

Article

A Software Repository Tag Method Based on Hybrid Search and Graph Enhancement

Min Wang ¹, Shanshan Wu ¹, Wanjia Lv ² and Yu Zhao ^{2,3,4,*}

¹ Nanjing Research Institute of Electronics Engineering, Nanjing 210007, China; minwangnuaa@gmail.com (M.W.); shanshanwunuaa@gmail.com (S.W.)

² School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China; lvwanj@nuaa.edu.cn

³ Shenzhen Research Institute, Nanjing University of Aeronautics and Astronautics, Shenzhen 518110, China

⁴ Key Laboratory of Safety-Critical Software Development and Verification, Ministry of Industry and Information Technology, Nanjing 211106, China

* Correspondence: zhao_yu@nuaa.edu.cn

Abstract

Software repositories play an essential role in modern software engineering by enabling code sharing, collaboration, and reuse. Repository tag recommendation is a key technique for improving the discoverability and organization of software repositories. However, existing methods still suffer from noisy annotations, incomplete tag coverage, long-tailed label distributions, and limited adaptability to emerging technical terms. To address these issues, this paper proposes ReG-TG, a retrieval-augmented framework for repository tag recommendation. ReG-TG integrates hybrid retrieval, a tag co-occurrence knowledge graph, and Chain-of-Thought (CoT) reasoning within a large language model (LLM)-based architecture. Specifically, the hybrid retrieval module combines dense semantic embeddings and sparse lexical matching to retrieve relevant reference repositories, followed by a reranking mechanism to refine candidate results. A tag co-occurrence graph is further constructed to model structural relationships among tags and provide graph-enhanced tag knowledge. The retrieved context and graph-enhanced information are then incorporated into structured prompts to guide LLM-based tag generation. Experimental results on multiple GitHub repository datasets show that ReG-TG consistently outperforms representative baselines, including TF-IDF+LR, ZestXML, and LEGION, in terms of Precision@5, Recall@5, and F1-score@5. Further ablation studies demonstrate that retrieval augmentation, hybrid retrieval, reranking, graph enhancement, and structured reasoning each contribute to performance improvement. In addition, long-tail analysis shows that ReG-TG maintains better robustness on low-frequency labels, indicating its effectiveness in alleviating the sparsity and long-tail challenges in repository tag recommendation.

Keywords: ReG-TG; hybrid retrieval; chain-of-thought; LLM; repository tag recommendation



Academic Editor: Hernán Astudillo

Received: 10 June 2026

Revised: 22 July 2026

Accepted: 3 August 2026

Published: 4 August 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Software repositories serve as centralized platforms for hosting source code, documentation, and related resources, enabling developers to efficiently manage, share, and collaborate on software projects [1,2]. With the rapid evolution of software engineering and the widespread adoption of artificial intelligence technologies, the number of projects hosted on open-source platforms has grown explosively in recent years. This

unprecedented scale brings significant challenges to repository management, particularly in enabling developers to effectively search, discover, and reuse relevant projects.

To facilitate repository organization and retrieval, modern platforms have introduced tag or topic recommendation mechanisms [3]. For example, GitHub employs the repo-topics framework, which analyzes README files using Term Frequency-Inverse Document Frequency (TF-IDF) to suggest descriptive tags [4]. These tags play a critical role in categorizing repositories and improving their discoverability. However, in practice, the final assignment of tags is often left to repository owners, which may contain spelling errors, excessive abbreviations, or highly idiosyncratic expressions that are difficult for other users to interpret. Beyond annotation quality, existing repository tags often suffer from limited expressiveness; many tags are overly coarse-grained or capture only a single aspect of a project, failing to provide a comprehensive description. Moreover, many repositories lack tags entirely. Such issues significantly hinder effective project retrieval and recommendation, as relevant repositories may not be returned even when users query with appropriate keywords.

To address these challenges, a substantial body of research has explored automated repository tag recommendation [5,6]. Early approaches primarily relied on traditional machine learning methods, such as Naïve Bayes and Support Vector Machines (SVMs), combined with shallow textual features like TF-IDF [5,7]. While these methods demonstrated the feasibility of automatic classification, they are limited in capturing deep semantic information and result in low accuracy. With the advent of deep learning techniques, neural models based on Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Pre-trained Language Models (PLMs) have been introduced, significantly improving semantic understanding [7,8]. However, these approaches typically formulate tag recommendation as a closed-set multi-label classification problem, which restricts their ability to handle the open and evolving nature of repository tags [9].

Another fundamental challenge lies in the extreme distribution of repository tags, which typically follows a long-tail pattern. Recent studies have shown that directly fine-tuning PLMs on such data leads to severe popularity bias, where models tend to recommend high-frequency “head” tags while performing poorly on the vast number of rare but important “tail” tags. In addition, most existing methods treat tags as independent labels, overlooking the rich semantic and structural relationships among them [10,11]. In practice, tags exhibit strong correlations and hierarchical dependencies: for example, “react” is closely related to “javascript,” and “deep learning” is a subfield of “machine learning”.

