Highlights

UERR: A Unified Effective Retrieval Model for Open-Source Repositories

Neng Zhang, Xin Gu, Jianga Shang, Haishen Lei, Chao Liu, Yiwang Huang, Zheng Lin, Bingnan Li

- A unified schema for characterizing open-source repositories.
- A query grammar for users to express personalized requirements.
- An effective model for retrieving repositories.

UERR: A Unified Effective Retrieval Model for Open-Source Repositories

Neng Zhang[a], Xin Gu[b], Jianga Shang[c,d], Haishen Lei[e,f], Chao Liu[g], Yiwang Huang[h,*], Zheng Lin[i], Bingnan Li[i]

[a]*School of Computer Science, Central China Normal University, Wuhan, China*
[b]*China Electronic Product Reliability and Environmental Testing Research Institute, Guangzhou, China*
[c]*Hubei Key Laboratory of Intelligent Geo-Information Processing, China University of Geosciences, Wuhan, China*
[d]*Engineering Research Center of Natural Resource Information Management and Digital Twin Engineering Software, Ministry of Education, Wuhan, China*
[e]*Fifth Electronic Research Institute of Ministry of Industry and Information Technology, Guangzhou, China*
[f]*Ministry of Industry and Information Technology Key Laboratory of Industrial Software Engineering Application Technology, Guangzhou, China*
[g]*School of Big Data and Software Engineering, Chongqing University, Chongqing, China*
[h]*School of Data Science, Tongren University, Tongren, China*
[i]*School of Software Engineering, Sun Yat-sen University, Zhuhai, China*

Abstract

Open-source repositories are widely used to improve the productivity and quality of modern software development. However, it is generally not an easy task for developers to find suitable repositories from a large-scale platform, e.g., GitHub. Although GitHub provides a search engine to support repository exploration, the search capability is limited as some important criteria (e.g., the closed ratio of issues) are not supported, and users cannot specify fine-grained preferences on different criteria. To address these limitations, we propose a unified effective repository retrieval model, named *UERR*. We first define a unified schema that is useful for retrieving repositories by analyz-

*Corresponding author.

Email addresses: nengzhang@ccnu.edu.cn (Neng Zhang), guxin@ceprei.com (Xin Gu), jgshang@cug.edu.cn (Jianga Shang), leihaishen@ceprei.com (Haishen Lei), liu.chao@cqu.edu.cn (Chao Liu), yjsyhyw@gztrc.edu.cn (Yiwang Huang), linzh228@mail2.sysu.edu.cn (Zheng Lin), libn23@mail2.sysu.edu.cn (Bingnan Li)

ing the metadata of GitHub repositories. Our schema includes 30 attributes grouped into six dimensions, such as *functionality*, *maintenance*, and *popularity*. Based on the schema, we design a query grammar for users to express their fine-grained requirements with personalized preferences and an integrated method to measure the relevance between a repository and a query. In particular, we expand acronyms and split conjoined words by leveraging large language models (LLMs) and design a weighted N-gram based method to measure functional relevance between textual descriptions of repositories and queries. Experiments with 20 diverse queries show that *UERR* significantly improves the top 1∼20 retrieved repositories by 27.12%-165.87% in terms of the three metrics: Pre@k, MRR@k, and NDCG@k, in comparison with the search engines provided by Elasticsearch and GitHub. The results also validate the superiority of our weighted N-gram based functional relevance measurement method over three representative methods.

Keywords: Open-Source Community, Unified Model, Repository Retrieval, GitHub, Large Language Model (LLM).

1. Introduction

To accelerate the sharing and development of software artifacts, many organizations and developers publish their artifacts in open-source communities [1, 2]. For example, as of 15 October 2025, the largest open-source community in the world, GitHub[1], has accumulated more than 420 million repositories. In modern software development, open-source repositories are widely used to promote the efficiency of the development process and improve the quality of software applications.

Existing open-source communities generally provide search engines to assist users in exploring their repositories by queries. A query can consist of multiple criteria (or conditional items) to support a combinatorial search. As described in the official documentation[2] of GitHub, users can search for repositories by repository name, description, size, visibility, etc. Fig. 1 presents the top-3 repositories returned by the GitHub search engine for a query containing five conditions: "*CAD in:name stars:>100 language:java language:python is:public*" (CAD is short for "Computer-Aided Design"). This query aims to

[1] https://github.com/
[2] https://docs.github.com/en/search-github/searching-on-github

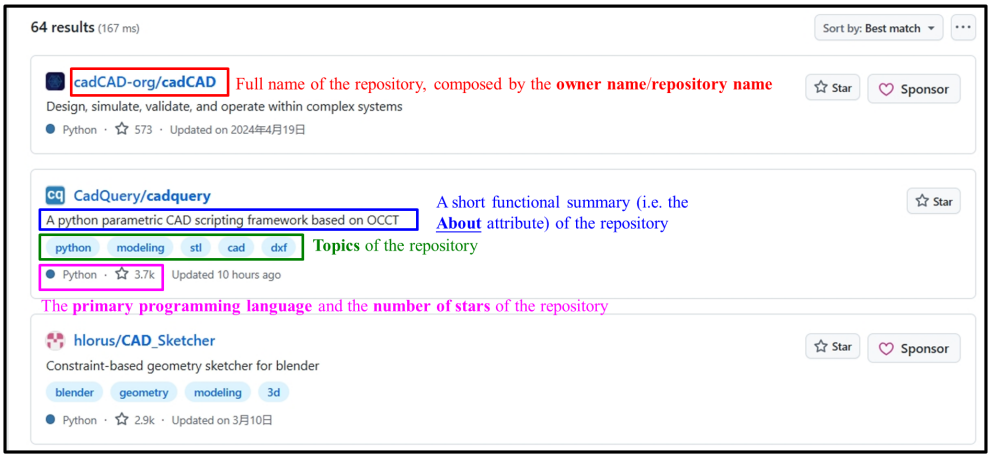

Figure 1: Top-3 repositories returned for a query by GitHub.

retrieve repositories satisfying four criteria: 1) the repository name contains the string 'CAD'; 2) the repository has received more than 100 stars; 3) the repository is primarily developed using Java or Python; and 4) the repository is public. From the documentation and the search results returned for several queries, we find that **the GitHub search capability has the following limitations**:

- **Only direct attributes (e.g., name and language) of a repository can be used in a query**, while some indirect attributes (e.g., the issue closure ratio (ICR)) of interest to users cannot be used as query conditions.
- **Users cannot specify personalized preferences on query conditions.** As for the example query above, the importance of the five query conditions can be different, e.g., "*CAD in:name*" should be more important than the others. However, there is no way for users to specify their preferences on the query conditions.
- **Unable to deal with slightly complex queries.** For example, only one repository "*ArivunidhiA/Digital-Twin-for-Supply-Chain-Scenario-Analysis*" of low quality (with unknown programming language (PL) and zero stars) is returned for the query "*digital twin for supply chain in:name*", while actually there are a number of repositories matching the query on GitHub.

Much work has been dedicated to recommending open-source repositories using various information from developers and repositories [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]. However, only a small number of studies [14, 15, 16] have been conducted to assist users in retrieving repositories by queries, which is a commonly used way to explore repositories in open-source communities. Moreover, existing repository retrieval approaches only considered part of attributes (e.g., PLs and the number of stars/forks) of repositories while neglecting many other information (e.g., license and ICR) that might be of interest to users. Therefore, **there still lacks a comprehensive method to help users effectively retrieve repositories from GitHub.**

To address the limitations of the GitHub search engine and existing repository retrieval methods, we propose a unified effective repository retrieval model, named *UERR*. Initially, we define a unified schema for characterizing open-source repositories by analyzing the metadata of repositories on GitHub. Our schema contains 30 attributes that are useful for users to search for repositories. The attributes are categorized into six dimensions, e.g., *functionality*, *maintenance*, and *popularity*, according to the aspects of repositories described by them. Based on the schema, we propose an integrated model to assist users in retrieving repositories. At first, we design a query grammar to guide users on how to express their requirements. A query can use any attribute in the schema associated with users' personalized preference. For a given query, *UERR* measures the relevance between the query and every candidate repository with respect to the query condition of each attribute. Particularly, we propose to expand acronyms and conjoined words by leveraging large language models (LLMs) and design a weighted N-gram [17] based method for measuring the functional relevance between the textual descriptions of a repository and the query. Finally, a comprehensive relevance is calculated for a repository by aggregating the relevance of all query conditions with the user's preferences.

To evaluate *UERR*, we create a relatively large dataset of 29,155 repositories from GitHub and conduct experiments with 20 diverse queries. The GitHub search engine and another popular search engine provided by the Elasticsearch (ES)[3] library are used as baselines. The results show that compared with the baselines, *UERR* improves the top 1∼20 recommended repositories by 27.12%-165.87% in terms of three popular metrics: Pre@k,

[3] https://www.elastic.co/elasticsearch

MRR@k, and NDCG@k. The effectiveness of our weighted N-gram based functional relevance measurement method is validated by comparing it with three representative methods, i.e., BM25, the Inverse-Document Frequency (IDF) [18]-weighted keyword matching, and the IDF-weighted word embedding matching.

The main contributions of this work are outlined below:

1) We define a unified schema of 30 attributes for characterizing open-source repositories on GitHub. The attributes are interested by users when they search for repositories.

2) We design a query grammar to help users express their requirements based on our unified schema. Users can add any attribute in the schema to a query and specify personalized preferences on the attributes.

3) We propose an effective model, *UERR*, for retrieving repositories relevant to a query. Particularly, in order to better measure the functional relevance between a repository and the query, we design an LLM-based method to expand acronyms and split conjoined words in their textual descriptions and also propose a weighted N-gram based functional measurement method.

4) We evaluate *UERR* by conducting experiments with 20 diverse queries in comparison with two popular search engines and three representative functional relevance measurement methods. The replication package of this study, including the experiment dataset and the source code of *UERR*, is publicly released at GitHub[4].

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 introduces the details of our work. Section 4 describes the experiments and results. Section 5 discusses threats to the validity of our work. Section 6 concludes the paper.

