# ROS package search for robot software development: a knowledge graph-based approach

Shuo WANG<sup>1,2</sup>, Xinjun MAO(⋈)<sup>1,2</sup>, Shuo YANG<sup>3</sup>, Menghan WU<sup>4</sup>, Zhang ZHANG<sup>1,2</sup>

- 1 College of Computer, National University of Defense Technology, Changsha 410000, China
- 2 Key Laboratory of Software Engineering for Complex Systems, Changsha 410000, China
- 3 College of Systems Engineering, National University of Defense Technology, Changsha 410000, China
  - 4 College of Computer Science and Technology, Zhejiang University, Hangzhou 310058, China

© Higher Education Press 2023

**Abstract** In recent years, ROS (Robot Operating System) packages have become increasingly popular as a type of software artifact that can be effectively reused in robotic software development. Indeed, finding suitable ROS packages that closely match the software's functional requirements from the vast number of available packages is a nontrivial task using current search methods. The traditional search methods for ROS packages often involve inputting keywords related to robotic tasks into general-purpose search engines (e.g., Google) or code hosting platforms (e.g., Github) to obtain approximate results of all potentially suitable ROS packages. However, the accuracy of these search methods remains relatively low because the taskrelated keywords may not precisely match the functionalities offered by the ROS packages. To improve the search accuracy of ROS packages, this paper presents a novel semantic-based search approach that relies on the semantic-level ROS Package Knowledge Graph (RPKG) to automatically retrieve the most suitable ROS packages. Firstly, to construct the RPKG, we employ multi-dimensional feature extraction techniques to extract semantic concepts, including code file name, category, hardware device, characteristics, and function, from the dataset of ROS package text descriptions. The semantic features extracted from this process result in a substantial number of entities (32,294) and relationships (54,698). Subsequently, we create a robot domain-specific small corpus and further fine-tune a pre-trained language model, BERT-ROS, to generate embeddings that effectively represent the semantics of the extracted features. These embeddings play a crucial role in facilitating semanticlevel understanding and comparisons during the ROS package search process within the RPKG. Secondly, we introduce a novel semantic matching-based search algorithm that incorporates the weighted similarities of multiple features from user search queries, which searches out more accurate ROS packages than the traditional keyword search method. To validate the enhanced accuracy of ROS package searching, we conduct comparative case studies between our semantic-based search approach and four baseline search approaches: ROS Index, GitHub, Google, and ChatGPT. The experiment results demonstrate that our approach achieves higher accuracy in terms of ROS package searching, outperforming the other approaches by at least 21% from 5 levels, including top1, top5, top10, top15, and top20.

**Keywords** robot software, ROS, package search, Knowledge Graph

# 1 Introduction

As autonomous robots are widely used in human society, developing robotic software has become a prevalent practice in the robotic community, responding to the high demand from robot consumers. However, the development of robotic software has always been challenging due to its requirement of interdisciplinary knowledge in AI, control, and software engineering, as well as significant programming effort for implementing robotic software components [1]. ROS has emerged as the de facto standard for robotic software development frameworks. It offers a pleth-ora of reusable ROS packages, each implementing an independent robotic software functionality such as task planning, actuation, and visual sensing [2]. By reusing and combining these

Received month dd, yyyy; accepted month dd, yyyy

E-mail: xjmao@nudt.edu.cn

package search process within the RPKG. Secondly, packages, developers can quickly construct a rowe introduce a novel semantic matching-based search bust robotic software framework and implement taskalgorithm that incorporates the weighted similarispecific algorithms when needed [3,4].

As of August 10, 2023, there are 2,644 software repositories and 7,570 packages that have been developed using the ROS framework<sup>1)</sup>. However, searching for the most suitable ROS package for a specific robotic software functionality remains a time-consuming and often inaccurate process. Existing search methods primarily rely on direct text matching, where a set of task-related keywords are inputted into a search engine to obtain approximate results. Unfortunately, this approach often yields irrelevant ROS packages due to its disregard for the semantic information of the tasks, which considerably prolongs the searching process.

Currently, four mainstream approaches are commonly used to search for ROS packages. (1) Searching using the ROS Index website, which serves as a gateway to search for ROS and ROS2 resources, including packages, repositories, system dependencies, and documentation<sup>2)</sup>. It allows developers to enter keywords and phrases in the search bar to find suitable packages, ranked by relevance calculated using the BM25 algorithm [5]. However, ROS Index often outputs irrelevant ROS packages due to its reliance on syntactic word matching. In addition, it lacks semantic comprehension, making it challenging to deal with synonyms and abbreviations. (2) Searching GitHub repositories, where most ROS packages can be found. GitHub matches user queries with names and descriptions of all repositories, yielding search results based on character syntactic matching [6]. For instance, the word "do" may be matched to "docker" in the description of a repository. Since the search repositories must

<sup>1)</sup>https://index.ros.org/stats/

<sup>&</sup>lt;sup>2)</sup>https://index.ros.org/

include all the words from the user queries, it is not uncommon for GitHub to return no results for queries with up to three words. Furthermore, each repository often contains more than one ROS package. (3) Searching using the Google search engine. By utilizing a custom search combining ROS Index and GitHub, developers can limit their search scope by adding "site:ros.org OR site:github.com" to their queries. All three search methods rely on text syntactic matching without extracting and understanding the semantics of ROS packages, leading to mismatches between search results and user queries. (4) Searching using ChatGPT. Recent advancements in Large Language Models (LLMs) have made ChatGPT<sup>3)</sup> one of the most powerful semantic models. ChatGPT improves the understanding of the semantics of ROS packages, allowing users to interact in natural language to find ROS packages. However, ChatGPT may struggle with domainspecific words such as "TurtleBot2" and "Turtle-Bot3", limiting its accuracy in searching ROS packages.

These four search approaches for ROS packages only utilize the text description of ROS packages syntactically and do not exploit the rich features of ROS packages on multiple dimensions. To gain a comprehensive understanding and representation of ROS packages, we require a high-level abstraction of ROS packages to identify the multi-dimensional concepts and relationships within them to facilitate the search task. Knowledge graphs have been widely applied in various fields such as software engineering in recent years as they provide structured knowledge representation and facilitate knowledge retrieval. To represent the multi-dimensional concepts and relationships of ROS packages in a structured format and support the search task,

we propose using a knowledge graph as a domain knowledge base for ROS packages. Additionally, keyword matching-based search methods are not suitable for knowledge graphs, necessitating the design of a new knowledge graph-based search algorithm for ROS packages.

To improve the search accuracy of ROS packages, this paper presents a novel semantic-based search approach that relies on the semantic-level ROS Package Knowledge Graph (RPKG) to automatically retrieve the most suitable ROS packages. Firstly, to enrich the semantics of ROS packages, we utilize multi-dimensional feature extraction techniques to extract the semantic concepts of categories, functions, characteristics, associated robot hardware, etc., based on the dataset of ROS package text descriptions. RPKG is constructed based on the extracted semantic concepts of 32,294 entities (11 types) and 54,698 relationships (10 types), which supports retrieving knowledge on ROS packages in a structured way. To represent the semantics of ROS package features, we build a corpus in the robot domain and further fine-tune the pretrained language model BERT, which improves the semantic similarity matching for natural language. Secondly, to support the search for ROS packages, we propose a novel semantic matching-based search algorithm by incorporating weighted similarities of multiple features of the user's search queries. To assess the quality of extracted semantic features from ROS packages, we engage two robot software developers to evaluate the accuracy of sampled features across multiple dimensions, which yields an accuracy rate of over 91.6%. To validate the enhanced accuracy of ROS package searching, we conduct comparative case studies, comparing our semantic-based search approach with four baseline search approaches: ROS Index, GitHub, Google,

<sup>3)</sup>https://chat.openai.com/

and ChatGPT. The experimental results demonstrate that our approach achieves higher accuracy in terms of ROS package searching, outperforming the other approaches by at least 21% from 5 levels, including top1, top5, top10, top15 and top20.