Recently, Large Language Models (LLMs) have strong contextual understanding and generation capabilities and can adapt to new tasks through contextual examples [12]. In repository tag recommendation, traditional discriminative models are constrained by the closed-world assumption, because their prediction space is limited to the fixed label set observed during training. However, the open-source community evolves rapidly, and new technical terms continue to emerge. In this paper, zero-shot or emerging tags refer to tags that are not included, or are not sufficiently represented, in the supervised label space of classification-based baselines, rather than labels that are absent from all available repository data. This setting makes it difficult for traditional models to cover newly appearing technical concepts [9,13]. LLMs can understand repository functions and technical features from complex and unstructured README documents and generate candidate tags in a more open-vocabulary manner. These capabilities make LLMs suitable as reasoning and tag-generation components for repository tagging. On the other hand, directly using an LLM for open-vocabulary generation may result in hallucinations, such as nonexistent tags or lengthy descriptive content rather than standardized short-phrase tags. To address this problem, retrieval-augmented generation can provide external evidence to

constrain the generation process. The external knowledge base provides relevant repository examples and tag knowledge, while the LLM focuses on logical reasoning and formatted output [14]. Through in-context learning, the model can generate candidate tags based on the retrieved reference information without being retrained whenever new repository topics appear [12,14].

Based on the above analysis, this study formulates three hypotheses to guide the design and evaluation of the proposed framework. The first hypothesis concerns the overall effectiveness of the framework. Existing repository tag recommendation methods usually rely on predefined label spaces and training data. However, repository tags in open-source communities are user-defined, sparse, and continuously evolving, which makes it difficult for traditional classification-based methods to cover newly emerging or low-frequency tags. By combining external repository retrieval with LLM-based semantic reasoning, the proposed framework is expected to improve recommendation performance across datasets with different label-space characteristics [6,9,13,14].

H1: *ReG-TG achieves better repository tag recommendation performance than representative baseline methods on different GitHub topic datasets.*

The second hypothesis concerns the contribution of the main components. Repository tag recommendation is affected by several different factors, including lexical mismatch between repositories, noisy or incomplete user-defined tags, insufficient modeling of tag relationships, and the uncertainty of LLM-generated outputs. These problems correspond to different stages of the recommendation process. Therefore, hybrid search, re-ranking, graph enhancement, and structured reasoning are expected to provide complementary effects rather than replace one another [6,15].

H2: *The complete ReG-TG framework achieves better recommendation performance than its variants with key components removed.*

The third hypothesis concerns the performance under long-tailed tag distributions. Long-tail tags are more difficult to recommend because they have fewer training examples and weaker statistical signals. Compared with head tags, tail tags require additional semantic and contextual evidence. If retrieved reference repositories and tag relationship information can provide such evidence, ReG-TG is expected to show more stable performance under long-tailed distributions [6,14,16].

H3: *ReG-TG achieves more stable recommendation performance under long-tailed tag distributions than representative baseline methods.*

To test these hypotheses, we propose ReG-TG, a repository tag recommendation framework that integrates hybrid retrieval, re-ranking, tag co-occurrence graph enhancement, and CoT-based structured reasoning. GitHub README documents contain both natural-language descriptions of repository functions and technical terms, such as library, framework, and API names. Dense retrieval can capture deep semantic information but may overlook exact matches of technical terms, whereas sparse retrieval can precisely match specific keywords but has limited ability to recognize semantic relationships between different expressions. Therefore, ReG-TG combines dense and sparse retrieval in a hybrid retrieval strategy. Since the initial retrieval results may still contain repositories with similar keywords but different functions, a cross-encoder re-ranking mechanism is further employed to perform fine-grained semantic matching on the candidate set. In addition, user-defined GitHub tags are often incomplete, and relying only on the tags of the retrieved

repositories may result in insufficient tag coverage. Therefore, ReG-TG constructs a tag co-occurrence graph from historical tag associations to identify potentially related tags that may be missing from the retrieved samples. Finally, the retrieved repository context and graph-enhanced tag knowledge are organized into a structured prompt. CoT prompting is used to guide the LLM through a structured reasoning process and to distinguish the reasoning content from the final tags. The framework consists of two main stages: offline index construction and online retrieval and recommendation.

In summary, the main contributions of this paper are as follows:

- We first propose the ReG-TG framework based on LLM for software repository tagging, which innovatively introduces hybrid search, knowledge graph enhancement and CoT reasoning mechanisms to enhance the contextual understanding of the repository.
- We conduct systematic evaluations on two recent datasets and one self-collected dataset to assess the effectiveness of the ReG-TG framework and each module.
- We conduct experiments on long-tailed distribution scenarios to verify the practicality of the ReG-TG framework.