2. Related Work

2.1. Open-Source Repository Recommendation & Retrieval

Repository Recommendation. As open-source repositories have gained increasing popularity in software development, many studies [3, 4, 5, 6, 19,

[4] https://github.com/nengz/UERR

7, 8, 9, 10, 11, 12, 13] have been conducted on recommending repositories to developers. For example, Su et al. [3] proposed a recommendation model for GitHub repositories based on a self-attention mechanism considering user's historical operation sequence of repositories. Sun et al. [10] proposed a personalized approach for recommending GitHub repositories by integrating developers' behavior (i.e., create, fork, and star) and project features (i.e., the terms from the description and source code). Zhang et al. [12] also presented a personalized repository recommendation approach by extracting and modeling data related to developers and repositories using a deep learning technique. Lin et al. [4] defined an open-source weighted heterogeneous information network, called OSWHIN, to integrate four entities (i.e., developers, repositories, issues, and pull requests) and their relationships on GitHub. Then, a repository recommendation framework was proposed using an OSWHIN-based embedding method. Yu et al. [6] proposed a GitHub repository recommendation model for developers by combining two kinds of similarities: the repository similarity measured from a repository knowledge graph and the developer similarity measured from a developer-repository matrix. These approaches are proposed to recommend repositories for developers and cannot be applied to help users search for repositories by queries.

Repository Retrieval. In recent years, some work [14, 15, 16] has been done on the repository retrieval task. For example, Bissyande et al. [14] developed an integrated search engine architecture based on a knowledge database built by combining various metadata from multiple resources: source code, version control systems, bug tracking systems, and collaborative development platforms, and providing a query language for users to retrieve repositories. Wu et al. [16] implemented an open-source repository retrieval service by extracting structured functional semantics of repositories from six dimensions, such as PL, operating platform, and data format. Yue et al. [15] implemented a visual retrieval tool, VisRepo, for open-source repositories by mining repository data from four perspectives, namely topic, technology, usability, and comprehensibility, and employing a visualization technique to help users explore the data interactively. Although these approaches have made some progress in repository retrieval, they do not fully explore the meaningful information of repositories.

2.2. Attribute Recommendation for Open-Source Repositories

In spite of repository recommendation and retrieval, there has been a large amount of research work [20, 21, 22, 23, 24, 25, 26, 27, 28, 29] on the

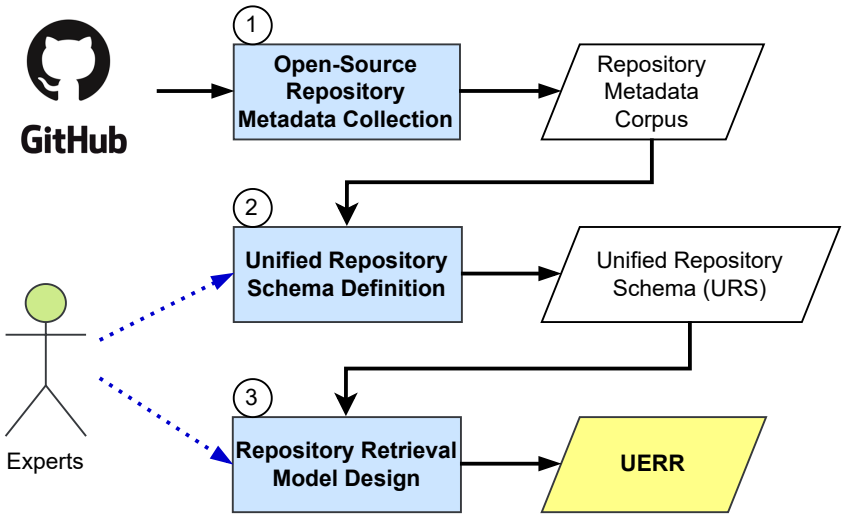

Figure 2: Overview of our research methodology.

recommendation of various attributes, e.g., developers, issues, reviewers, topics, and license, for open-source repositories. For example, Xuan et al. [26] proposed a ranking approach for recommending expertise developers to comment on bug reports by leveraging both the collaborative patterns of commenters and the specific content of bug reports. Samer et al. [21] introduced a recommendation approach that can support contributors of open-source communities in identifying relevant issues to work on. Izadi et al. [20] proposed two models for recommending topics to projects by incorporating the content of a repository and the semantic relationships among topics assigned to existing repositories. Zhang et al. [24] developed a machine learning-powered license recommendation method for open-source repositories based on their application scenarios. Some attributes considered in these approaches could be useful for users in retrieving repositories and are included in our unified schema and supported by our *UERR* model.

3. Research Methodology

Fig. 2 shows the overview of our research methodology. The details of the three main stages ① - ③ are described in the following subsections.

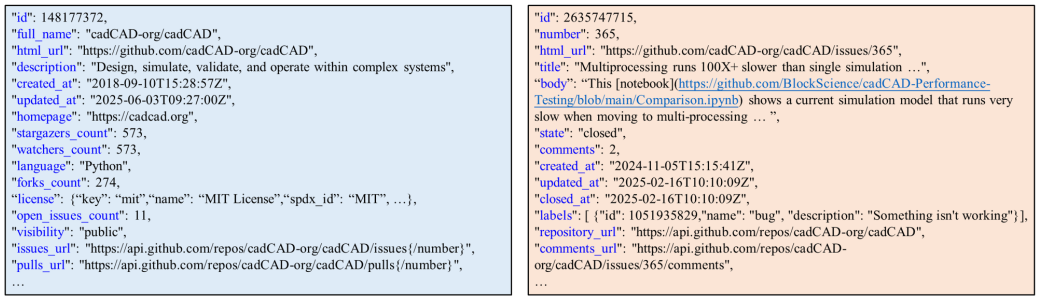

Figure 3: Part of the metadata of a repository and an issue of the repository.

3.1. Open-Source Repository Metadata Collection

In this first step, we collect the metadata of open-source repositories from GitHub. The metadata is used to define a **unified repository schema (URS)** that contains meaningful attributes for retrieving repositories. GitHub provides RESTful APIs[5] to access the data of repositories and their issues, pull requests, etc. We find that there are differences between the repositories' metadata. For example, most GitHub repositories have the attribute 'license', while some do not, e.g., "*FreeCAD/FreeCAD-library*[6]". To cover all possible attributes of GitHub repositories, we randomly sample 1,000 repositories from the set of 29,155 repositories collected for experiments (see Section 4) and then obtain their metadata. Fig. 3(a) shows part of the metadata of a repository, "*cadCAD-org/cadCAD*"[7]. By analyzing the metadata, there are 80 attributes in total. Seventy-seven attributes, e.g., 'full_name' and 'description', are possessed by all the repositories, while the three attributes 'homepage', 'language', and 'license' are missed by some repositories.

We also observe that there are a considerable number of attributes' values being URLs, e.g., 'issues_url'. The details of these attributes could be useful to users when searching for repositories. For example, if we know the total number of issues of a repository, we can compute the closed ratio of issues based on the attribute 'open_issues_count' shown in Fig. 3(a). This value should be cared about by users as it indicates the maintainability of a repository. Thus, we further obtain the details of these attributes by re-

[5] https://docs.github.com/en/rest?apiVersion=2022-11-28
[6] https://github.com/FreeCAD/FreeCAD-library
[7] https://github.com/cadCAD-org/cadCAD

questing their URLs using the corresponding APIs. Fig. 3(b) shows part of the attributes obtained for an issue.

3.2. Unified Repository Schema (URS) Definition

Based on the metadata of repositories and their URL-style attributes, we define a unified schema for GitHub repositories by identifying the information that users might consider for repository retrieval. For this purpose, we recruited three developers from the affiliation of the first co-author. The developers are interested in our study and have more than three years of experience exploring GitHub. As a preparation, we create a list of the 80 attributes with explanations. After introducing the task, we provided the attribute list to the developers and asked them to complete the task independently within one week. The developers are guided to identify attributes (including direct attributes and indirect attributes that can be derived from existing attributes) based on the importance for repository search, i.e., whether an attribute could affect their decision on the appropriate repository for a query. Moreover, they are allowed to search for any required information on the Web. From the results submitted by the developers, they identified 35 attributes in total; and there are eight attributes with disagreements. Then, we asked the developers to discuss their results together. Finally, they reached a common decision on a set of 30 attributes. Table 1 presents a brief description and the source of each attribute. The attributes have three kinds of sources:

- Seventeen attributes, e.g., 'full_name' and 'license', are obtained (or with a simple transformation) from the original attributes.
- Eleven attributes, e.g., 'languages' and 'total_issues_ count', are obtained from the response of URL-style attributes. For example, the issues of a repository are obtained by replacing '*/{number}*' in the URL of 'issues_url' with '*?state=all&page=i&per_page=100*'[8] where $i = 1, ..., m$; and m is the maximum page number which can be obtained from the response of the first page request. From the entire set of collected issues, we obtain the value of 'total_issues_count'.
- Two attributes, i.e., 'issue_closure_rate' and 'pull_request_completion _rate', are calculated using the values of other attributes. For exam-

[8] https://docs.github.com/en/rest/search/search?apiVersion=2022-11-28#search-issues-and-pull-requests

ple, the issue closure rate (ICR) of a repository, p, is computed as $ICR(p) = \frac{TIC(p)-OIC(p)}{TIC(p)}$, where $OIC(p)$ is the number of open issues of p, i.e., the value of 'open_issues_count'; and $TIC(p)$ is the total number of issues of p, i.e., the value of 'total_issues_count'.

To help users better understand the attributes, we categorize them into six dimensions according to the repository aspects described by the attributes:

- **Basic**. This dimension contains five attributes describing the basic information of a repository, including the 'full_name'[9], 'created_time', and 'homepage' of the repository, as well as the 'languages' (especially the 'main_language') used for the repository development.

- **Functionality**. This dimension contains three attributes that briefly describe the functionality of a repository, namely the 'about_description' (i.e., the original attribute 'description' in Fig. 3, which corresponds to the field 'About' on the webpage of the repository), 'readme_description' (extracted from the README file of the repository), and 'topics'.

- **Activity**. This dimension includes three attributes reflecting a repository's development activity: 'commits_count', 'releases_count', and 'last_update_time'. Generally, a repository has higher activity when it has more commits/releases or a more recent last update time.

- **Maintenance**. This dimension includes four attributes representing the total number of issues and pull requests (PRs) as well as their closure/completion rate. Generally, a repository with more issues/PRs and higher closure/completion rate means it has better maintenance.

- **Popularity**. This dimension comprises seven attributes that serve as indicators of a repository's popularity. Specifically, a repository's popularity is positively correlated with the number of user-assigned stars, developer-created branches/forks, and subscribers/watchers, and whether the repository has been downloaded.

[9]The full name of a repository may partially reflect its functionality. When retrieving repositories, it is suitable to use full_name as an attribute for measuring the functional relevance of a repository to a query.