To summarize, the main contributions of this paper are as follows:

- To understand and represent ROS packages more comprehensively, we first establish a high-level abstraction view of ROS packages, which discusses the main concepts of each ROS package and the main relationships between ROS packages.
- To enrich the semantics of ROS packages, we propose a multi-dimensional feature extraction technique and construct a ROS package knowledge graph, which more comprehensively represents ROS packages from multiple dimensions in a structured way.
- To support the knowledge graph-based search for ROS packages, we design a novel ROS package feature fusion matching algorithm by considering weighted similarities of multiple features from users' queries, which outperforms existing methods by at least 21% in search accuracy.

The remainder of this paper is organized as follows. We discuss the current related works in Section 2. Section 3 proposes the high-level abstraction view of ROS packages. Section 4 introduces the construction of RPKG. Section 5 presents the RPKG-based ROS search approach. Evaluations are shown in Section 6. We close with our conclusions and future work in Section 8.

# 2 Related work

## 2.1 Code Reuse in ROS

In recent studies, researchers pay more attention to the ecosystem of ROS and the dependencies between ROS packages, among other factors. Estefo et al. conducted an empirical study consisting of interviews and a survey with ROS developers to study the evolution challenges faced by the ROS ecosystem, especially the issues surrounding package reuse and bottlenecks to contribute to existing packages [7]. They also studied the code quality of the ROS ecosystem through an analysis of code duplication in ROS launchfiles [8]. Jiang and Mao conducted empirical research on questions about using launch files on Stack Overflow and ROS Answers (a Q&A platform for ROS) [9]. Witte and Tichy analyzed ROS nodes and their launch files to check consistency and issue warnings for robot software in ROS [10]. Malavolta et al. presented an observational study that unveils the state-of-thepractice for architecting ROS-based systems and provides guidelines for roboticists to properly architect their ROS-based systems [11]. Kolak et al. studied the growth of the ROS ecosystem and the collaboration among its members [1]. They also analyzed the dependencies between packages to study the structure of the ROS ecosystem. Canelas et al. reported the challenges and experiences of learning and using ROS from the perspective of ROS newcomers. They found that there are still many common misunderstandings and mistakes when newcomers use ROS to develop their robotic software [12]. Macenski et al. reviewed the philosophical and architectural changes of ROS 2, which powers the robotics revolution [13]. The aforementioned studies mainly focus on the usage and development of ROS but do not specifically investigate

the problem of finding suitable ROS packages for robot software.

#### Knowledge Graph in Software Engineering 2.2

Recently, there has been increasing interest in applying knowledge graphs in software engineering due to their powerful ability in knowledge representation, reasoning, and support for knowledge search [14, 15]. However, very few studies have focused on the application of knowledge graphs in robotics. Tenorth and Beetz designed the knowledge processing system KnowRob to equip robots with the capability to organize information in reusable using procedural knowledge and automatic proceknowledge chunks and to perform reasoning in an expressive logic [16]. Sun et al. constructed a ROS message knowledge graph (RMKG) to assist users in finding appropriate ROS message types [15]. They mainly extracted features of ROS messages from their names and proposed a fuzzy matching algorithm to support the search for ROS messages. Jiamou et al. developed open information extraction (OpenIE) techniques to extract candidates for task activities, activity attributes, and activity relationships from Android programming task tutorials [14]. Based on these extracted features, they built a knowledge graph TaskKG, which enables activitycentric knowledge search. Zou et al. proposed a code-centric software knowledge model and provided a two-layer plugin framework for knowledge graph construction and software Q&A [17].

# Search Methods in Software Development

Nowadays, developing software from scratch is still quite difficult and time-consuming. Software reuse has proven to be an advantageous method for simplifying software development. However, before reusing software or a component, it is necessary to

find a suitable one that meets the software requirements. In the past decades, researchers have conducted numerous studies on software search and recommendation. Seacord et al. described Agora as a prototype search engine that automatically generates and indexes a worldwide database of software products [18]. López Pino B E developed a tool to retrieve data from GitHub and ROS Answers, and then proposed a recommender algorithm to answer questions on ROS Answers [19]. Agora allows users to query Java components with methods or properties as inputs. Yang and Nyberg investigated two problems: search task suggestion dural knowledge base construction from search activities [20]. They proposed the creation of a threeway parallel corpus of queries, query contexts, and task descriptions, and reduced both problems to sequence labeling tasks. They also proposed a supervised framework to identify queryable phrases from task descriptions and extract procedural knowledge from search queries and relevant text snippets. Silva et al. proposed CROKAGE (Crowd-Knowledge Answer Generator), a tool that takes the description of a programming task (the query) as input and delivers relevant code examples for the task [21]. CROKAGE leverages the knowledge stored in Stack Overflow to generate solutions containing source code and explanations for programming tasks written in natural language.

# **High-level Abstraction of ROS Pack**ages

ROS packages, as commonly known by most roboticists, are implementation-level software artifacts that are frequently reused and programmed in various robotic software projects. In this paper, we establish a high-level abstraction view of ROS packages,

which discusses the main concepts of each ROS package and the main relationships between ROS packages.

Properties related to ROS packages is various, including authors, emails, development status, last updated date, code files, descriptions, etc. However, some of these properties are unimportant or irrelevant when searching for a ROS package, so there is no need to include them in the RPKG model. We first analyze the main concepts of ROS packages and then discuss the interconnected relationships between ROS packages.

# 3.1 ROS Package Concepts

We thoroughly investigate the features present in developers' requirements to guide the search for ROS packages. For this purpose, we examine the 100 most recent questions from ROS Answers, where users have requested assistance in searching for ROS packages. We delve into each question and its corresponding ROS package answers, ultimately identifying five key concepts that significantly contribute to finding the most suitable ROS packages:

- **Robot**: It specifies the robot that users use. Some ROS packages have specific scopes and adaptations for certain robots.
- **Sensor**: It specifies the sensor that users use.
- Package category: It specifies the category of the ROS package, such as Message package, Description package, Meta package, or Function package. The Message package is used for defining ROS messages transferred between packages. The Description package is used for describing the physical models of the robot hardware. The Meta package<sup>4)</sup> is used as a virtual package, which does not

- provide any functional code files, but rather references one or more related packages that are loosely grouped together. We define the Function package as a package that implements specific functions by processing data or interacting with the physical environment.
- Package function: It specifies the function that a package can provide, such as mapping, achieving navigation, or bringing up the robot.
- Package characteristics: It covers other constraints mentioned in users' questions, such as specific algorithms, environment requirements, etc.

Fig. 1 shows a query sample from ROS Answers. "Turtlebot2" represents the robot used; "Rplidar" refers to the sensor of the Rplidar series; "do SLAM" is the function of the required ROS package; and "Simulated" is the characteristic that limits this task to a simulated environment.