Paper Organization. For the remainder of this paper, the following structure is used: Section 2 discusses related works. Section 3 provides details of the proposed method ReG-TG. Section 4 presents key components involved in the experiment. Section 5 evaluates the effectiveness of the proposed method ReG-TG and performance comparison with the baseline methods. Section 6 discusses the impact of the number of retrieved reference repositories. Finally, Section 7 concludes the paper.

2. Related Work

This section introduces related work about Software repository tags.

A large number of studies have employed various classification algorithms and techniques to address the software repository tagging task. Early work [17,18] first demonstrated that textual information from software projects (e.g., descriptions, README files, and package metadata) can be used as input to traditional machine learning models such as Support Vector Machines (SVMs), Naïve Bayes, and topic models (Latent Dirichlet Allocation, LDA) to obtain baseline domain classification performance. However, these approaches are limited by textual noise, vocabulary diversity, and ambiguous category boundaries.

Subsequent studies [13,19,20] focused on constructing large-scale labeled or semi-automatically labeled datasets and investigating the reproducibility of “domain definitions,” i.e., how application domains or functionalities can be consistently defined as discriminative classes. These works emphasize the importance of annotation protocols. Zanartu et al. [13] proposed a hybrid human–machine labeling approach to define application domain categories and establish classification baselines, highlighting that classification performance is highly dependent on training data coverage and label consistency.

With the emergence of deep learning, researchers began to apply neural network-based methods to process textual and code features. Studies by Nguyen, Leclair, and others [8,21], as well as related work, adopted deep neural architectures or adapted text classification models for software categorization, achieving improved representation generalization compared to traditional methods. These approaches are particularly effective in scenarios involving long-form documentation (e.g., README files) and mixed code–text inputs. However, they still face two major practical challenges: (1) poor generalization across languages and ecosystems, and (2) performance degradation on rare classes due to data imbalance, which leads to bias toward high-frequency labels.

To address multi-source information fusion and structural relationship modeling, a line of research incorporates graph-based or heterogeneous information to represent both

internal repository structures (e.g., files, dependencies, and function calls) and external signals (e.g., authorship, forks, and stars). For example, Repo2vec [22] and Rep2Vec [23] propose learning unified repository embeddings that jointly encode code, metadata, and dependency relationships for downstream tasks such as classification and similarity retrieval.

In addition, for weakly supervised or minimal-label settings, researchers have explored hierarchical and weak-label-driven strategies. HiGitClass [24] introduces a keyword-driven hierarchical classification framework that combines human-defined label hierarchies and a small set of keywords with automated learning, thereby reducing annotation cost while leveraging hierarchical information to improve multi-class decision-making. Similarly, Zhang et al. [25,26] investigate minimal-supervision classification methods incorporating metadata such as programming language, dependencies, and file distributions. Their results show that metadata and structured prompts can significantly improve robustness under limited supervision. However, these methods are sensitive to the selection of keywords and metadata, and they generally fail to match fully supervised deep learning models in fine-grained classification performance.

To further clarify the position of ReG-TG in the existing literature, we summarize representative software repository tagging and categorization studies in Table 1. The table focuses on the key technical route of each type of method, especially whether it uses retrieval evidence, graph-based tag knowledge, LLM-based generation, and mechanisms for sparse or long-tailed labels.

Table 1. Comparison of representative studies on software repository tagging and categorization. TC: traditional classification; XML: extreme multi-label learning; NC: neural and pre-trained-language-model-based classification; GSM: graph or structure-aware methods.

Type	Representative Studies	Key Mechanism	Long-Tail/Open-Label Handling	Main Gap Compared with ReG-TG
TC	LDA, SVM, LR, MNB-based methods [5,7,17,18]	Uses textual features such as TF-IDF, topics, descriptions, and README metadata.	Limited; usually depends on a predefined label space.	Weak semantic modeling and limited adaptability to emerging tags.
XML	ZestXML and related XML methods [6,9]	Treats repository tagging as large-scale sparse multi-label prediction.	Designed for large and sparse label spaces.	Does not explicitly use retrieved repository evidence or LLM-based generation.
NC	Neural text classification and LEGION [8,16,21]	Learns semantic representations from repository text; LEGION uses distribution-balanced loss.	Improves long-tail learning through representation learning or loss design.	Still mainly constrained by supervised training labels.
GSM	Greta, Repo2Vec, Rep2Vec, HiGitClass [10,22–24]	Models tag relations, repository similarity, metadata, hierarchy, or structural signals.	Can use structural or hierarchical information.	Usually not designed for open-vocabulary tag generation with retrieved evidence.
ReG-TG	Our work	Combines hybrid retrieval, reranking, PMI-based tag graph expansion, and structured LLM generation.	Uses retrieved repositories and graph-expanded tags to support sparse and long-tailed labels.	Higher inference cost than pure classification methods.