- **Development Support**. This dimension comprises eight attributes that reflect the development support of a repository, including the number of contributors and collaborators, the license and visibility, the private fork permission (corresponding to the attribute 'allow_forking'), and the availability of the GitHub Projects[10], Wiki documentation[11], and GitHub Actions[12] features. Specifically, for a repository, its attribute 'allow_forking' can be true/false to allow/prevent private forks; it can use the GitHub Projects, an adaptable, flexible tool, to plan and track work; it can be equipped with a section called Wiki for hosting documentation to share long-form content, e.g., the usage, design, and core principles of the repository; and it can also enable the GitHub Actions, a continuous integration and continuous delivery platform, to allow users to automate the build, test, and deployment pipeline.

3.3. Repository Retrieval Model Design

Based on the URS, we propose a unified retrieval model, *UERR*, for open-source repositories, which contains two key parts: 1) **a query grammar** that introduces users on how to formulate a query; and 2) **a relevance measurement method** for calculating the relevance between a repository and a query.

3.3.1. Query Grammar

For each attribute in the URS, we provide a short identifier for it to be easily used as a ***query item*** and explain the setting constraint on the query item, as listed in Table 2. Apart from the query items corresponding to the 30 attributes, we provide two additional query items, FTA and FTAR, to help users effectively express functional requirements on the attributes: 'full_name', 'topics', 'about_description', and/or 'readme_description'. For example, if a user wants to search for repositories with 'CAD' appearing in any of the three attributes: 'full_name', 'topics', and 'about_description', he/she can simply create a query condition "*FTA:cad*". If a user wants to

[10] https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects

[11] https://docs.github.com/en/communities/documenting-your-project-with-wikis/about-wikis

[12] https://docs.github.com/en/actions/get-started/understand-github-actions.

search for repositories with 'CAD' appearing in any of the four attributes, he/she can create a query condition "*FTAR:cad*" for simplicity.

If there are multiple query items included in a query, the user can further express fine-grained preferences on the query items by assigning different weights. Formally, a ***query condition*** of a query item with a value setting and an optional preference weight can be expressed as

$$QI : value\_expression[: preference\_weight] \tag{1}$$

where QI is the short identifier of the query item; $value\_expression$ is the value setting expression of QI, which can be a single value (e.g., `true`, `20`, or `java`), a set of values (e.g., {`java,python`}), a range value (e.g., [`10,20`]), or a logical expression (e.g., >`20`), according to the setting constraint of QI (see Table 2); and $preference\_weight \in (0, 1.0]$ is an optional weight that can be assigned by the user to reflect his/her preference on QI. A higher weight indicates a higher preference. If QI does not have a weight, then it has a default weight of 0.5.

Multiple query conditions are connected by '&'. For example, suppose that Bob is a CAD developer. He wants to search for public repositories relevant to CAD, and the repositories need to be created after 2018-09-10, developed using Java or Python, licensed using 'MIT', and have received 100+ stars. Additionally, Bob has the following preference priority: *functionality* > *license* > *language* > *stars* > *visibility* > *others*. Finally, a query satisfying Bob's requirements and preference priority can be formulated as "*TP:cad:0.9 & FN:cad:0.9 & ADES:cad:0.9 & LIC:mit:0.8 & LAN:{java,python}:0.75 & StaC:>100:0.7 & VIS:public:0.6 & CT:(2018-09-10,)*". For simplicity, the three functional query conditions "*TP:cad:0.9 & FN:cad:0.9 & ADES:cad:0.9*" can be replaced by "*FTA:cad:0.9*".

3.3.2. Relevance Measurement Method

To apply our repository retrieval model, *UERR*, given a collection of repositories, R, a prerequisite step is to standardize each repository by obtaining or calculating the values of the 30 attributes defined in the URS (see Table 1). Moreover, to better measure the functional relevance between a repository and a query, we preprocess the values of the four attributes 'full_name', 'topics', 'about_description', and 'readme_description' of each repository using the following steps:

Table 1: Unified Repository Schema (URS)

No.	Dimension	Attribute	Description	Source
1	Basic	full_name	Full name of a repository	The repository's full_name attribute
2		created_time	Creation time of a repository	The repository's created_at attribute
3		homepage	Homepage of a repository	The repository's homepage attribute
4		main_language	Main programming language (PL) used by a repository	The repository's language attribute
5		languages	PLs used by a repository	Obtained from the response of the repository's languages_url
6	Functionality	about_description	A short summarized description of a repository	The repository's description attribute (corresponding to the *About* field on the repository's webpage)
7		readme_description	The readme description of a repository	Obtained from the README file of the repository
8		topics	A set of topics assigned to a repository	The repository's topics attribute
9	Activity	commits_count	Number of commits owned by a repository	Obtained from the response of the repository's commits_url
10		releases_count	Number of releases owned by a repository	Obtained from the response of the repository's releases_url
11		last_updated_time	Last updated time of a repository	The repository's updated_at attribute
12	Maintenance	total_issues_count	Total number of issues owned by a repository	Obtained from the response of the repository's issues_url
13		issue_closure_rate (ICR)	Issue closure rate of a repository	Calculated using the repository's total_issues_count (TIC) and open_issues_count (OIC), i.e., $ICR = \frac{TIC-OIC}{TIC}$
14		total_pull_requests_count	Total number of pull requests owned by a repository	Obtained from the response of the repository's pulls_url
15		pull_request_completion_rate (PRCR)	Pull request completion rate of a repository	Calculated using the repository's total_pull_requests_count (TPRC) and closed_pull_requests_count (CPRC) (obtained from the response of the repository's pulls_url), i.e., $PRCR = \frac{CPRC}{TPRC}$
16	Popularity	branches_count	Number of branches owned by a repository	Obtained from the response of the repository's branches_url
17		has_downloads	Whether the repository has been downloaded	The repository's has_downloads attribute
18		forks_count	Number of forks owned by a repository	The repository's forks_count attribute

Table 1 – continued from previous page

No.	Dimension	Attribute	Description	Source
19		owner_followers_count	Number of followers of a repository's owner (which can reflect the owner's popularity)	Obtained from the response of the owner's followers_url
20		stargazers_count	Number of stars owned by a repository	The repository's stargazers_count attribute
21		subscribers_count	Number of subscribers owned by a repository	Obtained from the response of the repository's subscribers_url
22		watchers_count	Number of watchers owned by a repository	The repository's watchers_count attribute
23	Development Support	allow_forking	Whether private forks are allowed by a repository	The repository's allow_forking attribute
24		contributors_count	Number of contributors owned by a repository	Obtained from the response of the repository's contributors_url
25		collaborators_count	Number of collaborators owned by a repository	Obtained from the response of the repository's collaborators_url
26		disabled	Whether the GitHub Actions workflow is disabled by a repository	The repository's disabled attribute
27		has_projects	Whether the GitHub Projects feature is enabled by a repository	The repository's has_projects attribute
28		has_wiki	Whether the Wiki documentation feature is enabled by a repository	The repository's has_wiki attribute
29		license	License of a repository	Obtained from the value of the 'spdx_id' key of the repository's license attribute (see Fig. 3)
30		visibility	Visibility of a repository	The repository's visibility attribute

Table 2: Query Items Defined Based on the URS (see Table 1)

No.	Attribute	Query Item	Setting Constraint	No.	Attribute	Query Item	Setting Constraint
1	full_name	FN	Any string	17	has_downloads	HasDown	true/false
2	created_time	CT	1) A complete/partial datetime, e.g., 2018, 2018-09, 2018-09-10, or 2018-09-10 15:28:57; 2) A datetime range, e.g., [2018-09,) or [2018-09,2018-12]	18	forks_count	FC	Similar to the setting of CC
3	homepage	HP	true/false, which means the homepage is accessible or inaccessible, respectively	19	owner_followers_count	OFC	Similar to the setting of CC
4	main_language	LAN	1) A PL, e.g., java;	20	stargazers_count	StaC	Similar to the setting of CC
5	languages		2) A set of PLs, e.g., {java,python}	21	subscribers_count	SubC	Similar to the setting of CC
6	about_description	ADES	Any string	22	allow_forking	AllowFork	true/false
7	readme_description	RDES	Any string	23	watchers_count	WatC	Similar to the setting of CC
8	topics	TP	Any string that can probably represent one or more topic keywords (e.g., 'cad' and 'cad,stl')	24	contributors_count	ConC	Similar to the setting of CC
9	commits_count	CC	1) An integer, e.g., 10; 2) An integer range, e.g., (10, 20]; 3) A logical integer expression, e.g., >10	25	collaborators_count	ColC	Similar to the setting of CC
10	releases_count	RC	Similar to the setting of BC	26	disabled	Disabled	true/false
11	last_update_time	LUT	Similar to the setting of CT	27	has_projects	HasProj	true/false
12	total_issues_count	TIC	Similar to the setting of CC	28	has_wiki	HasWiki	true/false
13	issue_closure_rate	ICR	1) A float, e.g., 0.5; 2) A float range, e.g., (0.5, 1.0]; 3) A logical float expression, e.g., >0.5	29	license	LIC	1) A license, e.g., MIT; 2) A set of licenses, e.g., {MIT, Apache License 2.0}
14	total_pull_requests_count	TPRC	Similar to the setting of CC	30	visibility	VIS	public/private/internal
15	pull_request_completion_rate	PRCR	Similar to the setting of ICR	31	–	FTA	Any string
16	branches_count	BC	Similar to the setting of CC	32	–	FTAR	Any string

1) **Non-text Content Cleaning.** We clean up non-text content from the 'about_description' and 'readme_description', including code snippets, scripts, images, markdown tags, and URLs, as such content is often noise and ignored by exiting repository retrieval methods.

2) **Tokenization, Conjoined Word Splitting & Acronym Expansion.** There are conjoined words and acronyms exist in the 'full_name', 'topics', 'about_description', and 'readme_description', which will affect the functional matching between a repository and a query. To solve these problems, we first tokenize the four attributes by hyphens ('-'), dots ('.'), underlines ('_'), slashes ('/'), and spaces. Then, we split conjoined words and expand acronyms using LLMs. In particular, we adopt the `deepseek-chat` model[13] which corresponds to the version of DeepSeek-V3.2 during this study. The two prompts used for conjoined word splitting and acronym expansion are shown in Fig. 4. For instance, the full name of a repository, "*realthunder/FreeCAD_assembly3*", is transformed to "*real thunder free computer aided design assembly 3*" after performing this step.