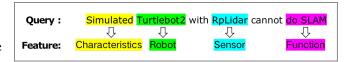


Fig. 1 A query sample from ROS Answers
(https://answers.ros.org/question/404381/
simulated-turtlebot2-with-rplidar-cannot-do-slam/)

In addition to the previously mentioned five key concepts, we have discovered that code files can also aid in locating a ROS package in certain cases. We study the code files of ROS packages to help search for suitable ROS packages. Fig. 2 demonstrates the file structure of a typical ROS package. In terms of the filesystem, a ROS package must contain a catkin compliant package.xml file, which provides meta-information about the package, and a CMakeLists.txt file which uses catkin. Codes of ROS nodes are often written in C++ or Python and are stored in separate /src and /scripts folders. Shell

<sup>&</sup>lt;sup>4)</sup>https://wiki.ros.org/Metapackages

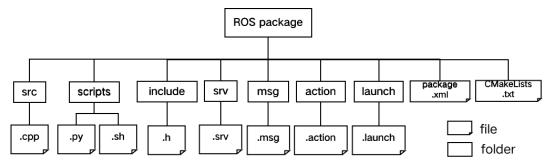


Fig. 2 Filesystem of the ROS package

scripts are also placed in the /scripts folder. Header files for C++ programs are placed in the /include folder. The /srv and /msg folders contain service files in .srv and message files in .msg format, respectively. The /action folder contains action files in .action format. Additionally, launch files with a .launch extension are used as configuration parameters of ROS nodes.

The modeled schema, depicting the abstraction of 11 entities, is illustrated in Fig. 3. Concepts related to code files include action, node, service, message, and launch.

# 3.2 ROS Package Relationships

Alongside the entities, we have identified 11 key relations between ROS packages and other entities. However, for the purposes of this paper, we choose to ignore the dependency relation between packages, as it mainly contributes to software configuration.

- **is\_for\_device**: Some packages have limitations in hardware devices. Package A may only be applied to Robot R.
- is\_in\_category: Different packages have different categories, such as meta package, message package, or description package. There are also differences between them in tasks, such as visualization, navigation, or manipulation.

- has function: A package has its own functions, such as implementing a mapping algorithm or driving a laser.
- has\_characteristics: Each package has some characteristics.
- includes\_\${file type}: Generally, a ROS package is composed of nodes, services, messages, launch files, a Cmakelist file, and an XML file. As the Cmakelist file and package XML file have fixed names, they are useless for retrieving ROS packages and are removed from RPKG.

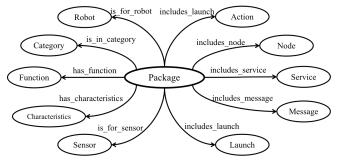
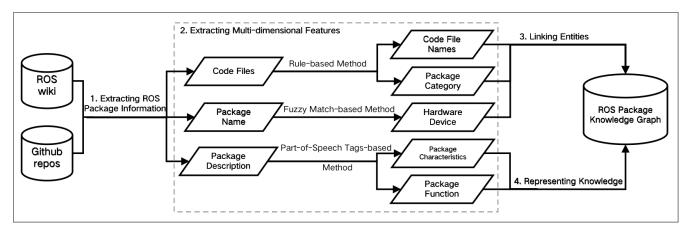


Fig. 3 Conceptual Schema of ROS Package Knowledge Graph

# 4 RPKG Construction

Fig. 4 presents four key steps for constructing RPKG from ROS wiki and GitHub repositories. We first design a conceptual schema to represent the entities and relations required to search ROS packages. Then we use the popular web crawler tool Scrapy4



**Fig. 4** The overview of construction approach for RPKG

to crawl ROS wiki tutorials and GitHub repositories. Next, we parse each package to retrieve its code files, package name, and package description. We propose a multi-dimensional feature extraction technique to process the following:

- Code Files: Extracted with a rule-based method to retrieve package category and names of nodes, messages, launch files, actions, and services.
- Package Name: Extracted with a fuzzy match method to retrieve the hardware device.
- Package Description: Extracted with a dependency syntactic parsing method to retrieve its function and characteristics.

In order to facilitate the search of ROS software packages, we adopt different knowledge representations for different features. For extracted features on code file names, package categories, and hardware devices, they can be easily retrieved by names. So we link them by names to entities in the ROS package knowledge graph and create corresponding relationships according to the conceptual schema in Fig. 3. For features on package characteristics and package function, they are usually phrases in natural language and their semantics are difficult to retrieve directly from text. Hence, we further fine-tune a BERT model to represent

knowledge of these features. Finally, using the features extracted earlier, we construct the RPKG.

# 4.1 Extracting ROS Package Information

As the ROS ecosystem continues to develop, a wealth of knowledge about ROS packages has emerged across various sources, including research papers, books, courses, and online websites. However, the collection of knowledge from these distributed sources poses a significant challenge. The ROS organization opens source a GitHub project "rosdistro" to maintain a list of repositories for each ROS distribution. From "rosdistro", we can retrieve the package names, repository URLs, and all official ROS packages included in each repository, which can help us locate the code files of ROS packages on GitHub. Fig. 5 shows the "turtlebot" repository in GitHub. The "turtlebot" repository includes 5 ROS packages related to the TurtleBot2 robot.

To maintain the documentation of ROS packages, the ROS organization applies the wiki system to encourage all developers to contribute their ROS packages to the ROS official wiki website https://wiki.ros.org. Fig. 6 shows the wiki page for the ROS package "turtlebot\_bringup". In the red

<sup>5)</sup>https://github.com/ros/rosdistro

Fig. 5 The "turtlebot" repository in GitHub (https://github.com/turtlebot/turtlebot)

box, we can see the package description "turtle-bot\_bringup provides roslaunch scripts for starting the TurtleBot base functionality.". It also provides information about the package's maintainer status, maintainer, author, license, bug or feature tracker, and source URL. Package links on the right of the red box show their dependency relations with other ROS packages. Below the red box are tutorials and launch files for the package, which may be outdated knowledge as the wiki page is not synchronized with the code repository in real-time.

After parsing the ROS packages from ROS wiki tutorials and GitHub repositories, we retrieve the code files, package names, and package descriptions for each ROS package.

## 4.2 Extracting Multi-dimensional Features

Since the knowledge regarding the categories and contents of ROS packages follows a structured format, we can apply rules to extract these features, including category, node, service, message, launch, and action. However, when dealing with features related to robot and sensor hardware, which often have various abbreviations, direct matching becomes challenging. To address this, we employ a

fuzzy match-based method to enhance the extraction of hardware features from the package name. Package descriptions typically provide information about the function and characteristics of packages in natural language, which is unstructured. To extract relevant features from these descriptions, we utilize part-of-speech (POS) tags to identify and extract relevant phrases as features on function and characteristics.

#### 4.2.1 Rule-based Method

Packages differ from each other in file structures. So we can extract the package category by applying the following four rules:

- Rule 1: If a package only includes basic package.xml and CMakeLists.txt without any ROS node related files, then it is a meta package.
- **Rule 2:** If a package includes /meshes or /robots folder to describe a robot model, then it is a description package.
- Rule 3: If a package includes /msg folder to declare ROS messages, then it is a message package.
- **Rule 4:** If the previous three rules are not met, then the package is a function package.

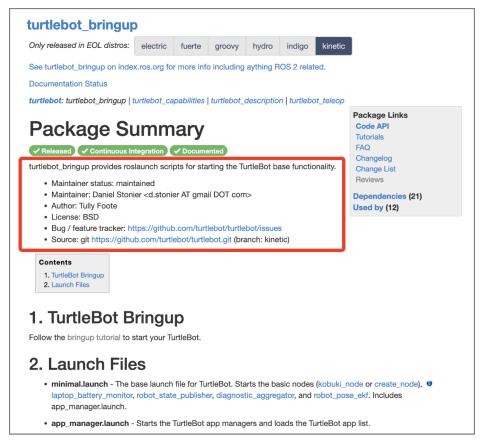


Fig. 6 The wiki page for the ROS package "turtlebot\_bringup" (https://wiki.ros.org/turtlebot\_bringup)

For code files in each package, we can extract code file names by applying the following four rules:

- Rule 5: If a file is in the "/scripts" folder and its name ends with ".py", then it is a Node file.
- Rule 6: If a file is in the "/srv" folder and its name ends with ".srv", then it is a Service file.
- Rule 7: If a file is in the "/msg" folder and its name ends with ".msg", then it is a Message file.
- Rule 8: If a file is in the "/action" folder and its name ends with ".action", then it is an Action file.

