predicate similarity feature, and only
topological structure feature. The only predicate similarity
feature is computed by the maximal predicate similarity of
$R_1$, if it exists. Only topological structure features involve
four features to represent the existence (take the value 1)
and nonexistence (take the value 0) of $R_1$, $R_2$, $R_3$, and $R_4$.

As shown in Fig. 11, the decision-making algorithm with
all features performs the best for all the metrics. This fig-
ure also indicates that the predicate similarity contributes to
the precision significantly. With only predicate similarity feature,
the precision almost can achieve as same as all features.
However, only predicate similarity feature obtains very low
recall. These results show 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.

Fig. 12 shows that the decision-making algorithm can
obtain 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.

2) Result: Improve the Performance of Developers During
Answer Search (RQ2): In this article, we use three indexes to
quantize the performance of developers during answer search:
1) answer accuracy: the proportion of correct answers in total
answers; 2) answer confidences: participants’ confidences to
answers of each task; and 3) answer time: the time (second)
used for each search task. We present the average values and
conduct the one-way ANOVA to show the result of RQ2.

Fig. 13 shows the result of the answer accuracy. It shows
that for tasks 1 and 2, both groups can finish the tasks with
high accuracy (above 0.86). The answer accuracy of the par-
ticipants who use only Google even reaches 1 on task 1. These
prove our preexperiment conjecture: Google has a very good
performance in simple information search tasks. For tasks 3–5,
the experimental group can finish with higher answer accuracy.
With the complexity of tasks increasing, the answer accuracy
of the control group gradually decreases compared with the
experimental group, which maintains at a high level (above
0.78). 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-ended questions, especially in
complex search tasks.

Fig. 14 shows the result of participants’ average confidences
between their answers. This figure indicates that for tasks 1–3,
the participants’ confidences of the control group are almost
the same as the experimental group. For tasks 4 and 5,
the participants’ confidences (4 and 4.3 for tasks 4 and 5,
respectively) of the experimental group show an overwhelming
advantage. With the complexity of tasks increasing, the dif-
ference of participants’ confidences of the two groups shows
Table V

| Task     | Answer Accuracy | Answer Confidence | Answer Time |
|----------|-----------------|-------------------|-------------|
| Task 1   | 0.32818         | 0.15220           | 0.75474     |
| Task 2   | 0.84698         | 0.13729           | 0.64641     |
| Task 3   | 0.00025         | 0.78410           | 0.63092     |
| Task 4   | 0.00090         | 0.07561           | 0.00465     |
| Task 5   | 0.00003         | 0.00013           | 0.06353     |
| All Tasks| 0.00003         | 0.00028           | 0.07994     |

Fig. 14. Result of answer confidence.

Fig. 15. Result of answer time.

A clear increasing trend except for task 3. According to the observation of the screen recording and results of the open discussion, this is because there is a web page that lists some direct answers to task 3, and each answer has a corresponding text explanation. The confidences of the participants in the control group increase by reading these text explanations. However, this is at the cost of increasing the time used on task 3, as shown in Fig. 15. These indicated that for the easy tasks, the control group was more confident in their answers because they were straightforward and did not require reasoning or summarization. For the complex tasks, despite spending a long-time searching answers, participants still feel less confident in their answers because they may lose during they try to summarize and reason the answers through many web pages. Therefore, as a search engine assistant, DeveloperBot can improve the participants’ confidences by providing explanations of answers.

The one-way ANOVA is also used to evaluate how DeveloperBot affects the performance of the user study. In this experiment, the significance level of one-way ANOVA is set at $\alpha = 0.05$. The result of the $p$-value of the one-way ANOVA is shown in Table V. From Table V, we can see the same trend and conclusion as above. For the simple tasks (tasks 1 and 2), the use of DeveloperBot has no significant effect on the results ($p$-value $> 0.05$). With the complexity of tasks increasing (tasks 3–5), the use of DeveloperBot is a critical factor that influenced developers’ performance in answer accuracy, answer confidence, and answer time.

3) Result: Comments to Explanations and Behavior Changes (RQ3): We evaluate the explanation and answer the RQ3 in three ways: the quantitative scoring (0–5) of participants in the questionnaires, the information obtained from open discussions, and our observation of screen recording.

The results of the quantitative scoring of explanations are shown in the radar graph Fig. 16. Five indexes are adopted in this experiment to evaluate the quality of the explanation. They are readable, understandable, answer convincing, wrong answer identification, and search keywords forming [71], [72], where readable and understandable mean the explanation is legible, easy to read, and understand. Answer convincing means that this explanation can make the direct answer more convincing. Wrong answer identification is the index that evaluates whether the explanation can help identify the wrong complex answer search tasks that take a long time to solve with only Google. The open discussion also reflects consistent results for the above conclusion. The special performance of the Google group at task 3 also illustrates that direct answers and the explanation (even text) can significantly reduce the search time and improve the participants’ confidences in their answers at the same time.
answer. Search keywords forming shows the capacity of the explanation to help form better search keyword.