3) **Stopword Removal.** We remove stopwords from the token sequences (obtained after step 2)) based on the stopword list provided in NLTK[14].

4) **Stemming.** We reduce each token to its root form (aka. *stem*) using the `SnowballStemmer` module of NLTK, thus eliminating morphologically varied words. For example, 'create', 'creation', 'creating', and 'created' are all stemmed to 'creat'.

Considering that there can be tens of millions of repositories in R, to promote the efficiency of repository retrieval for a query, we index all preprocessed repositories using the Elasticsearch (ES) library.

After preprocessing and indexing the repositories, for a given user query, e.g. q, expressed according to the query grammar (see Section 3.3.1), our *UERR* retrieves a list of repositories satisfying q as follows. At first, the input query string is parsed and represented as $q = \wedge C_i$, where C_i is a query condition as demonstrated in Eq. 1. Then, *UERR* needs to efficiently obtain a candidate set of repositories, $R(q)$, from R. This task can be achieved

[13] https://api-docs.deepseek.com/
[14] https://www.nltk.org/

1) Prompt used for conjoined word splitting
Please split multi-word tokens and camel case tokens in the following text into individual words. For example: "helloWorld" should be split into "hello World", "deepLearning" into "deep Learning". Output only the processed text without additional explanation. **Text**: **{text to be splitted}**
2) Prompt used for acronym expansion
Please expand common abbreviations (e.g., cad, nlp, etc.) in the following text into their full forms. Output only the processed text without additional explanation. **Text**: **{splitted_text to be expanded}**

Figure 4: Prompts used for LLM-based conjoined word splitting and acronym expansion.

using the ES search engine based on the ES index built for the preprocessed R. However, we find that it is not a good choice to obtain the candidate repositories by directly inputting all query conditions into the search engine, which will result in the missing of many functionally relevant repositories that do not satisfy some non-functional requirements. Considering that the functional relevance of a repository is generally cared more about by users, we obtain $R(q)$ concentrating on the functional query conditions, i.e., $C_i.QI \in$ {FN, TP, ADES, RDES, FTA, FTAR}.

Next, we measure the relevance between each candidate repository $r \in R(q)$ and q by integrating the relevance between r and each query condition C_i, denoted as $rel(r, C_i)$. Specifically, $rel(r, C_i)$ is calculated based on the type of $C_i.value\_expression$ as follows.

- **Single Value (a).** For this type of value[15], the closer the corresponding attribute value of r is to a, the higher $rel(r, C_i)$ should be. Let A_i denotes the attribute of r that corresponds to the query item $C_i.QI$, and $r.A_i$ is the value of A_i. Then, $rel(r, C_i)$ is calculated using Eq. 2[16] In the equation, $r.mPL$ represents the main PL of r; $r.PLs$ represents the set of PLs used by r; $\mathbb{1}\{\}$ is an indicator function; $secs()$ calculates the number of seconds in a datetime interval.

[15]For the string-typed single value from the value expression of a functional query condition, $rel(r, C_i)$ is measured using Eq. 6 instead of Eq. 2.

[16]For the branches of Eqs. 2∼5, the computation of $rel(r, C_i)$ is performed sequentially from the top branch to the bottom branch based on the if-conditions.

$$\begin{cases} 2^{\mathbb{1}\{a==r.mPL\}-1} & \text{if } a \in r.PLs \\ 1 & \text{if } a \text{ equals/belongs to } r.A_i \\ 0.99 * (1 - \frac{|r.A_i - a|}{max_{r_j \in R(q)} |r_j.A_i - a|}) & \text{if } a \text{ is an integer or a float} \\ 0.99 * (1 - \frac{secs(|r.A_i - a|)}{max_{r_j \in R(q)} secs(|r_j.A_i - a|)}) & \text{if } a \text{ is a datetime} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$\begin{cases} 1 & \text{if } r.A_i \in range \\ 0.99 * (1 - \frac{min(|r.A_i - a|, |r.A_i - b|)}{max_{r_j \in R(q)} min(|r_j.A_i - a|, |r_j.A_i - b|)}) & \text{if } a \text{ and } b \text{ are integers/floats} \\ 0.99 * (1 - \frac{min(secs(|r.A_i - a|), secs(|r.A_i - b|))}{max_{r_j \in R(q)} min(secs(|r_j.A_i - a|), secs(|r_j.A_i - b|))}) & \text{if } a \text{ and } b \text{ are datetimes} \end{cases} \tag{3}$$

$$\begin{cases} 1 & \text{if } r.A_i \in S \\ 0.99 * (1 - \frac{min_{e \in S} |r.A_i - e|}{max_{r_j \in R(q)} min_{e \in S} |r_j.A_i - e|}) & \text{if the elements in } S \text{ are integers/floats} \\ 0.99 * (1 - \frac{min_{e \in S} secs(|r.A_i - e|)}{max_{r_j \in R(q)} min_{e \in S} secs(|r_j.A_i - e|)}) & \text{if the elements in } S \text{ are datetimes} \\ \frac{\sum_{e \in r.PLs \cap S} 2^{\mathbb{1}\{e==r.mPL\}-1}}{max_{r_j \in R(q)} \sum_{e \in r_j.PLs \cap S} 2^{\mathbb{1}\{e==r_j.mPL\}-1}} & \text{if the elements in } S \text{ are PLs} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

- **Range Value ($range$=$[a, b]$, (a, b), $(a, b]$, or $[a, b)$).** For this type of value, $rel(r, C_i)$ is calculated using Eq. 3 based on whether $r.A_i$ falls within the range or its proximity to the lower and upper bounds of the range. Note that if a is not given (e.g., (,20]), it is set to 0, 0.0, or the launch datetime of GitHub, respectively; and if b is not given (e.g., [20,)), it is set to the MAX_INT, 1.0, or the datetime when q is submitted, respectively.

- **Set Value (S=$\{a, b, ...\}$).** For this type of value, $rel(r, C_i)$ is calculated using Eq. 4 based on whether $r.A_i$ contains any element in the set or its best proximity to any element in the set.

- **Logical Expression (LE=> a, $\geq a$, $< a$, or $\leq a$).** For this type of value, $rel(r, C_i)$ is calculated using Eq. 5 based on whether $r.A_i$ satisfies the expression or or its proximity to the lower/upper bound of LE.

$$\begin{cases} 1 & \text{if } r.A_i \text{ satisfies } LE \\ 0.99 * (1 - \frac{|r.A_i - a|}{max_{r_j \in R(q)} |r_j.A_i - a|}) & \text{if } a \text{ is an integer or a float} \\ 0.99 * (1 - \frac{secs(|r.A_i - a|)}{max_{r_j \in R(q)} secs(|r_j.A_i - a|)}) & \text{if } a \text{ is a datetime} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

- **String Value (s).** This kind of value appears in the functional query conditions. If $C_i.QI \in \{\text{FN, TP, ADES, RDES, FTA, FTAR}\}$, $rel(r, C_i)$ is calculated by measuring the functional relevance between s and $r.A_i$. Specifically, s is first preprocessed using the steps described previously. We then compute $rel(r, C_i)$ using a weighted N-gram based method, i.e.,

$$\frac{\sum_{k=1}^{len(T(s))} k\times \mid G_k(T(s)) \cap G_k(T(r.A_i)) \mid}{max_{r_j \in R(q)} \sum_{k=1}^{len(T(s))} k\times \mid G_k(T(s)) \cap G_k(T(r_j.A_i)) \mid} \tag{6}$$

  where $T(s)$ denotes the preprocessed token sequence of s; $len(T(s))$ is the length (i.e., the number of tokens) of $T(s)$; and $G_k(T(s))$ represents the set of k-grams obtained from $T(s)$.

After calculating the relevance scores between all query conditions of q and the corresponding attribute values of r, the overall relevance between r and q, denoted as $rel(r, q)$, is calculated by incorporating the user's preferences:

$$rel(r, q) = \sum_{C_i \in q} rel(r, C_i) \times C_i.preference\_weight \tag{7}$$

Notice that if there are a relatively large number of candidate repositories obtained using the ES search engine, it could be time-consuming to measure the relevance between every candidate repository and the query using Eqs.2∼7. We can reduce the search space by retrieving an initial set of top-n (e.g., 100 by default according to our experiment results) repositories. Although the default keyword-based matching mechanism employed by the ES search engine cannot accurately measure the functional relevance between a repository and the query, the reduced set of candidate repositories can filter most of the functionally dissimilar repositories and greatly improve time efficiency. After that, we measure the relevance between each of the top-n candidate repositories and the query using our proposed method. By sorting

the candidate repositories in descending order according to their relevance to the query, a repository recommendation list is produced.

Through observation, there can be repositories in the recommendation list that have the same relevance but differ in the values of user-queried attributes. The rankings of such repositories can be refined according to the magnitude or popularity of the attribute values. Generally speaking, the functional relevance measured on the four attributes corresponding to the query items FN, TP, ADES, and RDES should be as high as possible. For the other attributes, if their values do not satisfy the query conditions: 1) the integer or float values (e.g., the values of 'stargazers_count' and 'issue_closure_rate') should be larger the better; 2) the enumeration values (e.g., the values of 'main_language' and 'license') with higher popularity[17] is preferred; 3) for the boolean values (e.g., the values of 'has_downloads' and 'has_wiki'), `true` is better than `false`. Based on these heuristics, we re-rank the candidate repositories with the same relevance. If such repositories have multiple attributes with different values, their rankings are adjusted sequentially following the priority of the user's preferences on the query conditions. Specifically, the repositories are first re-ranked according to the values of the attribute, A, with the highest preference; after that, if multiple repositories still have the same value of A, they are further re-ranked according to the next attribute by preference.

Finally, for each recommended repository, the values of its attributes corresponding to the query items along with the relevance scores are displayed to help the user effectively determine the suitable repository.

4. Evaluation

In this section, we evaluate our proposed *UERR* by conducting a series of experiments to answer the following four research questions (RQs).

- **RQ1**. What is the proper setting of the parameter n?
- **RQ2**. How effective is *UERR* compared with existing search engines?

[17]The popularity of a PL, license, or visibility, e.g., e, can be measured as $\frac{|R(e)|}{|R|}$, where $R(e)$ denotes the set of repositories labeled with e, licensed by e, or set as e.

Table 3: Statistics of the numbers of stars and commits of 29,155 experimental repositories

Attribute	Min	Max	Mean	Std
#Stars	0	64,652	22.79	687.78
#Commits	1	564,658	92.09	3,435.95

- **RQ3**. Do the LLM-based preprocessing steps and the weighted N-gram based functional relevance measurement method contribute to better performance of *UERR*?
- **RQ4**. How efficient is *UERR*?