A ROS node can be compiled from C++ code files ending with ".cpp" with "CMakeLists.txt". So we can apply regular expressions to retrieve names

of ROS nodes from "CMakeLists.txt".

With the rule-based method, we can retrieve package categories and code file names as features of ROS packages.

# 4.2.2 Fuzzy Match-based Method

Each package name uniquely refers to a ROS package. The name of a ROS package is important as it contains key features of the package. For example, we can determine from the package name "turtle-bot3\_msgs" that it is specific to the Turtlebot3 robot and it is a ROS message package, indicating that it is used for defining message types for the Turtlebot3 robot.

In 2015, the ROS organization provided guidelines and rules for package naming (ROS REP 0144)<sup>6)</sup>,

<sup>6)</sup>https://ros.org/reps/rep-0144.html

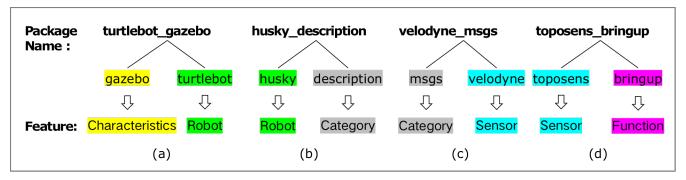


Fig. 7 Naming structures of 4 ROS packages

which also assist in extracting features from package names. Essentially, each package name uses "\_" separators among feature words. If the package is specialized for a hardware device, the word before the first "\_" separator usually refers to that device. For example, Fig. 7 shows the naming structure of four ROS packages. In the case of the "turtlebot\_gazebo" package, "turtlebot" indicates that Sensors. Then, we compute a sorted fuzzy simiit is suitable for the Turtlebot2 robot and "gazebo" indicates that it works in the Gazebo simulator. "velo- first word in the package name. If the highest score dyne\_msgs" provides ROS message definitions for Velodyne (a 3D lidar sensor). The "msgs" part signifies that it belongs to the category of message packages. "toposens\_bringup" is used for bringing up the Toposens ultrasonic 3D sensor. Here, "bringup" refers to the function feature of this package.

To determine the related hardware device, we need to analyze the first word in the package name. However, this analysis is challenging due to the absence of an organized list of existing hardware devices in robotics. Moreover, abbreviations are common in package naming, and package developers have different naming styles. In previous works [15], a fuzzy match-based method was proven effective in dealing with abbreviations. In this study, we apply the fuzzy match-based method to recognize the entities of robots and sensors. Firstly, we collect existing known robots and sensors as a vocabulary for fuzzy matching. We extract features of known robots from official ROS website for robots https://robots.ros.org/, which includes the robot name, robot category, website URL, wiki page URL, robot description, and related tags. Sensor names and related packages can be retrieved from ROS wiki for sensors https://wiki.ros.org/ larity score list between all hardware names and the exceeds the threshold, then we consider the corresponding hardware as the related hardware entity for the package.

To implement the fuzzy match, we utilize the Levenshtein Distance algorithm to calculate the similarity score between two words. The threshold value has an impact on the accuracy of recognizing robot and sensor features. If the threshold is set too high, many positive feature words may be missed in the fuzzy match process, resulting in a low precision rate. Conversely, if the threshold is set too low, some incorrect words may be erroneously recognized as hardware features, leading to a low recall rate. Through testing, we determine the threshold of 90 to achieve the best F1 score, striking a balance between precision and recall rates.

The fuzzy match-based extraction method consists of four steps:

• Step 1: Split the ROS package name to ob-

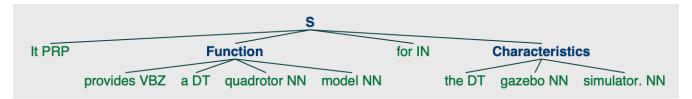


Fig. 8 Conceptual Schema of ROS Package Knowledge Graph

tain the first word.

- **Step 2**: Calculate the similarity scores of the word within the hardware vocabulary.
- **Step 3**: Sort the similarity scores in descending order and identify the hardware name with the highest score.
- **Step 4**: Add the hardware feature to the ROS package if its score exceeds the threshold of 90.

# 4.2.3 Part-of-Speech Tags-based Method

As the package descriptions are unstructured text, it is challenging to directly extract features from them. Inspired by the TaskKG work of Sun et al. [14], we leverage Part-of-Speech (POS) tags to extract noun phrases (NP) and verb phrases (VP) from package description sentences as characteristics and functions of ROS packages. The regular expressions for parsing POS tags generated from the package descriptions are shown in Table 1.

 Table 1
 Regular expressions for extracting features.

Feature	Regular Expression
Function	(VB.*) + (CD) * (DT)?(CD) * (JJ) *
	(CD)*(VBD VBG)*(NN.*)*(POS)*
	(CD)*(VBD VBG)*(NN.*)*(VBD VBG)*
	(NN.*) * (POS) * (CD) * (NN.*) +
Category	(CD) * (DT)?(CD) * (JJ) * (CD) *
	(VBD VBG)*(NN.*)*(POS)*(CD)*
	$(VBD VBG) \ * \ (NN.*) \ * \ (VBD VBG) \ *$
	(NN.*) * (POS) * (CD) * (NN.*) +

Fig. 8 shows an example of parsing the descrip-

tion sentence of the "hector\_quadrotor\_gazebo" ROS package. Each word in the sentence is tagged with a part of speech (in green color). By utilizing the previous regular expressions, we can extract the features for function and characteristics. In this example, "provides a quadrotor model" and "the gazebo simulator" are extracted as the function and characteristics, respectively.

With the POS tags-based method, we can retrieve both the characteristics and function features of ROS packages.

# 4.3 Linking Entities

With the extracted features, we can create various entities and relations to the RPKG. For features on robot, sensor, category, function and characteristics, they can be uniquely defined by their names. If a feature name matches an entity in the knowledge graph, we can directly link this feature to that entity. If not, we create a new entity with that feature name. For features on action, node, service, message and launch, the same feature name may refer to files with different contents. Therefore, we create a new entity for each feature name.

# 4.4 Representing ROS Package Knowledge

All current search approaches for ROS packages represent ROS package and its related features in text and search ROS package by matching text in syntax. They lack understanding and exploitation of the semantics of ROS packages, which results in

Phrase#1	Phrase#2	Cosine Similarity of Embeddings		
1 III ase#1	1 III asc#2	BERT-Base	BERT-ROS	
Turtlebot	Turtlebot2	0.834	0.954	
Turtlebot	Turtlebot3	0.831	0.936	
Turtlebot2	Turtlebot3	0.931	0.915	
rplidar	rospeex	0.796	0.682	
bringup rplidar	start rplidar sensor	0.868	0.885	
MoveIt framework	MoveIt library	0.898	0.901	
Gazebo simulator	Gazebo simulation environment	0.864	0.881	
OMPL	Open Manipulator Planning Library	0.658	0.779	

Table 2 Comparisons of cosine similarity of phrase embeddings between BERT-Base and BERT-ROS

their weakness when dealing with synonyms and abbreviations, and unsatisfactory performance in search accuracy. BERT is one of the most pretrained language models in Natural Language Processing (NLP) [22]. It provides a representation for the semantics of text by encoding text to an embedding in high-dimensional vector spaces. For texts with similar semantics, their embeddings are also close in high-dimensional vector spaces, and vice versa. However, the BERT-Base model fails to represent robot domain terms properly into embeddings as it is adapted to general domains and its training corpus barely covers the robotics domain. For example, the word "rplidar" is often used to refer to a type of lidar sensor. As it is not covered by the vocabulary of the BERT-Base, it will be tokenized into four word tokens, including "r", "##pl", "##ida" and "##r", and then encoded into an embedding that does not properly represent its semantics.