As shown in Fig. 16, 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 is second and third, with average values of 4.38. Even the wrong answer identification achieves the lowest score, 4.13 is still acceptable. The results demonstrate that the reasoning subgraph is a reasonable and high-quality explanation of direct answer and meet the desired design goals.

In addition, Fig. 16 also shows the overall scores of using confidence as an explanation. Compared with the reasoning subgraph, the overall scores of confidence are relatively lower (3.71–4.17). The indexes readable, understandable, and answer convincing achieve the highest score 4.17. The scores of wrong answer identification and search keywords forming are 3.79 and 3.71, respectively. Even the overall scores of confidence are lower than the reasoning subgraph, especially the search keywords forming and wrong answer identification. However, the good performance of indexes, such as readable, understandable, and answer convincing, indicates that confidences are good explanations to direct answers.

The results of the questionnaire and the contents of the open discussion show that, using the reasoning subgraph extracted following the cognitive process and answer confidence as explanations 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.

VI. CONCLUSION

The developers have a need for complex information search that provides specific direct answers. The current search engines are weak at the reasoning capacity for complex closed-ended questions. In order to address these issues, in this article, a brain-inspired search engine assistant named DeveloperBot is proposed. DeveloperBot aligns to the cognitive process of humans and has the capacity to answer complex queries with good explainability by learned knowledge. The experimental results show that the novel features of the subgraph can estimate the answers and answer confidences with high accuracy. The results of the user study show that compared with using only Google, with the assistance of DeveloperBot, users can find answers faster and with more accuracy. In addition, with the explanations extracted following the cognitive process, DeveloperBot can significantly improve the developers’ trust and adoption of the answers. These explanations also assist the developers to understand the answers more deeply, improve the answer accuracy, and form better search keywords. Furthermore, for relatively complex queries, with the assistance of DeveloperBot, the search performance improvement of the developers is more significant. In the future, we will augment the knowledge graph through developers’ development behaviors and working context to further improve the efficiency of information search [62], [73], [74]. We also want to further evaluate the effectiveness of DeveloperBot for some other search engines (e.g., Bing and Yahoo) and other knowledge graphs.

ACKNOWLEDGMENT

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of the National Research Foundation, Singapore. The authors appreciate all participants for the user study, and they especially thank Naoya Sawada for the assistance of demo website construction. They also appreciate the article polishing and constructive discussion of Dr. Ching-Hung Lee and Tan Wang.

REFERENCES

[1] B. Sarrafzadeh, O. Vechtomova, and V. Jokic, “Exploring knowledge graphs for exploratory search,” in Proc. 5th Inf. Interact. Context Symp., Aug. 2014, pp. 135–144.
[2] C. N. Carlson, “Information overload, retrieval strategies and internet user empowerment,” in P. C. Good, Bad Irrelevant (COST Action), Helsinki, Finland, Sep. 2003.
[3] G. Singer, U. Norbisrath, and D. Lewandowski, “Ordinary search engine users carrying out complex search tasks,” J. Inf. Sci., vol. 39, no. 3, pp. 346–358, Jun. 2013.
[4] F. Joel. Google: Our New Search Strategy is to Compute Answers. Accessed: Aug. 14, 2006. [Online]. Available: https://thenextweb.com/google/2011/06/01/google-our-new-search-strategy-is-to-compute-answers-not-links/
[5] G. Seth. Breakthrough Analysis: Search is Not the Answer. Accessed: Aug. 14, 2006. [Online]. Available: https://www. informationweek.com/software/information-management/breakthrough- analysis/search-is-not-the-answer/id=1046118
[6] G. Liz. Breakthrough Analysis: Search is Not the Answer. Accessed: Mar. 14, 2013. [Online]. Available: http://allthingsds.com/20130314/how-search-is-evolving-finally-beyond-caveman-queries/
[7] X. Zhao, F. Xiao, H. Zhong, J. Yao, and H. Chen, “Condition aware and revise transformer for question answering,” in Proc. Web Conf., Apr. 2020, pp. 2377–2387.
[8] X. Wu et al., “Knowledge engineering with big data,” IEEE Intell. Syst., vol. 30, no. 5, pp. 46–55, Sep./Oct. 2015.
[9] R. Hong, M. Wang, G. Li, L. Nie, Z.-J. Zha, and T.-S. Chua, “Multimedia question answering,” IEEE MultimediaMag., vol. 19, no. 4, pp. 72–78, Oct. 2012.
[10] 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. Hershey, PA, USA: IGI Global, 2013, pp. 474–490.
[11] S. Quarteroni and S. Manandhar, “A chatbot-based interactive question answering system,” in Proc. 11th Workshop Semantics Pragmatics Dialogue (Ducalog), Trento, Italy, vol. 83, 2007, pp. 83–90.
[12] Z. Vargas, F. Weng, and H. Pon-Barry, “Interactive question answering and constraint relaxation in spoken dialogue systems,” in Proc. 37th SIGdial Workshop Discourse Dialogue (SigDIAL), 2006, pp. 28–35.
[13] H.-C. Tu and J. Hsiang, “An architecture and category knowledge for intelligent information retrieval agents,” Decis. Support Syst., vol. 28, no. 3, pp. 255–268, May 2000.
[14] W. Wu, H. Chen, J. Liu, G. Wu, R. Lu, and N. Zheng, “Knowledge engineering with big data (BiKE): A 54-month, 45-million RMB, 15-institution national grand project,” IEEE Access, vol. 5, pp. 12696–12701, 2017.
[15] S. Chari, D. M. Gruen, O. Seneviratne, and D. L. McGuinness, “Foundations of explainable knowledge-enabled systems,” 2020, arXiv:2003.07520. [Online]. Available: http://arxiv.org/abs/2003.07520
[16] X. Yanghua. (Oct. 13, 2017). Explainable Artificial Intelligence Based on Knowledge Graph: Opportunities and Challenges. [Online]. Available: https://blog.biostat.com/ID=00660-6bd13a69-50b0-4853-9413-6c4849d2c92f
[17] G. Maheshwari, P. Trivedi, D. Lukovikov, N. Chakraborty, A. Fischer, and J. Lehmann, “Learning to rank query graphs for complex question answering over knowledge graphs,” in Proc. Int. Semantic Web Conf. Cham, Switzerland: Springer, 2019, pp. 487–504.
[71] R. Confalonieri et al., “What makes a good explanation? Cognitive dimensions of explaining intelligent machines,” in Proc. CogSci, 2019, pp. 25–26.