Our experimental environment is a laptop with Intel(R) Core(TM) i7-1360P 2.20 GHz and 32GB RAM, running Windows 11 OS.

4.1. Experiment Setup

Dataset. We collect 29,155 repositories primarily related to five industrial topics, i.e., computer numerical control (CNC), computer-aided design (CAD), computer-aided manufacturing (CAM), programmable logic controller (PLC), and supply chain optimization, from GitHub. Specifically, for each topic, we obtain all relevant repositories retrieved by requesting the GitHub repository search API[18] with the topic keywords. For instance, the URL used to retrieve the repositories about "*computer numerical control*" is: "*https://api.github.com/search/repositories?q=computer numerical control in:name,description,topics*". The limitation of the GitHub API that returns at most 100 repositories per request is addressed using a strategy of pagination and multi-batch retrieval based on creation datetime partitioning, which are implemented by setting the parameters `per_page` and `created`.

By performing a preliminary analysis of the repositories, they involve 179 languages, e.g., Jupyter Notebook and Python. The numbers of stars and commits of the repositories both have a wide distribution, ranging from 0 to 64,652 (with a mean of 22.79) and from 1 to 564,658 (with a mean of 92.09), respectively, as listed in Table 3. These results indicate that the repositories have good diversity and can effectively cover GitHub repositories with different characteristics.

[18] https://docs.github.com/en/rest/search/search?apiVersion=2026-03-10#search-repositories

Table 4: Experimental Queries

ID	Query	UERR Query
Q1	plc simulator	FTA:plc simulator
Q2	cnc gcode	FTA:cnc gcode
Q3	geometry kernel occt	FTA:geometry kernel occt
Q4	programmable logic controller simulator	FTA:programmable logic controller simulator
Q5	computer numerical control gcode	FTA:computer numerical control gcode
Q6	3d-printing marlin using c, and functionality is more important	FTA:3d-printing marlin:0.9 & LAN:c
Q7	supply chain simulation in python/c++, and functionality is more important	FTA:supply chain simulation:0.9 & LAN:{python,c++}:0.7
Q8	inventory optimization with c++, and functionality is more important	FTA:inventory optimization:0.9 & LAN:c++:0.7
Q9	digital twin for supply chain, c#, and functionality is more important	FTA:digital twin for supply chain:0.9 & LAN:c#:0.7
Q10	gcode generator cam python, and functionality is more important	FTA:gcode generator cam:0.9 & LAN:python:0.7
Q11	plc simulator using python, stars≥100, with priority "functionality > language > stars"	FTA:plc simulator:0.9 & LAN:python:0.7 & StaC:≥100
Q12	cnc gcode with c#/c++, stars≥100, with priority "functionality > language > stars"	FTA:cnc gcode:0.9 & LAN:{c#,c++}:0.7 & StaC:≥100:0.6
Q13	geometry kernel occt, python, stars≥100, with priority "functionality > language > stars"	FTA:geometry kernel occt:0.9 & LAN:python:0.7 & StaC:≥100:0.6
Q14	3d-printing marlin, c, stars≥100, with priority "functionality > language > stars"	FTA:3d-printing marlin:0.9 & LAN:c:0.7 & StaC:≥100:0.6
Q15	inventory optimization with c++, stars≥100, with priority "functionality > language > stars"	FTA:inventory optimization:0.9 & LAN:c++:0.7 & StaC:≥100:0.6
Q16	plc simulator using python, stars≥100, with over 50% issues are solved, with priority "functionality > language > stars > others"	FTA:plc simulator:0.9 & LAN:python:0.7 & StaC:≥100:0.6 & ICR:(0.5,)
Q17	cnc gcode with c#, stars≥100, with 50+% issues and pull requests are completed, with priority "functionality > language > stars > others"	FTA:cnc gcode:0.9 & LAN:c#:0.7 & StaC:≥100:0.6 & ICR:>0.5 & PRCR:>0.5
Q18	geometry kernel occt, python, stars≥100, with over 50% issues are solved, with priority "functionality > language > stars > others"	FTA:geometry kernel occt:0.9 & LAN:python:0.7 & StaC:≥100:0.6 & ICR:>0.5
Q19	3d-printing marlin, c/c++/c#, stars≥100, with over 50% issues are solved, with priority "functionality > language > stars > others"	FTA:3d-printing marlin:0.9 & LAN:{c,c++,c#}:0.7 & StaC:≥100:0.6 & ICR:>0.5
Q20	inventory optimization with c++, stars≥100, with over 50% issues are solved, with priority "functionality > language > stars > others"	FTA:inventory optimization:0.9 & LAN:c++:0.7 & StaC:≥100:0.6 & ICR:>0.5

We finally standardize each repository by extracting or computing the values of the 30 attributes defined in our URS (see Table 1). The content of the attributes 'full_name', 'topics', 'about_description', and 'readme_description' are preprocessed using the steps described in Section 3.3.2. Then, we create an ES index for the preprocessed repositories.

Queries. We create 20 experimental queries as listed in Table 4. The column "*UERR* Query" presents the queries formulated according to our query grammar (see Section 3.3.1), and the column "GitHub Query" presents the queries expressed using the repository search syntax of GitHub APIs.

The queries exhibit various characteristics. Some queries, e.g., Q1∼Q3, have acronyms, such as 'plc', 'cnc', and 'occt' (short for "Open CASCADE Technology[19]"), while some queries, e.g., Q4 and Q5, use the full expanded forms of the acronyms. Moreover, the queries vary in complexity, with 0∼4 non-functional requirements, e.g., "*using c*", "*stars*$\geq$*100*", and "*over 50% issues are solved*". Furthermore, some queries, e.g., Q6∼Q20, are associated with user-defined priority, e.g., "*functionality is more important*" and "*functionality* > *language* > *stars*". Notice that the query grammar of *UERR* has more powerful expression capability than the search syntax of GitHub APIs. For example, some non-functional requirements, e.g., those on the issue closure ratio (ICR) and pull request completion ratio (PRCR) of Q17∼Q20, cannot be expressed using the GitHub search syntax. The user's preference priority is also not supported by GitHub.

Baselines. We select two popular search engines and three representative retrieval methods as baselines, which are briefly introduced as follows.

- **GitHub Search Engine (GHSE)**. The capability of GitHub search engine is implemented by the GitHub search APIs. We retrieve the repositories for a query by calling the repository search APIs using the corresponding "GitHub Query" shown in Table 4.

- **Elasticsearch search engine (ESSE)**. This is the default search engine implemented by the ES library, which has been widely adopted in both industry (e.g., AWS and Google Cloud) and academia [30, 31]. For a query, we transform the "*UERR* Query" according to the query

[19]Open CASCADE Technology (OCCT) is a software development platform providing services for 3D surface and solid modeling, CAD data exchange, and visualization. More details can be found on the official website https://dev.opencascade.org/

grammar of ES and retrieve the repositories by calling the ES search API based on the ES index built for the preprocessed repositories.

- **BM25-based Functional Matching (BM25)**. BM25 is a popular functional matching algorithm used for text/document retrieval. We implement the algorithm using the `bm25s` module[20] and apply it to measure the functional relevance between a repository and a query. Specifically, we first create four BM25 indexes for the full_name (FN), topics (TP), about_description (ADES), and functional text (FTA) of the preprocessed repositories and then use them to retrieve similar repositories for the corresponding conditions of a query. The resulting functional relevance between a repository and the query is used to replace our weighted N-gram based method.

- **IDF-weighted Keyword Matching (IDF+KWM)**. This is a simple classic method for measuring the functional relevance between a candidate repository, r, and a query, q, denoted as $rel(r, q)$. We first calculate an asymmetric relevance of r to q using Eq. 8. Similarly, another asymmetric relevance of q to r can be calculated by exchanging the order of r and q in Eq. 8. Finally, $rel(r, q)$ is calculated as the *harmonic mean* of both asymmetric relevance scores.

$$rel(r \to q) = \frac{\sum_{t \in T(r) \cap T(q)} idf(t)}{\sum_{t \in T(r)} idf(t)} \tag{8}$$

  where $T(r)$ and $T(q)$ represent the two sets of tokens contained in the textural descriptions of r and q, respectively; and $idf(t)$ is the IDF value of the word t. The IDF values of all words are calculated using the `TfidfVectorizer` module of Scikit-learn[21] from the preprocessed descriptions of all repositories.

- **IDF-weighted Word Embedding Matching (IDF+WEM)**. This is a popular method widely used to measure semantic functional relevance between the textual descriptions of a candidate item and a query by leveraging the distributed vector representations of words learned using word embedding techniques, e.g., Word2vec [32]. To employ this

[20] https://github.com/xhluca/bm25s
[21] https://scikit-learn.org/stable/

method, we obtain the embedding vectors of words by applying the `Word2vec` module of Gensim[22] to the set of descriptive sentences from all preprocessed repositories. Then, for a candidate repository, r, and a query, q, the asymmetric semantic relevance of r to q is calculated as

$$\frac{\sum_{t_i \in T(r)} max_{t_j \in T(q)} csim(V(t_i), V(t_j)) \times idf(t_i)}{\sum_{t_i \in T(r)} idf(t_i)} \tag{9}$$

where $V(t)$ denotes the embedding vector of the word t; and $csim()$ calculates the Cosine similarity between two embedding vectors. After calculating another asymmetric relevance of q to r by exchanging their order in Eq. 9, the final semantic relevance between r and q is calculated as the *harmonic mean* of both asymmetric relevance scores.

The two search engines, GHSE and ESSE, are used to answer RQ2. The three functional relevance measurement methods, BM25, IDF+KWM, and IDF+WEM, are used to answer RQ3 by implementing three variants of *UERR* (see Section 4.4).

Metrics. We use three widely used metrics: Pre@k (Precision at k), MRR@k (Mean Reciprocal Rank at k), and NDCG@k (Normalized Discounted Cumulative Gain at k), to evaluate the top-k repositories produced by *UERR* and the baselines.

$$Prec@k = \frac{\#\text{ relevant repositories in top-}k}{k}$$

$$MRR@k = \begin{cases} \frac{1}{\text{rank}_1} & \text{if rank}_1 \leq k \\ 0 & \text{otherwise} \end{cases}$$

$$NDCG@k = \frac{1}{IDCG_k} \sum_{i=1}^{k} \frac{2^{rel_i - 1}}{log_2(1+i)}$$

where $rank_1$ is the ranking position of the first relevant repository in the recommendation list; rel_i is the relevance of the repository at the ranking position i; and $IDCG_k$ represents the maximum possible DCG score through position k that can be achieved for a query.