We further fine-tune BERT-Base for a better embedding representation model for the ROS domain, called BERT-ROS. This involves mainly three steps:

- Collect the robot context from ROS.org to build a corpus in the robot domain.
- the robotics domain terms.

• Further fine-tune BERT-Base for Masked Language Model (MLM) tasks to build BERT-ROS, which can be used to encode text with robotics domain terms into embeddings.

Table 2 shows the cosine similarities of the embeddings for phrases with BERT-Base and BERT-ROS. The phrases "Turtlebot" and "Turtlebot2" are often used to refer to the second generation of the Turtlebot robot in robotic context. After fine-tuning, BERT-ROS can better distinguish them from "Turtlebot3", the third generation of Turtlebot. "rplidar" and "rospeex" are two different domain terms both in syntax and in semantics so the similarity computed with BERT-ROS is lower than with BERT-Base. For synonym phrases like "bringup rplidar" and "start rplidar sensor", BERT-ROS also performs better. "OMPL" is an abbreviation for "Open Manipulator Planning Library". They are closer in embeddings representation with BERT-ROS. With improved semantic representation, the BERT-ROS model will contribute to ROS package search.

# **RPKG-based ROS Package Search**

As shown in Fig. 9, there are three steps to imple-• Extend the tokenizer from BERT-Base to cover ment the RPKG-based search for ROS packages. In the first step, feature extraction is the same as in



Fig. 9 RPKG-based ROS Package Search

Section 4.2. The second step is computing similarity scores between the query and all ROS packages, which is explained in detail in Section 5.1. We explain the third step in Section 5.2.

We design a fusion search algorithm that combines word-vector similarity matching and fuzzy matching to support the search of ROS packages based on RPKG. Word vectorization provides a better representation in terms of semantics, but it is not suitable for domain words in ROS packages as most of them are not common words and abbreviations are often used. On the other hand, fuzzy matching performs well in word syntactic matching, especially for domain words [15]. Hence, we combine these two methods by using vector similarity to match general functional feature words and using the fuzzy matching method to match special domain words in the descriptions of ROS packages. We rank the searched ROS packages by the weighted sum of the vector similarity score and fuzzy matching score. The algorithm is listed in Algorithm 1.

Table 3 shows the query template we design for users to describe their requirements for ROS packages from multiple dimensions.

We formalize the ROS package P as a set of features: robot R, sensor SS, category CG, function F, characteristics CR, action A, node ND, service SV, message M, and launch L. The user query Q is also formalized in the same way.

$$P = \langle R, SS, CG, F, CR, A, ND, SV, M, L \rangle$$

$$Q = \langle R, SS, CG, F, CR, A, ND, SV, M, L \rangle$$

**Table 3** The query template for ROS packages.

Features	Hints for User Input
Robot	What robot do you use?
Sensor	What sensor do you use?
Category	What is the category of the package?
Function	What is the expected function of the package?
Characteristics	List the characteristics of the package, separated by commas
Action	What is the action name in the package?
Node	What is the node name in the package?
Service	What is the service name in the package?
Message	What is the message name in the package?
Launch	What is the launch name in the package?

# 5.1 Similarity Calculation Method

For robot and sensor features, we compute the Levenshtein Distance as the similarity metric:

$$SIM(R_p, R_q) = \max_{i \in R_p} (LD(i, R_q))$$
 (1)

For function and characteristics features, we compute the average Cosine Distance as the similarity metric:

$$SIM(F_p, F_q) = \frac{1}{|F_q|} \sum_{v_j \in F_q} \max_{v_i \in F_p} (\frac{v_i \cdot v_j}{|v_i||v_j|})$$
 (2)

For features in category, action, node, service, message, and launch, we compute the average of the summed Sign Function as the similarity metric:

$$SIM(ND_p, ND_q) = \frac{1}{|ND_q|} \sum_{j \in ND_q} \sum_{j \in ND_p} Sgn(i = j)$$
(3)

We compute the weighted sum of similarities of multiple features as the similarity between each package and the query. The calculation of similarity is described in Eq. 4.  $\omega_k$  represents the weight  $(0 < \omega_k <= 1)$  for the kth feature in *P*.

$$SIM(P,Q) = \frac{1}{\sum_{k \in [0,9]} \omega_k} \sum_{k \in [0,9]} \omega_k SIM(P_k, Q_k)$$
 (4)

# 5.2 Feature Fusion Search Algorithm

We design the Feature Fusion Search Algorithm (FFS) to search the most related ROS packages, which is shown in Algorithm 1.

# **Algorithm 1** Feature Fusion Search (FFS)

**Input**: Query features: Robot R, Sensor SS, Category CG, Function F, Characteristics CR, Action A, Node ND, Service SV, Message M and Launch L

Output: Top K most suitable packages

# **Procedure:**

- 1:  $Q \leftarrow \langle R, SS, CG, F, CR, A, ND, SV, M, L \rangle$
- 2: Initialize S as an empty dictionary
- 3: Initialize *R* as an empty list
- 4: **for** package *P* in Package Set **do**
- $score \leftarrow SIM(P, Q)$
- $S[P] \leftarrow score$
- 7:  $R \leftarrow \text{Sort all packages in } S \text{ by } score \text{ in de-}$ scending order
- 8: **return** first K packages in R

In FFS, we first extract the features of a user query from the input. Then, we traverse all ROS packages and calculate the similarity score for each package. Finally, we sort all ROS packages by scores in descending order and output the top K packages as search results.

# **Evaluation**

The constructed RPKG has 32,294 entities (11 types) Similar to previous studies [15,23–25], we adopt a and 54,698 relationships (10 types), which can de-

scribe ROS packages from 10 dimensions and enrich the semantics of ROS packages. The structured knowledge representation provided by RPKG and the embedding representation of knowledge text are more convenient for supporting the subsequent search for ROS packages.

We conducted evaluations to explore the following research questions:

- **RQ1** (Quality): How is the intrinsic quality of the knowledge in the constructed RPKG? This research question aims to evaluate the accuracy of the knowledge encapsulated within RPKG.
- **RQ2** (Accuracy): How does RPKG perform in search accuracy for ROS package compared with other methods? This research question aims to assess the accuracy of our proposed RPKG-based search method for ROS packages.
- **RQ3** (Necessity): To what extent does the removal of individual feature dimension impact the accuracy of our search method? The aim of the ablation analysis is to investigate the necessities of summarized feature dimensions for ROS package search.

# RQ1: Quality of the Constructed RPKG

Since knowledge of category, node, service, message, and launch is extracted from the structured code file system, it is inherently accurate. Our quality evaluation focuses on the robot, sensor, function, and characteristics related to the ROS package.

#### **Experiment Setup** 6.1.1

sampling method [26] to ensure that the ratios ob-

		•		
Aspect	Accuracy#1	Accuracy#2	Accuracy#F	Kappa Agreement
Hardware	95.8%	96.9%	95.8%	0.852
Function	93.0%	90.1%	91.6%	0.898
Characteristics	90.3%	92.5%	93.0%	0.748

Table 4 Accuracy of ROS Package Features

served in the sample are representative of the population within a 95% confidence level, with a confidence interval of 5.