[72] R. R. Hoffman, S. T. Mueller, G. Klein, and J. Litman, “Metrics for explainable AI: Challenges and prospects,” 2018, arXiv:1812.04608. [Online]. Available: http://arxiv.org/abs/1812.04608

[73] 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 Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops (ISSREW), Oct. 2018, pp. 24–29.

[74] H. Li et al., “AmAssist: In-IDE ambient search of online programming resources,” in Proc. IEEE 22nd Int. Conf. Softw. Anal., Evol., Reengineering (SANER), Mar. 2015, pp. 390–398.

Xuejiao Zhao received the Ph.D. degree in computer science from the School of Computer Science and Engineering (SCSE), Nanyang Technological University, Singapore, in 2021. He is currently a Research Fellow with the NTU-UBC Joint Research Centre of Excellence in Active Living for the Elderly (LILY), Nanyang Technological University. His research interests include knowledge graph, explainable artificial intelligence (AI), AI in healthcare, graph neural networks, and software engineering.

Huanhuan Chen (Senior Member, IEEE) received the B.Sc. degree from the University of Science and Technology of China (USTC), Hefei, China, in 2004, and the Ph.D. degree in computer science from the University of Birmingham, Birmingham, U.K., in 2008. He is currently a Full Professor with the School of Computer Science and Technology, USTC. His current research interests include neural networks, Bayesian inference, and evolutionary computation. Dr. Chen was a recipient of the 2015 International Neural Network Society Young Investigator Award, the 2012 IEEE Computational Intelligence Society Outstanding Ph.D. Dissertation Award, the IEEE TRANSACTIONS ON NEURAL NETWORKS Outstanding Paper Award (bestowed in 2011 and only one paper in 2009), and the 2009 British Computer Society Distinguished Dissertations Award. He is also an Associate Editor of the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS and the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL INTELLIGENCE.

Chunyan Miao (Senior Member, IEEE) is currently a Professor with the School of Computer Science and Engineering (SCSE), Nanyang Technological University (NTU), Singapore. She is also the Director of the NTU-UBC Joint Research Centre of Excellence in Active Living for the Elderly (LILY). Prior to joining NTU, she was an Instructor and a Post-Doctoral Fellow with the School of Computing, Simon Fraser University, Burnaby, BC, Canada. Her research focuses on studying the cognitive and social characteristics of intelligent agents in multiagent and distributed artificial intelligence (AI/CI) systems, such as trust, emotions, incentives, motivated learning, ecological, and organizational behavior. She has worked on new disruptive AI approaches and theories that synergize human intelligence, AI, and behavior data analytics (AI powered by humans). Her current research interests include human–agent interaction, cognitive agents, human computation, and serious games.

Zhencang Xing is currently an Associate Professor with the School of Computing, Australian National University, Canberra, ACT, Australia. Previously, he was an Assistant Professor with the School of Computer Science and Engineering, Nanyang Technological University, Singapore, from 2012 to 2016. Before joining NTU, he was a Lee Kuan Yew Research Fellow with the School of Computing, National University of Singapore, Singapore, from 2009 to 2012. He has over 150 peer-reviewed publications in highly competitive and reputable journals and conferences. His research area is in the interdisciplinary areas of software engineering, human–computer interaction, and artificial intelligence, with the specific focus on developing knowledge graph methods and behavior analytics for improving software development productivity and software quality. Prof. Xing has received multiple distinguished paper awards from the top software engineering conferences, including two ACM SIGSOFT Distinguished Paper Awards and two IEEE TCSE Distinguished Paper Awards. He regularly serves on the program committees of the top software engineering conferences. He was the Program Committee Co-Chair of the 36th IEEE International Conference on Software Maintenance and Evolution (ICSME2020). He is an Associate Editor of the Journal of Software: Evolution and Process.