[22] https://github.com/piskvorky/gensim

4.2. RQ1: What is the proper setting of the parameter n?

Motivation. As described in Section 3.3, there is a key parameter in *UERR*, i.e., the top-n candidate repositories retrieved using the ES search engine. The setting of n has an impact on the performance of *UERR*. A small n (e.g., 50) may miss some repositories relevant to the query, while a relatively large n (e.g., 500) may introduce many irrelevant repositories. Therefore, it is necessary to determine the proper setting of n to ensure good performance of *UERR*.

Experiment. We perform *UERR* for each query listed in Table 4 by varying n from 50 to 1,000. Then, we collect the entire set of the top-20 repositories recommended for each query under the different n values. Two master students from the affiliation of the first co-author evaluate the relevance of each repository to the query by five grades: 0-'strongly irrelevant', 1-'irrelevant', 2-'neutral', 3-'relevant', and 4-'strongly relevant'. The students, respectively, have three and five years of experience exploring repositories on GitHub. In total, there are 844 repositories to be evaluated. Both students first perform the evaluation independently; and they have disagreements on the relevance of 129 repositories. We measure the inter-rater agreement between the two students using Cohen's Kappa [33]. The value is 0.79, indicating substantial agreement. After discussing the disagreements together, both students reach a common decision on the evaluation results. Then, for each setting of n, we measure the Pre@k, MRR@k, and NDCG@k values for each recommendation list and compute the average of every specific metric (e.g., Pre@1) for all queries.

Result. **To ensure an overall good performance of *UERR*, it is recommended to set n=100.** Table 5 presents the average Pre@k, MRR@k, and NDCG@k of *UERR* under the different settings of n. As n increases from 50 to 1,000, the performance of *UERR* first rises and then declines in terms of every specific metric in most cases (except Pre@3). Most peak values are reached when n=100. We analyze the relevance of recommended repositories and find that increasing n from 50 to 100, more relevant repositories are included in spite of some noise, which leads to performance improvement. When n increases to more than 100, a considerable number of irrelevant repositories are introduced with only a few relevant ones, resulting in a decrease in performance.

Table 5: Average performance of *UERR* under different n values

n	Pre@1	Pre@3	Pre@5	Pre@10	Pre@15	Pre@20
50	1.00	0.97	0.92	0.89	0.84	0.81
100	1.00	0.95	0.95	0.90	0.86	0.84
200	0.95	0.92	0.86	0.84	0.79	0.78
300	0.90	0.87	0.81	0.80	0.75	0.72
500	0.85	0.83	0.81	0.82	0.78	0.73
1000	0.85	0.83	0.80	0.80	0.77	0.72
n	MRR@1	MRR@3	MRR@5	MRR@10	MRR@15	MRR@20
50	1.00	1.00	1.00	1.00	1.00	1.00
100	1.00	1.00	1.00	1.00	1.00	1.00
200	0.95	0.97	0.97	0.97	0.97	0.97
300	0.90	0.92	0.92	0.92	0.92	0.92
500	0.85	0.88	0.90	0.90	0.90	0.90
1000	0.85	0.89	0.90	0.90	0.90	0.90
n	NDCG@1	NDCG@3	NDCG@5	NDCG@10	NDCG@15	NDCG@20
50	0.91	0.90	0.90	0.89	0.88	0.87
100	0.94	0.90	0.91	0.90	0.88	0.88
200	0.91	0.87	0.85	0.86	0.85	0.84
300	0.89	0.86	0.83	0.84	0.83	0.82
500	0.86	0.83	0.82	0.84	0.82	0.81
1000	0.86	0.83	0.81	0.81	0.80	0.78

4.3. RQ2: How effective is UERR compared with existing search engines?

Motivation. *UERR* aims to address the limitations of the GitHub search engine (GHSE). We need to validate whether *UERR* can retrieve more relevant repositories for queries than GHSE. Moreover, ES provides an efficient search engine for a large-scale corpus, which has been widely used in industry and academia. We also want to validate the effectiveness of *UERR* by comparing it with the ES search engine (ESSE).

Experiment. We implement GHSE by calling the GitHub repository search API with the experimental queries expressed according to the search syntax of GitHub, and also implement ESSE by applying the ES search API to each query based on the ES index built for the preprocessed repositories, as described in Section 4.1. The relevance of the top-20 repositories retrieved using GHSE and ESSE are evaluated by the two master students involved in the experiment of RQ1. In total, both baselines recommend 413 new repositories that need to be evaluated. By independently performing the evaluation, both students have 53 disagreements; and the Cohen's Kappa value is 0.82, which indicates almost perfect agreement. The higher agreement can be attributed to the students' prior evaluation experience. After discussing the disagreements, both students also reach a consensus. Then,

for each baseline, we measure the Pre@k, MRR@k, and NDCG@k of each recommendation list and compute the average metric values for all queries.

We further calculate the performance improvement ratio of *UERR* compared to each baseline B as $\frac{M(UERR,m)-M(B,m))}{M(B,m)}$ for every specific metric m. $M(UERR, m)$ and $M(B, m)$ respectively denote the average value of m achieved by *UERR* and B. Moreover, we examine whether the performance improvement of *UERR* over B is statistically significant using the Wilcoxon signed-rank test [34].

Result. ***UERR* achieves the best performance and significantly improves ESSE and GHSE by 27.12%∼165.87% in terms of all metrics.** Table 6 presents the comparison of Pre@k, MRR@k, and NDCG@k between *UERR* and the two baselines. The rows '*UERR* vs ESSE' and '*UERR* vs GHSE' in the table present the performance improvement ratio with significance of *UERR* over the two baselines.

UERR achieves the optimal performance and improves both baselines by a relatively large margin. Specifically, in terms of Pre@k, as k increases from 1 to 20, the average performance of *UERR* decreases from 1.00 to 0.84, while the average performances of ESSE and GHSE decrease from 0.55 to 0.39 and from 0.75 to 0.32, respectively. The improvement ratios of *UERR* compared to ESSE and GHSE are 81.82%∼114.74% and 33.33%∼165.87%, respectively. In terms of MRR@k, the average metric values of *UERR*, ESSE, and GHSE are almost stable (except for a slight increase observed on the MRR@3 of ESSE), at 1.00, 0.57, and 0.75, respectively. *UERR* improves ESSE and GHSE by 73.91% ∼ 81.82% and 33.33%, respectively. The NDCG@k values of *UERR*, ESSE, and GHSE decrease from 0.94 to 0.88, from 0.61 to 0.47, and from 0.74 to 0.38, respectively. The improvement ratios of *UERR* compared to ESSE and GHSE are 53.06%∼87.7% and 27.12%∼130.86%, respectively. Almost all the improvements of *UERR* over the baselines (except "*UERR* vs GHSE" on NDCG@1) are statistically significant.

From the perspective of Pre@k and NDCG@k, we observe that GHSE performs better than ESSE when k is less than 5, and thereafter GHSE becomes worse than ESSE. This contrasting result is due to the fact that although most repositories retrieved by GHSE are relevant to the queries, however GHSE can return only a few (e.g., less than 5) repositories for 15 (75%) queries. In contrast, ESSE recommends more than 20 repositories for 13 (65%) queries. Through our analysis, ESSE and GHSE share a common issue: they cannot distinguish the importance priority between functional and

Table 6: Performance comparison between *UERR* and two baselines

Method	Pre@1	Pre@3	Pre@5	Pre@10	Pre@15	Pre@20
ESSE	0.55	0.55	0.50	0.44	0.41	0.39
GHSE	0.75	0.58	0.49	0.38	0.34	0.32
UERR	1.00	0.95	0.95	0.90	0.86	0.84
UERR vs ESSE	81.82%**	72.73%**	90.00%**	104.55%***	109.76%***	114.74%***
UERR vs GHSE	33.33%*	62.86%**	93.88%***	136.84%***	155.45%***	165.87%***
Method	MRR@1	MRR@3	MRR@5	MRR@10	MRR@15	MRR@20
ESSE	0.55	0.57	0.57	0.57	0.57	0.57
GHSE	0.75	0.75	0.75	0.75	0.75	0.75
UERR	1.00	1.00	1.00	1.00	1.00	1.00
UERR vs ESSE	81.82%**	73.91%**	73.91%**	73.91%**	73.91%**	73.91%**
UERR vs GHSE	33.33%*	33.33%*	33.33%*	33.33%*	33.33%*	33.33%*
Method	NDCG@1	NDCG@3	NDCG@5	NDCG@10	NDCG@15	NDCG@20
ESSE	0.61	0.56	0.54	0.50	0.48	0.47
GHSE	0.74	0.61	0.54	0.45	0.40	0.38
UERR	0.94	0.90	0.91	0.90	0.88	0.88
UERR vs ESSE	53.06%**	61.25%**	69.82%***	79.76%***	82.58%***	87.70%***
UERR vs GHSE	27.12%	47.57%**	69.81%**	100.12%***	117.75%***	130.86%***

*: p<0.05, **: p<0.01, ***:p<0.001

non-functional requirements. They try to find repositories that could *strictly satisfy* (adopted by GHSE) or *partially satisfy* (adopted by ESSE) all the requirements specified in a query. This is the reason why they cannot recommend enough candidate repositories for some queries. *UERR* overcomes this issue by obtaining an initial set of top-n candidate repositories according to the functional requirements only and then measuring the comprehensive relevance between every candidate repository and the query. Therefore, *UERR* can always return a sufficient number of candidate repositories for a query.

We also find that GHSE cannot match acronyms with their full expanded forms (e.g., 'CAD' vs "computer-aided design"). As a result, only one repository is returned for Q4 and Q5, respectively, while 130 and 624 repositories are returned for Q1 and Q2, respectively. This limitation is addressed by *UERR* using the LLM-based acronym expansion step.

4.4. RQ3: Do the LLM-based preprocessing steps and the weighted N-gram based functional relevance measurement method contribute to better performance of UERR?

Motivation. Despite the URS and the query grammar designed for *UERR*, we propose another two special mechanisms: 1) *LLM-AE+CWS*: the

LLM-based acronym expansion and conjoined word splitting step used to preprocess the textual descriptions of repositories and queries; and 2) *Weighted-Ngram*: the weighted N-gram based method for measuring the functional relevance between a repository and a query. It is necessary to verify whether these mechanisms can help improve the performance of *UERR*.