We randomly select 96 hardware devices, 344 functions, and 371 characteristics to conduct the experiment. Two developers (who are not involved in this study and are familiar with ROS) independently perform the examination. Given a ROS package and a feature in each round, developers are required to determine whether this feature is related to the package in a specific dimension. All decisions are binary (the accuracy rates are Accuracy#1 and Accuracy#2 respectively). For the data instances where the two annotators disagree, they have to discuss and come to a consensus. We compute the final accuracy after resolving the disagreements (Accuracy#F) and compute Cohen's Kappa agreement (Kappa Agreement) [27] to evaluate the inter-rater agreement. Based on the consensus annotations, we evaluate the quality of the knowledge created about the ROS package.

#### 6.1.2 Results and Analysis

The results are shown in Table 4. We can see the agreement rate are all above 0.74, indicating substantial or almost perfect agreement between the

Typical problems of ROS package hardware extraction include: (1) mismatched semantics, for exwas incorrectly matched to the "pointgrey camera".

(2) Incomplete hardware list, for example, the package "baxter\_gazebo" is developed for the Baxter robot. However, this robot is not included in the official ROS robot list, so the fuzzy match-based method will miss this feature.

Typical problems of ROS package function and characteristics extraction include: (1) POS tagging error, for the "audio\_play" package with the description "Outputs audio to a speaker from a source node.", "Outputs" is tagged as a noun and the phrase "Outputs audio" is recognized as a characteristics feature, when in fact it is a function feature. (2) Meaningless characteristics, "This package" is extracted as a characteristics feature from "This package provides launch files for operating Care-O-bot.". (3) Morphological difference, the noun phrase "face detection" from "Face detection in images." describes the function "detect face", but it is extracted as a characteristics feature.

The results also prove that our proposed multidimensional feature extraction technique is effective and can support the knowledge extraction for ROS packages.

# RQ2: Accuracy of the Search Method

To validate the enhanced accuracy of ROS package two annotators. The accuracy is generally high (above searching, we conducted comparative case studies, 0.91), which represents the high quality of our RPKG. comparing our semantic-based search approach with four baseline search approaches: ROS Index, GitHub, Google, and ChatGPT. Through these case studies, ample, the word "point" in "point\_cloud\_publisher\_tutorialgained valuable insights into the strengths and limitations of our approach.

## 6.2.1 Experiment Setup

To evaluate the effectiveness of our search approach, we selected 29 robot tasks from three ROS tutorial books (Mahtani, 2016 [28]; Fairchild, 2017 [29]; Gandhinathan, 2019 [30]), which cover common tasks in robotics, such as navigation and manipulation. These three referenced books provide rich ROS-based robot software development cases, aiming to guide newcomers to quickly get started with ROS.

As shown in Table 5, each task involves several subtasks. Each subtask requires a ROS package. We summarized 100 subtasks, involving 100 ROS packages. According to the template shown above in Table 3, we formulated a query based on the description for each subtask. For example, subtask#4-3 requires to "visualize Turtlebot2 with RViz", which was formulated into the query with " $\langle$  Robot:Turtlebot2, Function:visualize Turtlebot2, Characteristics:RViz $\rangle$ ". We know that the desired ROS package is "turtlebot\_rviz\_launchers".

For features on function and characteristics, we assigned a weight of 0.8 to achieve better search performance, while we set it to 1.0 for other feature dimensions. We set 5 search levels, with K values of 1, 5, 10, 15, and 20.

### 6.2.2 Baselines

We use ROS Index, GitHub, Google, and ChatGPT as the baseline tools. Each of them implements a different search algorithm.

 ROS Index: It uses the BM25 algorithm to calculate the relevance score between ROS packages and user queries. Designed to search ROS packages and repositories, ROS Index is the most commonly used search platform, especially for ROS newcomers.

- GitHub: It uses Elasticsearch as its search engine, which implements the inverted index algorithm to search data. Moreover, most of the official ROS packages are hosted on GitHub.
- Google: It uses PageRank and many other text match-based algorithms to search for relevant documents. To improve the search results, we can use the "site" keyword to limit its search scope to include only "ros.org" and "github.com".
- ChatGPT: It uses large-scale language models, including gpt-3.5-turbo, which greatly improves the ability to understand the semantics of texts and codes.

The inputs differ in format between the four baselines and our method. For example, if the search input consists of more than 3 words, then GitHub may not return any results. Hence, we determine the best input for each task through several trials, which helps us find the desired ROS packages.

## 6.2.3 Results and Analysis

Fig. 10 shows the comparisons on search accuracy between the existing four methods and our RPKG-based search method. For each query, if the desired ROS package appears in the top@K search results, then this set of K ROS packages is considered accurate. From this table, we can observe a significant improvement of up to 21%, which demonstrates that our method can find more accurate ROS packages. ChatGPT exhibits slightly higher accuracy than Google at four top levels due to its better semantic understanding.

We analyze those query cases and find the reason why our method performs better is that we greatly improve the representation and understanding of

 Table 5
 Statistics of 6 tasks including 20 subtasks from reference books

Task No.	Subtask Description	Query for ROS Package	Desired Package	
	1-1 Discover other ROS masters	Function: discover other ROS masters	master_discovery_fkie	
1	1-2 Synchronize the local ROS master	Function:synchronize the local ROS	master_sync_fkie	
		master		
	1-3 Use GUI to pilot the robot with Twist	Function:pilot the robot	nat nahat ataanina	
	message	Characteristics: GUI, Twist message	rqt_robot_steering	
	2.1 Materials for early installing	Sensor: Velodyne HDL-64E 3D lidar		
	2-1 Meta-package for easily installing	Function:install velodyne	velodyne_simulator	
2	Velodyne simulation components	Category:meta package		
	2-2 Description package for Velodyne lidar	Sensor: Velodyne HDL-64E 3D lidar	velodyne_description	
		Category:description package		
	2.2 Has Comple alumin to annuida simulated	Sensor: Velodyne HDL-64E 3D lidar		
	2-3 Use Gazebo plugin to provide simulated	Function:provide simulated data	velodyne_gazebo_plugin	
	data from Velodyne lidar	Characteristics: Gazebo plugin		
		Robot:Turtlebot2		
2	3-1 Simulate Turtlebot2 in Gazebo	Function:simulate Turtlebot2	turtlebot_gazebo	
3		Characteristics: Gazebo		
	220 111 16 F 41 0	Robot:Turtlebot2	441.141.11	
	3-2 Open dashboard for Turtlebot2	Function: open dashboard	turtlebot_dashboard	
	4.1.0	Robot:Turtlebot2		
	4-1 Start the Turtlebot2 robot	Function:start the TurtleBot2	turtlebot_bringup	
4	4-2 Use Turtlebot2 to create maps with	Robot:Turtlebot2		
4		Function: create maps	turtlebot_navigation	
	gmapping_demo.launch	Launch:gmapping_demo.launch		
		Robot:Turtlebot2		
	4-3 Visualize Turtlebot2 with RViz	artlebot2 with RViz Function:visualize Turtlebot2	turtlebot_rviz_launchers	
		Characteristics: RViz		
	4.4 m. 1. m. 1.1. 1.0. 1.1. 1.1.	Robot:Turtlebot2		
	4-4 Teleoperate Turtlebot2 with joysticks or	Function:teleoperate Turtlebot2	turtlebot_teleop	
	keyboard	Characteristics: joysticks, keyboard		
	4-5 Save the map	Function:save the map	map_server	
	5-1 Package that includes tools for Bebop	Function:include tools	1.1 1	
	robot	Characteristics:Bebop	bebop_tools	
5		Characteristics:Bebop	bebop_msgs	
	5-2 Message package for Bebop robot	Category:message package		
	5-3 Package that includes	Characteristics:Bebop		
	bebop_driver_node node	Node:bebop_driver_node	bebop_driver	
	5-4 Description package for Bebop robot	Characteristics:Bebop	bebop_description	
		Category:description package		
		Function:start the setup helper	moveit_setup_assistant	
_	6-1 Start the setup helper for MoveIt	Characteristics: MoveIt		
6		Function:provide base markers	aruco_ros	
	6-2 Provide base 3D markers	Characteristics:3D marker		
		Function:start the server	moveit_simple_grasps	
	6-3 Start the server for grasp with MoveIt	Characteristics: grasp, MoveIt		