As shown in Table 1, existing studies have made important progress, but several gaps remain. Traditional classification and neural classification methods usually rely on predefined label spaces, which limits their adaptability to emerging or rarely observed tags. Extreme multi-label learning methods improve scalability under sparse labels, but they do not explicitly use retrieved repository examples as contextual evidence. Graph-based and structure-aware methods capture useful relationships among tags or repositories,

but they are usually not combined with LLM-based open-vocabulary generation. In contrast, ReG-TG treats repository tag recommendation as a retrieval-enhanced generation task. It retrieves semantically and lexically similar repositories, expands candidate tag evidence through a PMI-based tag co-occurrence graph, and guides an LLM to generate normalized tags with a structured prompt. Therefore, ReG-TG differs from existing studies by jointly integrating retrieval evidence, graph-enhanced tag knowledge, and structured LLM reasoning.

3. A Software Repository Tag Recommendation Method Based on Hybrid Search and Graph Enhancement

3.1. Problem Formulation and Unified Framework Description

Let $\mathcal{D} = \{r_i\}_{i=1}^N$ denote a repository corpus, where each repository r_i is associated with a textual description x_i constructed from its repository name, short description, and README content. Let $\mathcal{V} = \{t_1, t_2, \dots, t_{|\mathcal{V}|}\}$ denote the global tag vocabulary, and let $Y_i \subseteq \mathcal{V}$ be the ground-truth tag set of repository r_i . Given a target repository r_q with textual input x_q , the goal of repository tag recommendation is to generate a ranked tag list $\hat{Y}_q^K = \{\hat{t}_1, \hat{t}_2, \dots, \hat{t}_K\}$, where each predicted tag $\hat{t}_j \in \mathcal{V}$ or is normalized into the GitHub topic format, and the predicted list should maximize its relevance to the true tag set Y_q .

Unlike conventional discriminative multi-label classification methods that directly learn a mapping $f : x_q \rightarrow Y_q$ over a fixed label space, ReG-TG formulates repository tag recommendation as a retrieval-enhanced generation process. First, the target repository text x_q is used to retrieve an initial candidate set $\mathcal{C}_q$ from the indexed repository corpus through hybrid dense–sparse retrieval:

$$S_{\text{final}}(x_q, x_i) = \alpha S_{\text{dense}}(x_q, x_i) + (1 - \alpha) S_{\text{sparse}}(x_q, x_i),$$

where S_{dense} measures semantic similarity, S_{sparse} measures lexical matching, and α controls their relative contribution. The top-ranked candidates are then refined by a reranker to obtain a smaller reference set $\mathcal{R}_q$.

Second, ReG-TG constructs a tag co-occurrence graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ from the training repository set, where nodes are tags and edge weights are computed using Pointwise Mutual Information (PMI). Given the tags contained in the retrieved reference repositories, the graph enhancement module extracts related neighbor tags and forms a graph-enhanced tag evidence set $\mathcal{N}_q$. Finally, the target repository text x_q , retrieved reference repositories $\mathcal{R}_q$, and graph-enhanced tag evidence $\mathcal{N}_q$ are organized into a structured prompt. The LLM-based generator G_θ produces the final ranked tag list:

$$\hat{Y}_q^K = G_\theta(x_q, \mathcal{R}_q, \mathcal{N}_q).$$

Therefore, the complete ReG-TG framework can be viewed as a unified process consisting of candidate retrieval, semantic reranking, graph-based tag evidence expansion, and constrained LLM-based tag generation.

Based on the above formulation, as shown in Figure 1, we propose ReG-TG, a topic recommendation framework that integrates graph structure, retrieval enhancement generation, and a large language model. The core process of this framework consists of two main stages: offline index construction and online retrieval and recommendation. In the offline index construction stage, based on the metadata from massive historical repositories, ReG-TG first utilizes hybrid text encoding to build a vector index library covering both dense semantic representations and sparse lexical representations. And then automatically constructs a lightweight tag co-occurrence knowledge graph based on statistical association features between tags. These two components are the foundation for the system's

underlying structured prior knowledge. In the online retrieval and recommendation stage, when faced with a new target repository, ReG-TG executes a hybrid retrieval strategy, synergistically leveraging the deep semantic capture capabilities of dense vectors and the precise technical terminology matching capabilities of sparse vectors to efficiently recall semantically similar reference fragments from a large-scale knowledge base. Subsequently, ReG-TG uses a re-ranking mechanism to perform secondary refinement of the candidate reference set to eliminate semantic noise in the initial retrieval. Based on this, ReG-TG further collaborates with the tag co-occurrence knowledge graph to mine potential semantic associations in the candidate tag set through a graph nearest neighbor algorithm. Finally, ReG-TG organizes the retrieved repository context with the graph-enhanced association knowledge into a structured prompt, submits it to LLM, and performs structured reasoning to generate the final label. To make the processing procedure of ReG-TG explicit and to complement the framework-level view in Figure 1, we summarize the complete workflow as follows:

1. Repository preprocessing: raw README documents are cleaned, segmented, and normalized to obtain repository text fragments.
2. Offline index construction: repository fragments are encoded by BGE-M3 and stored in the vector database, while GitHub topic tags are used to construct the PMI-based tag co-occurrence graph.
3. Initial hybrid retrieval: given a target repository, ReG-TG combines dense semantic retrieval and sparse lexical retrieval to recall the Top-50 candidate repository fragments.
4. Candidate reranking: BGE-Reranker-v2-m3 reranks the Top-50 candidates and selects the Top-K most relevant reference repositories.
5. Graph-based tag expansion: the tags of the Top-K reference repositories are used as anchor nodes in the tag co-occurrence graph to obtain graph-enhanced neighbor tags.
6. Structured prompt construction: the target repository text, retrieved reference repositories, and graph-enhanced tags are organized into Retrieval Context, KG Context, and Target Input fields.
7. LLM-based tag generation and parsing: the LLM generates a reasoning process and outputs normalized tags after the fixed marker, and the parser extracts the final JSON-formatted tag list.

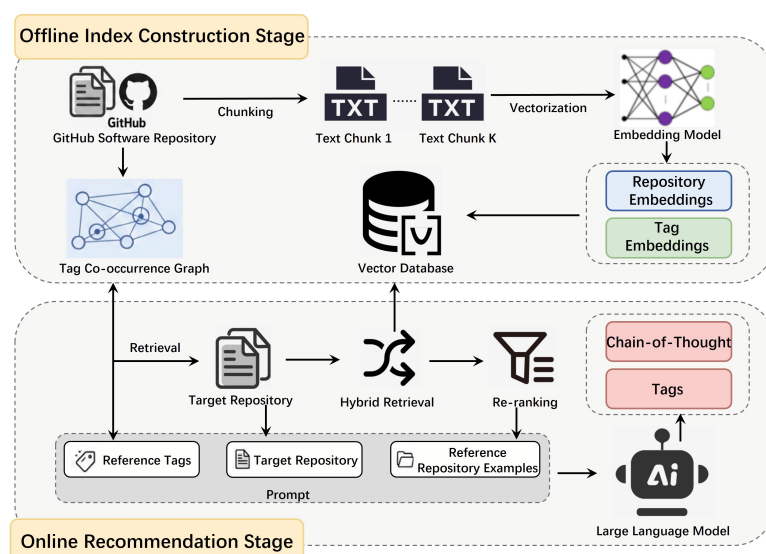


Figure 1. The framework of the proposed ReG-TG.

3.2. Offline Index Construction

Building an offline index is the foundation of the entire ReG-TG framework. Its main task is to transform raw, unstructured GitHub repository data into vector representations that computers can efficiently retrieve and compare. Specifically, this process includes two phases, i.e., vector database construction and label co-occurrence graph construction. The former is used to encode repositories into vector representations and build an efficient similarity search index for large-scale retrieval. The latter is used to capture global relationships among tags by modeling their co-occurrence patterns, thereby providing structured semantic guidance for tag recommendation.

(1) Vector database construction

First, ReG-TG utilizes hybrid text encoding to construct a vector database, which includes three steps: text sequence construction, vector representation, and index storage.

Text sequence construction We employ an overlapping block strategy to construct the input text sequence for semantic encoding. Specifically, by setting a fixed sliding window size and overlap step, long texts are segmented into several sub-segments with overlapping context. This approach helps preserve the semantic information of the document and maintain semantic coherence at the segmentation boundaries through overlapping regions, thereby reducing contextual breaks caused by hard segmentation.

Vectorization representation After constructing the text sequence, we vectorize the fragment text sequence using embedding techniques. Specifically, to simultaneously capture the deep semantics and literal matching features of the text, we employ the BGE-M3 (BAAI General Embedding M3) model for multi-granularity text encoding [27]. This model possesses multilingual, multi-functional, and multi-granularity characteristics, capable of mapping the input text fragments into both dense and sparse vector representations. The former captures the high-dimensional semantic information of the text, addressing semantic mismatch issues. The latter learns the lexical weight of each token in the text, focusing on the precise matching of proper nouns and technical terms, compensating for the shortcomings of dense vectors in precise keyword matching. Through this hybrid encoding strategy, ReG-TG can balance the breadth of semantic understanding with the accuracy of keyword matching, providing rich information for subsequent hybrid retrieval.