Experiment. We implement four variants of *UERR*, namely 1) **UERR (BM25)**: This variant replaces *Weighted-Ngram* with the BM25 method described in Section 4.1; 2) **UERR(KWM)**: This variant replaces *Weighted-Ngram* with the IDF+KWM method described in Section 4.1; 3) **UERR (WEM)**: This variant replaces *Weighted-Ngram* with the IDF+WEM described in Section 4.1; and 4) **UERR(-LLM)**: This variant removes *LLM-AE+CWS* from the preprocessing pipeline for repositories and queries.

We apply the four variants of *UERR* to each query. The two master students participated in the experiments of RQ1 and RQ2 evaluate the top 20 repositories recommended by the variants for each query. There are 914 newly recommended repositories. Both students first evaluate the relevance of the repositories independently, which results in 80 disagreements. The Cohen's Kappa value is 0.88, indicating almost perfect agreement. The students eventually reach a consensus by discussing the disagreements. Then, for each variant, we measure the Pre@k, MRR@k, and NDCG@k for each recommendation list and compute the average metric values for all queries. Similar to the experiment of RQ2, we calculate the performance improvement ratios of *UERR* compared to each variant and test the statistical significance of the improvement achieved by *UERR* over the variants.

Result. **Both *LLM-AE+CWS* and *Weighted-Ngram* contribute to better performance of *UERR*. And, *Weighted-Ngram* contributes more than *LLM-AE+CWS*.** Table 7 presents the comparison of Pre@k, MRR@k, and NDCG@k between *UERR* and the four variants. We can see that after removing *LLM-AE+CWS*, the performance of *UERR* decreases by 1.79%∼11.3%, 5.26%∼11.11%, and 4.06%∼11.94% with respect to the Pre@k, MRR@k, and NDCG@k metrics, respectively. Compared with the BM25 method, *Weighted-Ngram* improves *UERR* by 14.16%∼42.86%, 22.2% ∼42.86%, and 9.05%∼22.95% in terms of Pre@k, MRR@k, and NDCG@k, respectively. Compared with the classic IDF+KWM method, *Weighted-Ngram* improves *UERR* by 11.76%∼25%, 11.11%∼25%, and 16.89%∼29.31% in terms of Pre@k, MRR@k, and NDCG@k, respectively. In contrast with the popular IDF+WEM method, *Weighted-Ngram* improves the Pre@k, MRR@k, and NDCG@k of *UERR* by 5.56%∼ 17.65%, 8.11%∼17.65%, and 13.55%∼

Table 7: Performance comparison between *UERR* and its four variants

Method	Pre@1	Pre@3	Pre@5	Pre@10	Pre@15	Pre@20
UERR	1.00	0.95	0.95	0.90	0.86	0.84
UERR(BM25)	0.70	0.80	0.80	0.77	0.75	0.72
UERR(KWM)	0.80	0.85	0.83	0.81	0.76	0.73
UERR(WEM)	0.85	0.90	0.86	0.81	0.76	0.76
UERR(-LLM)	0.90	0.93	0.90	0.87	0.80	0.75
UERR vs UERR(BM25)	42.86%*	18.75%*	18.75%*	16.88%	14.16%**	16.32%**
UERR vs UERR(KWM)	25.00%*	11.76%*	14.46%*	11.80%**	12.66%**	13.95%**
UERR vs UERR(WEM)	17.65%	5.56%	10.47%*	11.80%*	12.66%*	10.93%*
UERR vs UERR(-LLM)	11.11%	1.79%	5.56%	3.45%	7.95%	11.30%*
Method	MRR@1	MRR@3	MRR@5	MRR@10	MRR@15	MRR@20
UERR	1.00	1.00	1.00	1.00	1.00	1.00
UERR(BM25)	0.70	0.81	0.82	0.82	0.82	0.82
UERR(KWM)	0.80	0.90	0.90	0.90	0.90	0.90
UERR(WEM)	0.85	0.93	0.93	0.93	0.93	0.93
UERR(-LLM)	0.90	0.95	0.95	0.95	0.95	0.95
UERR vs UERR(BM25)	42.86%*	23.71%*	22.20%*	22.20%*	22.20%*	22.20%*
UERR vs UERR(KWM)	25.00%*	11.11%*	11.11%*	11.11%*	11.11%*	11.11%*
UERR vs UERR(WEM)	17.65%	8.11%	8.11%	8.11%	8.11%	8.11%
UERR vs UERR(-LLM)	11.11%	5.26%	5.26%	5.26%	5.26%	5.26%
Method	NDCG@1	NDCG@3	NDCG@5	NDCG@10	NDCG@15	NDCG@20
UERR	0.94	0.90	0.91	0.90	0.88	0.88
UERR(BM25)	0.76	0.81	0.82	0.81	0.81	0.80
UERR(KWM)	0.72	0.75	0.76	0.76	0.75	0.75
UERR(WEM)	0.79	0.79	0.78	0.77	0.75	0.76
UERR(-LLM)	0.84	0.87	0.85	0.85	0.84	0.83
UERR vs UERR(BM25)	22.95%*	11.68%*	11.85%*	10.41%*	9.05%*	9.83%**
UERR vs UERR(KWM)	29.31%**	20.12%***	20.58%***	18.43%***	16.89%***	17.02%***
UERR vs UERR(WEM)	19.05%*	13.55%***	17.02%***	16.36%***	16.70%***	15.21%***
UERR vs UERR(-LLM)	11.94%	4.06%	6.71%*	5.36%*	5.33%*	5.25%

*: p<0.05, **: p<0.01, ***:p<0.001

19.05%, respectively. Moreover, in terms of NDCG@k (which considers the fine-grained relevance of recommended repositories), the performance improvements of *UERR* contributed by both mechanisms are statistically significant in most cases.

Through an in-depth analysis of the repositories recommended for the queries, we find that without preprocessing the repositories and queries using *LLM-AE+CWS*, UERR(-LLM) cannot effectively match acronyms with their full expanded forms, leading to the missing of repositories relevant to some queries. For example, many repositories containing 'plc' in their 'full_name', 'topics', and/or 'about_description' are not retrieved by UERR(-LLM) for Q4. As for UERR(BM25) and UERR(KWM), they both measure the functional relevance between a repository and a query based on the Bags-Of-

Words (BOW) model, which does not consider the order of words and thus cannot accurately calculate the functional relevance, resulting in the retrieval of irrelevant repositories. This issue is addressed by *Weighted-Ngram* as the n-grams (with n >1) can capture the sequential semantics of words and thus can measure the functional relevance accurately. The performance of UERR(WEM) is slightly better than that of UERR(KWM). This is because the word embedding techniques (e.g., Word2vec) can help measure semantic relevance between words. As a result, UERR(WEM) can retrieve semantically relevant repositories for queries. However, some irrelevant repositories may also be returned by UERR(WEM) due to the inaccurate semantic relevance measured based on the embedding vectors of words.

4.5. RQ4: How efficient is UERR?

Motivation. We have validated in RQ2 and RQ3 that *UERR* can retrieve more relevant repositories for queries than state-of-the-art baselines. In this RQ, we want to examine the time efficiency of *UERR* to see whether it could be accepted by users for practical uses.

Experiment. We apply *UERR*, ESSE, GHSE, and the four variants of *UERR* to each experimental query three times. We record the amount of time required by each method to produce the repository recommendation lists. Then, for each method, M, we compute the average time of M on each query and the overall average time of M on all queries. We also calculate the standard deviation for the average times of M on all queries.

Result. **The overall average time required by *UERR* to recommend repositories for a query is 3.77 seconds, which is close to that of GHSE. And, the efficiency of *UERR* is more stable than GHSE.** Table 8 presents the average times of seven methods on each query as well as the overall average time (i.e., the 'Avg' column) and the standard deviation of the average times (i.e., the 'Std' column) of the methods. In terms of the overall average time, the order of the methods is UERR(WEM) (8.60s) > ESSE (5.09s) > UERR(BM25) (4.64s) > UERR (3.77s) > UERR(KWM) (3.63s) > GHSE (3.20s) > UERR(-LLM) (0.21s). According to the standard deviation values, the four methods UERR(-LLM), UERR(KWM), *UERR*, and ESSE are more stable than UERR(BM25), UERR(WEM), and GHSE. Based on these results, although *UERR* performs a little slower than GHSE, its efficiency is more stable than GHSE. Specifically, *UERR* takes 3.16s∼4.91s to produce the repository recommendation list for a query, while GHSE may

Table 8: Time efficiency (in seconds) comparison of seven methods

Query	ESSE	GHSE	UERR(BM25)	UERR(KWM)	UERR(WEM)	UERR(-LLM)	UERR
Q1	4.69	6.42	5.90	3.13	7.25	0.24	3.19
Q2	6.07	29.65	6.76	4.36	8.10	0.07	4.42
Q3	4.98	1.47	5.80	3.57	13.05	0.21	4.24
Q4	6.57	1.40	3.64	3.64	7.72	0.44	3.69
Q5	5.97	1.63	6.95	4.88	6.26	0.66	4.91
Q6	4.68	2.94	5.45	3.37	4.74	0.10	3.41
Q7	4.46	2.93	5.64	3.53	4.46	0.17	3.54
Q8	4.06	2.46	5.78	3.45	4.94	0.31	3.46
Q9	4.78	1.16	5.74	3.42	14.14	0.18	3.49
Q10	5.23	1.23	6.17	4.18	18.98	0.18	4.30
Q11	4.76	1.22	3.15	3.15	12.40	0.14	3.20
Q12	5.33	1.54	4.60	4.60	12.58	0.07	4.72
Q13	5.28	1.22	3.15	3.16	20.76	0.31	4.10
Q14	5.02	1.33	3.64	3.65	7.37	0.11	3.78
Q15	5.11	1.23	3.17	3.17	3.71	0.31	3.17
Q16	5.02	1.14	3.70	3.70	5.17	0.12	3.72
Q17	5.33	1.31	3.78	3.78	5.11	0.06	3.81
Q18	4.99	1.16	3.35	3.35	6.66	0.21	3.63
Q19	4.81	1.29	3.12	3.12	4.57	0.07	3.16
Q20	4.71	1.23	3.37	3.37	3.93	0.30	3.38
Avg	5.09	3.20	4.64	3.63	8.60	0.21	3.77
Std	0.58	6.35	1.36	0.50	5.03	0.15	0.52

take 1.14s∼29.65s. Therefore, the response time of *UERR* is more predictive than that of GHSE.