ROS packages by extracting richer semantic features. For those cases where our method performs poorly, to sum up, the first reason is that the pretrained language model we fine-tuned, BERT-ROS, has limited semantic representation and understanding capabilities, and its understanding of user query intentions is not as good as that of ChatGPT. The second reason is that certain ROS software packages have not been given relevant descriptions by developers, which results in few features extracted. In the next step, we plan to combine large language models such as GPT3.5 represented by ChatGPT and knowledge graphs to enhance semantic representation and understanding for ROS package search Five feature dimensions (robot, sensor, category, task.

Table 6 Comparison of search accuracy on different sample sizes

Sample Size	Top1	Top5	Top10	Top15	Top20
10	0.70	0.80	0.80	0.90	0.90
20	0.65	0.70	0.75	0.75	0.75
30	0.50	0.73	0.73	0.80	0.80
40	0.63	0.73	0.78	0.80	0.80
50	0.58	0.72	0.80	0.80	0.82
60	0.55	0.73	0.80	0.82	0.82
70	0.56	0.71	0.77	0.79	0.80
80	0.58	0.73	0.76	0.78	0.79
90	0.60	0.72	0.79	0.81	0.82
100	0.56	0.70	0.76	0.78	0.79

Table 6 shows the comparisons of search accuracy on different sample sizes. We can find that the variations in search accuracy range between 0.1 and 0.2 when the sample size is no more than 30. When the sample size exceeds 30, the variations in search accuracy are within 0.1 across 5 top levels, which indicates the robustness and consistency of our method. This consistent performance suggests that our method's search accuracy is relatively stable even when confronted with varying sample sizes. This finding reinforces our method's reliability and

reinforces our confidence in its suitability for practical applications.

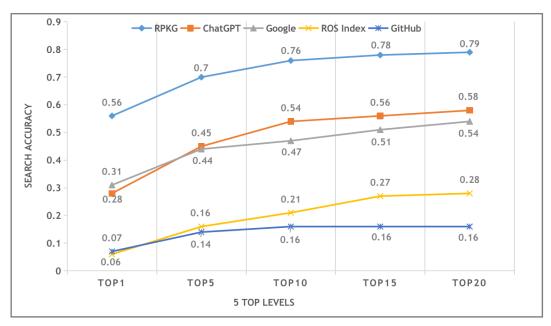
We conducted extensive experiments to evaluate the effectiveness of our proposed ROS package search approach. The results clearly demonstrate that our approach significantly outperforms existing methods in terms of search accuracy and relevance.

#### 6.3 RQ3: Necessity of Multi-dimensional Features

function, and characteristics) are summarized for searching ROS packages in section 4. In order to verify their necessity for our search method, we set up five individual ablation experiments to observe their impacts on the search accuracy for ROS packages. The code file features, including action, node, service, message, and launch, are inherent structural features of a ROS package, so there is no need to carry out ablation experiments for them. The ablation analysis serves to elucidate the significance of each feature dimension within our search method. This investigation enables us to quantify the impact of individual features on the accuracy of package retrieval, thereby providing a clearer understanding of the role that each dimension plays in enhancing the search effectiveness of our approach.

#### 6.3.1 **Experiment Setup**

We select five dimensions of features: robot, sensor, category, function, and characteristics. For each dimension, we construct two sets of queries: one with the inclusion of the specific feature and another with the exclusion of that feature. From a pool of 100 diverse query tasks in section 6.2, we



Comparison of search accuracy on 5 top levels

identified queries that contain each specific feature. We identified 15 queries with the "robot" feature, 16 queries with the "sensor" feature, 14 queries with the "category" feature, 52 queries with the "function" feature, and 82 queries with the "characteristics" feature.

For each of the five feature dimensions, we created two distinct query groups:

- that include the feature under consideration. For instance, the "robot inclusion" group consists of queries containing the "robot" feature.
- Exclusion Group: In contrast, the exclusion group contains queries where the specific feature is deliberately removed.

#### 6.3.2 Results and Analysis

For each query group within every feature dimension, we conduct a series of searches using our method We record the accuracy of search results for each query task in terms of whether the relevant ROS

package is successfully retrieved on 5 top levels.

Fig. 11 presents the accuracy variation of our search method before and after ablation for each of the five feature dimensions: robot, sensor, category, function, and characteristics. From the five sub-figures, it is evident that the exclusion groups • Inclusion Group: This group comprises queriesexhibit a noticeable decrease in search accuracy. Furthermore, it can be observed that the features "robot" and "sensor" have a more pronounced impact on the search method compared to other features. This suggests that these two features play a more crucial role in enhancing the effectiveness of ROS package search.

> We conducted ablation experiments to evaluate the necessity of our summarized five feature dimensions. The results clearly reflect the importance of the features we summarize, and their different degrees of influence on the search accuracy.

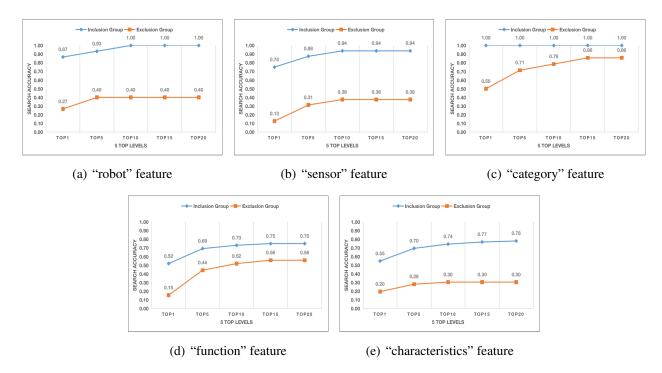


Fig. 11 Comparisons on search accuracy between inclusion group and exclusion group for five feature dimensions

# 7 Discussion

# 7.1 Implications

For researchers, they can build upon this work by further mining features of ROS packages from code files or optimizing search method. For developers, they can query related knowledge of certain ROS packages from RPKG or search related ROS packages for their robot software projects. For tool makers, this work can guide the design of new search tools for ROS packages.

# 7.2 Threats to Validity

A threat to internal validity is the subjective judgment in evaluation for the quality of extracted knowledge. To alleviate this threat, we have provided the agreement level for all subjective judgments. Another threat to internal validity is that our dictionary for identifying hardware features of ROS packages may not cover all the latest devices in

robotics, including many moving bases, sensors, and UAVs. In the future, we plan to launch an open project for collecting robotic hardware information and domain-specific words about algorithms, tools, etc., which will contribute to reducing this threat.

The major threat to external validity is the generalization of our experiment for our search method. To alleviate this threat, we are going to apply our approach to more open tasks related to searching ROS packages in Q&A platforms like ROS Answers and release our knowledge graph for public evaluations.

# **8** Conclusion and Future Work

In conclusion, this research article aims to address the problem of searching for ROS packages by proposing a comprehensive approach that integrates feature extraction, knowledge graph representation, semantic matching, and a search method based on weighted similarities.