Index storage To support efficient vector matching and retrieval, we use the Milvus Lite library to build a hybrid text vector index. During the index construction, a metadata mapping database was established, associating the unique ID of each vector with the original name of the repository, the list of real tags, and the block index. This ensures that when similar vectors are retrieved subsequently, the corresponding real tags can be immediately obtained as a reference for generation. Furthermore, given that the text fragments generated after the repository documents are segmented reach hundreds of thousands, we adopted the classic IVF_FLAT (Inverted File Flat) algorithm from the approximate nearest neighbor search field to optimize query response speed while maintaining retrieval accuracy, thereby reducing computational overhead and supporting efficient query responses.

(2) Label co-occurrence graph construction

Second, ReG-TG utilizes metadata from massive historical repositories to construct a global tag co-occurrence knowledge graph. The specific process is illustrated in Algorithm 1. This graph is essentially a weighted tag co-occurrence matrix, where nodes represent unique tags, and edge weights are calculated using Pointwise Mutual Information (PMI) to measure the association strength between two tags. Unlike simple co-occurrence frequency statistics, PMI-based weighting reduces the influence of high-frequency generic tags and helps identify more informative tag associations. For example, statistical patterns show that NumPy and Data Analysis co-occur frequently, and PyTorch is often

associated with Deep Learning. In this way, the graph makes the implicit dependencies scattered across massive repositories explicit, solidifying them into a queryable domain knowledge structure.

Algorithm 1: Label co-occurrence knowledge graph construction

Input: $\mathcal{D}_{train}$: repository set, each repository r includes tags set T_r ; δ : minimum co-occurrence frequency threshold; τ : PMI filtering threshold; K_{graph} : maximum number of neighbors to retain for each node

Output: $\mathcal{G} = (V, E, W)$: weighted directed label co-occurrence graph

```

1  $V \leftarrow \emptyset$ ;  $E \leftarrow \emptyset$ ;  $W \leftarrow \emptyset$ 
2 Initialize frequency maps:
3  $Count(t) \leftarrow 0$ ;  $CoCount(t_i, t_j) \leftarrow 0$ 

4 Step 1: compute label frequency and co-occurrence frequency
5 foreach repository  $r \in \mathcal{D}_{train}$  do
6   foreach tag  $t_i \in T_r$  do
7      $Count(t_i) \leftarrow Count(t_i) + 1$ 
8      $V \leftarrow V \cup \{t_i\}$ 
9     foreach tag  $t_j \in T_r$  do
10      if  $t_j \neq t_i$  then
11         $CoCount(t_i, t_j) \leftarrow CoCount(t_i, t_j) + 1$ 
12      end
13    end
14  end
15 end

16 Step 2: calculate PMI weights and construct edges
17  $N \leftarrow |\mathcal{D}_{train}|$ 
18 foreach pair  $(t_i, t_j)$  in  $CoCount$  do
19    $c_{ij} \leftarrow CoCount(t_i, t_j)$ ;  $weight \leftarrow \log \frac{c_{ij} \cdot N}{Count(t_i) \cdot Count(t_j)}$ 
20   if  $c_{ij} \geq \delta$  and  $weight \geq \tau$  then
21      $E \leftarrow E \cup \{(t_i, t_j)\}$ ;  $W(t_i, t_j) \leftarrow weight$ 
22   end
23 end

24 Step 3: sparse knowledge graph (Top-K prune)
25 foreach node  $t_i \in V$  do
26    $Neighbors \leftarrow$  outgoing edges of  $t_i$ 
27   Sort  $Neighbors$  by weight in descending order
28   Keep only top  $K_{graph}$  edges and remove others
29 end
30 return  $\mathcal{G}$ 

```

3.3. Online Retrieval and Tag Recommendation

Online retrieval and tag recommendation is the core of the entire ReG-TG framework. Its main task is to retrieve the most similar candidate reference fragments for a new target repository and combine them with enhanced knowledge graph semantics to construct structured reasoning of LLM to generate the final tag. Specifically, this process includes four phases, i.e., hybrid search, resort, label co-occurrence knowledge graph enhancement, and structured prompts construction and constraint reasoning. The following will introduce the process step by step:

(1) *Hybrid search*

The first step of online retrieval and tag recommendation is to sift through massive historical data to create a preliminary set of reference fragments. Benefiting from the previously constructed vector database employing a hybrid encoding of dense and sparse vector representations, we also adopt a hybrid retrieval strategy based on the BGE-M3 model. Specifically, dense vector retrieval captures deep semantics, which could understand metaphors and synonyms. And sparse vector retrieval precisely anchors technical terms, which could automatically identify highly discriminative technical terms in the text and assign them higher weights, while reducing the weight of stop words or generic words. For the similarity metric, we choose cosine similarity, which reduces the influence of vector magnitude differences caused by fragments of different lengths in the vector space. Finally, a weighted fusion is used to obtain a score for each reference fragment. The dual scores are merged using a linear weighting method, as shown below, to identify the Top-50 candidate reference fragments with the highest scores:

$$S_{final} = \alpha \cdot S_{dense} + (1 - \alpha) \cdot S_{sparse}$$

Here, the dense score reflects semantic fit, while the sparse score reflects keyword overlap. S_{dense} represents the normalized dense vector similarity, S_{sparse} represents the sparse vector matching score, and α is a balancing hyperparameter used to adjust the weights of semantic understanding and literal matching.