UERR(-LLM) has the optimal overall average time and the smallest standard deviation. By combining the results from Tables 8 and 6, **UERR(-LLM) outperforms GHSE on all the effective and efficient metrics. UERR(-LLM) provides an alternative for *UERR* if efficiency is important.** Moreover, from the overall average times of *UERR* and UERR(-LLM), we find that the LLM-based acronym expansion and conjoined word splitting of a query[23] is the most costly step in the online workflow of *UERR*, which occupies 94.43%(=(3.77-0.21)/3.77) of the total time.

[23]This time can be notably reduced by constructing a dictionary offline for mapping acronyms and conjoined words to their full expanded forms or word sequences. After that, when parsing a query, only the acronyms and conjoined words outside the dictionary need to be processed by calling the LLM API.

5. Discussion

Threats to internal validity relate to the errors in the implementation of methods and the subjective bias of participants in manual tasks. As for our *UERR* and its variants, we implement them according to the details described in the paper. As for the baselines, namely ESSE, GHSE, BM25, IDF+KWM, and IDF+WEM, we implement them based on the APIs and usage specification provided by GitHub, ES, and mature Python modules/toolkits (e.g., `bm25s` and `Gensim`). To ensure the correctness of the implementation of these methods, we double-check the code and inspect the output of intermediate steps during the testing with example queries. As for the URS definition and the relevance evaluation of the repositories recommended by the methods, three developers and two master students (who are not co-authors of the paper) are recruited to collaboratively perform the two tasks, respectively, which can help eliminate the bias of a single person.

Threats to external validity relate to the generalizability of experiment results. To conduct the experiments, we collect 290+ hundred repositories from the world's largest open-source platform, GitHub. Moreover, we create 20 queries differing in several aspects such as the inclusion of acronyms, the number of non-functional requirements, and the user's preference priority. The relatively large-scale dataset of real-world repositories and the diversity of queries can improve the generalizability of our experiment results.

6. Conclusion

This paper proposes a unified effective model named *UERR* for retrieving open-source repositories. We first define a unified repository schema (URS) of 30 attributes for characterizing the information of repositories interested by users when retrieving repositories. Next, we design a query grammar to guide users to express requirements using the attributes along with personalized preferences. Finally, we propose an integrated model for retrieving repositories relevant to a query by measuring the relevance between a repository and the query based on the different value types of the query conditions. Particularly, to accurately measure functional relevance, we propose an LLM-based method to expand acronyms and split conjoined words (referred to as *LLM-AE+CWS*) in the textual descriptions of repositories and queries and also propose a weighted N-gram based functional relevance measurement method, referred to as *Weighted-Ngram*. The results of experiments with 20

diverse queries show that *UERR* significantly outperforms the search engines provided by GitHub and Elasticsearch. The proposed *LLM-AE+CWS* and *Weighted-Ngram* contribute to better performance of *UERR*, in comparison with three representative functional relevance measurement methods.

In future work, we plan to extend this study by supporting the retrieval of open-source repositories for other platforms, e.g., Gitee[24]. Moreover, we will try to enhance the capability of *UERR* by mining valuable information from the source code files included in repositories.

Acknowledgment

This work is supported by the Open Project of Key Laboratory of Industrial Software Engineering and Application Technology, Ministry of Industry and Information Technology (HK202303542), the Joint Open Fund of the Research Platforms of School of Computer Science, China University of Geosciences, Wuhan (PTLH2024-A-01), the National Natural Science Foundation of China (62066040), and the Cultivation of Comprehensive Talents in Korean Studies and Establishment of a High-level Research Platform in Central China (AKS-2025-INC-2230002).

References

[1] S. A. Ajila, D. Wu, Empirical study of the effects of open source adoption on software development economics, Journal of Systems and Software 80 (9) (2007) 1517–1529.

[2] K. Alrashedy, A. Binjahlan, How do software engineering researchers use github? an empirical study of artifacts & impact, in: 2024 IEEE International Conference on Source Code Analysis and Manipulation (SCAM), IEEE, 2024, pp. 118–130.

[3] B. Su, K. Zheng, W. Wang, A github project recommendation model based on self-attention sequence, in: Proceedings of the 2021 3rd International Conference on Big Data Engineering, 2021, pp. 110–116.

[4] H. Lin, G. Liang, Y. Wu, B. Wu, C. Tian, W. Wang, Open source software supply chain recommendation based on heterogeneous information

[24] https://gitee.com/

network, in: International Symposium on Benchmarking, Measuring and Optimization, Springer, 2022, pp. 70–86.

[5] H. Yang, S. Sun, J. Wen, H. Cai, M. Mateen, Improving personalized project recommendation on github based on deep matrix factorization, in: Collaborative Computing: Networking, Applications and Worksharing: 17th EAI International Conference, CollaborateCom 2021, Virtual Event, October 16-18, 2021, Proceedings, Part I 17, Springer, 2021, pp. 318–332.

[6] S. Yu, W. Liu, H. Wu, Z. Liao, Github project recommendation based on knowledge graph and developer similarity, The Computer Journal (2025) bxaf026.

[7] R. Beniwal, S. Dahiya, D. Kumar, D. Yadav, D. Pal, Npmrec: Npm packages and similar projects recommendation system, in: Data Analytics and Management: Proceedings of ICDAM, Springer, 2021, pp. 701–710.

[8] W. Xu, X. Sun, J. Hu, B. Li, Repersp: recommending personalized software projects on github, in: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2017, pp. 648–652.

[9] W. Xu, X. Sun, X. Xia, X. Chen, Scalable relevant project recommendation on github, in: Proceedings of the 9th Asia-Pacific Symposium on Internetware, 2017, pp. 1–10.

[10] X. Sun, W. Xu, X. Xia, X. Chen, B. Li, Personalized project recommendation on github, Science China Information Sciences 61 (2018) 1–14.

[11] J.-Y. Jiang, P.-J. Cheng, W. Wang, Open source repository recommendation in social coding, in: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2017, pp. 1173–1176.

[12] P. Zhang, F. Xiong, H. Leung, W. Song, Funkr-pdae: Personalized project recommendation using deep learning, IEEE Transactions on Emerging Topics in Computing 9 (2) (2018) 886–900.

[13] D. Ford, N. Shrestha, T. Zimmermann, Reboc: recommending bespoke open source software projects to contributors, in: 2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), IEEE, 2022, pp. 1–5.

[14] T. F. Bissyandé, F. Thung, D. Lo, L. Jiang, L. Réveillere, Orion: A software project search engine with integrated diverse software artifacts, in: 2013 18th international conference on engineering of complex computer systems, IEEE, 2013, pp. 242–245.

[15] X. Yue, C. Liu, N. Zhang, H. Hu, X. Zhang, Visrepo: A visual retrieval tool for large-scale open-source projects, in: Proceedings of the 15th Asia-Pacific Symposium on Internetware, 2024, pp. 499–502.

[16] J. Wu, Y. Sun, J. Zhang, An open-source repository retrieval service using functional semantics for software developers, in: 2022 International Conference on Service Science (ICSS), IEEE, 2022, pp. 12–20.

[17] C. Y. Suen, N-gram statistics for natural language understanding and text processing, IEEE transactions on pattern analysis and machine intelligence (2) (1979) 164–172.

[18] S. Robertson, Understanding inverse document frequency: on theoretical arguments for idf, Journal of documentation 60 (5) (2004) 503–520.

[19] L. Zhang, Y. Zou, B. Xie, Z. Zhu, Recommending relevant projects via user behaviour: an exploratory study on github, in: Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies, 2014, pp. 25–30.

[20] M. Izadi, M. Nejati, A. Heydarnoori, Semantically-enhanced topic recommendation systems for software projects, Empirical Software Engineering 28 (2) (2023) 50.

[21] R. Samer, A. Felfernig, M. Stettinger, Towards issue recommendation for open source communities, in: IEEE/WIC/ACM International Conference on Web Intelligence, 2019, pp. 164–171.

[22] H. Ren, M. Ma, X. Zhang, Y. Cao, C. Nie, Effective recommendation of cross-project correlated issues based on issue metrics, in: Proceedings of the 14th Asia-Pacific Symposium on Internetware, 2023, pp. 1–1.

[23] S. Bai, L. Liu, H. Liu, M. Zhang, C. Meng, P. Zhang, Find potential partners: A github user recommendation method based on event data, Information and Software Technology 150 (2022) 106961.

[24] X. Zhang, H. Xu, Q. Yu, S. Zeng, S. Dai, H. Yang, S. Wu, License recommendation for open source projects in the power industry, Information and Software Technology 167 (2024) 107391.

[25] J. Vargovich, F. Santos, J. Penney, M. A. Gerosa, I. Steinmacher, Givemelabeledissues: An open source issue recommendation system, in: 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), IEEE, 2023, pp. 402–406.

[26] J. Xuan, H. Jiang, H. Zhang, Z. Ren, Developer recommendation on bug commenting: A ranking approach for the developer crowd, Science China Information Sciences 60 (2017) 1–18.

[27] H. A. Çetin, E. Doğan, E. Tüzün, A review of code reviewer recommendation studies: Challenges and future directions, Science of Computer Programming 208 (2021) 102652.

[28] Q. Qi, J. Cao, D. Wang, Gat-team: A team recommendation model for open-source software projects., in: SEKE, 2023, pp. 286–291.

[29] V. Kovalenko, N. Tintarev, E. Pasynkov, C. Bird, A. Bacchelli, Does reviewer recommendation help developers?, IEEE Transactions on Software Engineering 46 (7) (2018) 710–731.

[30] Z. Chen, N. Zhang, P. Si, Q. Chen, C. Liu, Z. Zheng, Shellfusion: An answer generator for shell programming tasks via knowledge fusion, in: 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), IEEE, 2023, pp. 93–97.

[31] C. Liu, X. Xia, D. Lo, Z. Liu, A. E. Hassan, S. Li, Codematcher: Searching code based on sequential semantics of important query words, ACM Transactions on Software Engineering and Methodology (TOSEM) 31 (1) (2021) 1–37.

[32] K. W. Church, Word2vec, Natural Language Engineering 23 (1) (2017) 155–162.

[33] L. M. Hsu, R. Field, Interrater agreement measures: Comments on kappan, cohen's kappa, scott's π, and aickin's α, Understanding Statistics 2 (3) (2003) 205–219.

[34] R. F. Woolson, Wilcoxon signed-rank test, Wiley encyclopedia of clinical trials (2007) 1–3.