Firstly, we introduce a novel feature extraction technique specifically designed for ROS packages. This technique allows us to extract multi-dimensional terns to support the development of robot software. features that capture the essential characteristics of the packages, enabling a more accurate and effective search.

Secondly, we construct a knowledge graph to store and represent these extracted features as knowledge. The knowledge graph provides a structured representation of the relationships between different ROS packages, facilitating the retrieval of relevant information and enhancing the search process.

Furthermore, to support semantic matching, we fine-tune a BERT model specifically tailored to the ROS domain. This model allows us to encode the textual information associated with ROS packages into meaningful embeddings, enhancing the understanding and similarity computation between packages.

Finally, we design a search method that leverages the computed sum of weighted similarities across multiple features. By assigning appropriate weights to different features, we can prioritize and find the most related ROS packages to users, ensuring that the found ROS packages are both relevant and useful.

Overall, our approach offers a comprehensive solution to the problem of searching for ROS packages, integrating feature extraction, knowledge representation, semantic matching, and a robust search method. Through our experiments and evaluations, we demonstrate the quality and accuracy of our proposed approach, showcasing its potential to enhance the discovery and utilization of ROS packages in various robotic applications.

As a next step, we intend to further study how to mine software design patterns from existing ROSbased robot software in ROS wiki tutorials, instructional books, and research papers. We also plan to further study how to utilize the mined design pat-

**Acknowledgements** This work was supported by the National Natural Science Foundation of China (Grant Nos. 62172426).

## References

- 1. Kolak S, Afzal A, Le Goues C, Hilton M, Timperley C S. It takes a village to build a robot: An empirical study of the ros ecosystem. In: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME). 2020, 430-440
- 2. Quigley M, Conley K, Gerkey B, Faust J, Foote T, Leibs J, Wheeler R, Ng A Y, others . Ros: an open-source robot operating system. In: ICRA workshop on open source software. 2009, 5
- 3. Pichler M, Dieber B, Pinzger M. Can i depend on you? mapping the dependency and quality landscape of ros packages. In: 2019 third IEEE international conference on robotic computing (IRC). 2019, 78-85
- 4. Cousins S, Gerkey B, Conley K, Garage W. Sharing software with ros [ros topics]. IEEE Robotics & Automation Magazine, 2010, 17(2): 12-14
- 5. Robertson S, Zaragoza H, others . The probabilistic relevance framework: Bm25 and beyond. Foundations and Trends® in Information Retrieval, 2009, 3(4): 333–389
- 6. Booth J D. Github succinctly, 2016
- 7. Estefo P, Simmonds J, Robbes R, Fabry J. The robot operating system: Package reuse and community dynamics. Journal of Systems and Software, 2019, 151: 226-242
- 8. Estefó P, Robbes R, Fabry J. Code duplication in ros launchfiles. In: 2015 34th International Conference of the Chilean Computer Science Society (SCCC). 2015, 1-6
- 9. Jiang G, Mao X. Exploring problems and solutions about launch files in ros from q&a community. In: Proceedings of the 2022 3rd International Conference on Big Data & Artificial Intelligence & Software Engineering. 2022, 9-14
- 10. Witte T, Tichy M. Checking consistency of robot software architectures in ros. In: Proceedings of the 1st International Workshop on Robotics Software Engineering. 2018, 1-8
- 11. Malavolta I, Lewis G A, Schmerl B, Lago P, Garlan D. Mining guidelines for architecting robotics software. Journal of Systems and Software, 2021, 178: 110969
- 12. Canelas P, Tavares M, Cordeiro R, Fonseca A, Timperley C S. An experience report on challenges in learning the robot operating system. In: 2022 IEEE/ACM

- 4th International Workshop on Robotics Software Engineering (RoSE). 2022, 33–38
- Macenski S, Foote T, Gerkey B, Lalancette C, Woodall W. Robot operating system 2: Design, architecture, and uses in the wild. Science Robotics, 2022, 7(66): eabm6074
- Sun J, Xing Z, Chu R, Bai H, Wang J, Peng X. Knowhow in programming tasks: From textual tutorials to task-oriented knowledge graph. In: 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME). 2019, 257–268
- Bo S, Mao X, Yang S, Chen L. Towards an efficient searching approach of ros message by knowledge graph. In: 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC). 2022, 934–943
- Tenorth M, Beetz M. Knowrob: A knowledge processing infrastructure for cognition-enabled robots. The International Journal of Robotics Research, 2013, 32(5): 566–590
- 17. ZOU Y, WANG M, XIE B, others . Software knowledge graph construction and q&a technology based on big data. Big Data Research, 2021, 7(1): 22–36
- 18. Seacord R C, Hissam S A, Wallnau K C. Agora: A search engine for software components. IEEE Internet computing, 1998, 2(6): 62
- López Pino B E. Sistema de recomendación de expertos para ROS Answers. Universidad de Chile, 2019
- Yang Z, Nyberg E. Leveraging procedural knowledge for task-oriented search. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval. 2015, 513–522
- Silva d R F G, Roy C K, Rahman M M, Schneider K A, Paixão K, Dantas C E d C, Maia M d A. Crokage: effective solution recommendation for programming tasks by leveraging crowd knowledge. Empirical Software Engineering, 2020, 25: 4707–4758
- 22. Kenton J D M W C, Toutanova L K. Bert: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of naacL-HLT. 2019, 2
- Li H, Li S, Sun J, Xing Z, Peng X, Liu M, Zhao X. Improving api caveats accessibility by mining api caveats knowledge graph. In: 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME). 2018, 183–193
- Ren X, Ye X, Xing Z, Xia X, Xu X, Zhu L, Sun J. Api-misuse detection driven by fine-grained api-constraint knowledge graph. In: Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. 2020, 461–472
- 25. Liu Y, Liu M, Peng X, Treude C, Xing Z, Zhang

- X. Generating concept based api element comparison using a knowledge graph. In: Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. 2020, 834–845
- Singh R, Mangat N S. Elements of survey sampling. volume 15. Springer Science & Business Media, 2013
- 27. Landis J R, Koch G G. An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers. Biometrics, 1977, 363–374
- Mahtani A, Sanchez L, Fernandez E, Martinez A. Effective robotics programming with ROS. Packt Publishing Ltd, 2016
- Fairchild C, Harman T L. ROS Robotics By Example: Learning to control wheeled, limbed, and flying robots using ROS Kinetic Kame. Packt Publishing Ltd, 2017
- Gandhinathan R, Joseph L. ROS Robotics projects: build and control robots powered by the Robot Operating System, machine learning, and virtual reality. Packt Publishing Ltd, 2019



Shuo Wang is a Ph.D graduate student in the College of Computer, National University of Defense Technology, China. His research interests include robotics software engineering, data mining

and knowledge graph.



Xinjun Mao is a professor in the College of Computer, National University of Defense Technology, China. He received his Ph.D. degree in computer science from the National University of Defense Technology, China in 1998.

His research interests include intelligent software engineering, multi-agent system, autonomous robot software, self-adaptive system, and crowdsourcing.



Shuo Yang is a lecturer in the College of Systems Engineering, National University of Defense Technology, China. He received his Ph.D. degree in software engineering from the National University of Defense Technology, China



in 2022. His research interests include robotics software engineering, automated planning, and component-based robotic frameworks.



Menghan Wu is a Ph.D. student at the College of Computer Science and Technology, Zhejiang University, China. Her current research interests include artificial intelligence for software engineering. Zhang Zhang is a Ph.D graduate student in the College of Computer, National University of Defense Technology, China. His work interests include open source software engineering, data mining, and crowdsourced learning.