(2) *Resort*

The main purpose of the hybrid retrieval strategy is to exclude most irrelevant samples from the full database and retrieve similar samples with coarse-grained retrieval. In hybrid retrieval, both dense and sparse vectors essentially belong to the Bi-Encoder paradigm. In this architecture, the target and candidate are encoded independently, and they are invisible to each other in the deep network. Although the BGE-M3 model greatly improves feature representation capabilities, the information compression bottleneck caused by independent encoding still exists. This bottleneck is particularly evident when processing GitHub repositories. For example, a tutorial repository named “Learn-React” and a framework repository named “React-Core” share a large number of the same technical keywords, resulting in them being very close in the vector space. However, for the tag recommendation, their natures are completely different; one is a tutorial, and the other is a framework. Therefore, to solve this, we adopt the BGE-Reranker-v2-m3 model as the re-ranking core of the ReG-TG framework to understand the fine-grained interaction relationships between texts [27]. Specifically, instead of processing queries and documents independently, BGE-Reranker-v2-m3 employs a cross-attention interaction mechanism. This involves concatenating the description of the target repository and candidate text fragments into a single input sequence, which is then used as the overall input. Under this architecture, a multi-layered self-attention mechanism allows each token in the target text to interact directly with each token in the candidate text across sequences. This means that ReG-TG can not only see which words appear in two texts but also gain a deeper understanding of the modification relationships, negation logic, and subtle differences in item properties between words. ReG-TG uses BGE-Reranker-v2-m3 to score each of the Top-50 candidate samples, ultimately outputting a scalar score between 0 and 1, representing the confidence level of deep semantic relevance between the target repository and the reference fragment. Based on this score, the candidate list is reordered in descending order, and the Top-K with the highest final scores are selected, which are considered the best reference samples that highly match the target repository in terms of semantics, keywords, and item properties.

(3) *Label co-occurrence knowledge graph enhancement*

The previous search and reordering steps yielded the Top-K reference samples that are closest in content and intent. However, GitHub tag data inherently exhibits strong sparsity and a long-tail distribution. Many high-quality repositories have incomplete tags. For example, a data analysis project might only be labeled with the specific library name “numpy”, omitting its overarching concept “data-analysis” or the associated technology “python”. If LLM relies solely on these incomplete local references, it can easily lead to insufficient tag coverage.

To address this issue, we design a tag co-occurrence graph enhancement module, aim to inject global statistical knowledge into local reference samples. Specifically, ReG-TG extracts the tag set of the reordered Top-K reference repositories and uses it as “anchor points” input into the tag co-occurrence knowledge graph. Then, it automatically searches the knowledge graph for the first-order semantic nearest neighbor with the highest connection weight to the anchor tag. For example, if the retrieved reference repository tag set is “pytorch, transformer,” the graph analysis might recommend the implicit neighbor nodes “natural-language-processing, bert.” Finally, these expanded neighbor labels are deduplicated and merged with the original reference labels, serving as contextual input. This step provides the LLM with additional domain prior knowledge beyond the current README text and suggests potentially related tags based on general co-occurrence patterns in the GitHub community. In this way, the graph enhancement module is expected to improve tag coverage and may help reduce low-recall cases for long-tail labels caused by incomplete reference sample annotations.

(4) *Structured prompts construction and constraint reasoning*

After obtaining high-quality reference samples and extended domain prior knowledge, the next step is to fuse this content with the original description of the target repository using constraint instructions to create a composite prompt. The actual prompt used in our experiments is shown in Table 2. It guides the LLM to first produce a short reasoning process and then output the final tags after a fixed marker, i.e., “### Final Tags:”. To reduce parsing ambiguity, only the content after this marker is required to be a valid JSON object with a single key “tags”. This design separates the reasoning text from the final machine-readable tag list and makes the output easier to parse. The prompt consists of three core information domains.

For reproducibility, we further specify how retrieved repositories and graph-enhanced tags are converted into the LLM input. Each retrieved reference item is formatted as a short text-label pair in the form “Description: {text} | Tags: {labels}”, where {text} denotes the retrieved repository fragment and {labels} denotes its original GitHub topic tags. The Top-K reranked reference items are concatenated in descending reranking score order and placed in the Retrieval Context field. The graph-enhanced tags are obtained by using the tags of these Top-K reference repositories as anchor nodes in the PMI-based tag co-occurrence graph, and the selected neighbor tags are placed in the KG Context field. The target repository name and truncated README content are placed in the Target Input field. Therefore, the LLM receives three explicitly separated evidence sources: retrieved examples, graph-expanded tag candidates, and the target repository text.

Retrieval context domain, which consists of the Top-K reference samples obtained through hybrid search and re-sort in the vector database. This information limits the generated style specifications (such as tag granularity and naming conventions) by showing the LLM similar existing tags, ensuring that the tags conform to community standards.

Knowledge graph neighbor domain, which consists of semantically extended neighbors mined from the tag co-occurrence knowledge graph. This information supplements

the LLM with domain knowledge. For example, when the repository only mentions the specific name, the knowledge graph neighbor domain guides the LLM to focus on its implicit higher-level concepts, thus compensating for the lack of long-tail information caused by simple retrieval.

Target repository domain, which is the summary and description of the repositories to be tagged, is the main content that LLM needs to analyze.

Existing generative methods typically require the LLM to output comma-separated text. This unstructured output is prone to parsing errors and makes it difficult to distinguish between the model’s reasoning process and the final tag list. Therefore, ReG-TG uses a fixed output marker and a JSON constraint for the final tag list. Specifically, the LLM is allowed to provide its reasoning process before the marker “### Final Tags:”, while the content after this marker must be a valid JSON object in the form `{“tags”: [“tag1”, “tag2”, “tag3”]}`. The parser used in our experiments follows a deterministic rule-based procedure. It first locates the marker “### Final Tags:” using regular expression matching and discards all text before the marker. It then parses the remaining content as a JSON object and extracts the value of the “tags” field. If JSON decoding fails, a fallback delimiter-based extraction is applied to split the remaining text into candidate tags. Finally, the extracted tags are normalized by converting them to lowercase, replacing spaces and underscores with hyphens, removing invalid punctuation, deduplicating repeated tags, and keeping the Top-K tags for evaluation.

Furthermore, to prevent logical jumps when the LLM handles complex technology stacks, ReG-TG embeds the CoT mechanism in the reasoning text before the final JSON output. Specifically, the LLM is guided to execute the following three-step reasoning logic: (1) analyze the similarities between the target repository and the retrieval context to determine whether standard labels in the reference sample should be inherited; (2) check the suggestions provided by the knowledge graph neighbor domain to determine whether macro-level classification labels need to be supplemented; and (3) scan the target repository’s unique descriptions to identify new terms that are not present in the reference context but have significant technical representativeness.

Table 2. ReG-TG structured prompt template

System Role	You are an expert software engineering assistant specializing in technical taxonomy and open-source repository classification. Your task is to extract precise technical tags based on the provided evidence.
Retrieval Context	[Retrieval Context: Similar Repositories] (Content retrieved from Milvus and BGE-Reranker) Description: {text} Tags: {labels}
KG Context	[Domain Knowledge Graph: Related Concepts] Based on global statistics, the tags appearing in the context are associated with the following broader concepts. You may consider using them if they fit the target repository: {neighbor_labels}.
Target Input	[Target Repository to Tag] Repository Name: {repo_name} README Snippet: {readme_content_truncated}
Instructions	1. Analyze the technical stack, language, application, functionality, and purpose. 2. Output a brief reasoning process before the final tag list. 3. Output the final tags strictly after “### Final Tags:”. 4. Format: lowercase, hyphenated (e.g., machine-learning, python-client).
Output Format	1. The content after “### Final Tags:” MUST be a valid JSON object with a single key “tags”. 2. The value of “tags” MUST be a list of strings. 3. Format: ### Final Tags: {“tags”: [“tag1”, “tag2”, “tag3”]} Let’s think step by step:

3.4. Computational Cost and Scalability Discussion

The computational cost of ReG-TG mainly comes from five stages: offline index construction, hybrid retrieval, cross-encoder reranking, graph-based tag expansion, and LLM-based generation. In the offline stage, repository README documents are cleaned, segmented, encoded, and stored in Milvus Lite. Following the design described in the retrieval-augmented generation literature, approximate nearest neighbor (ANN) indexing can support sublinear retrieval over a large document collection. In our implementation, the IVF_FLAT index is used to divide the high-dimensional vector space into several cells, so that online retrieval only searches the regions closest to the query vector. This design reduces the retrieval overhead while maintaining query efficiency for large-scale repository fragments.

In the online stage, the initial dense-sparse hybrid retrieval module first filters the full repository index and returns a limited candidate set. The cross-encoder reranker is more computationally expensive because it needs to jointly encode the target repository and each candidate repository fragment. Therefore, following the implementation strategy described above, ReG-TG first uses the efficient BGE-M3 bi-encoder retrieval module to remove most irrelevant samples and retains only the Top-50 candidates for BGE-Reranker-v2-m3 scoring. This two-stage design avoids applying cross-encoder reranking to the entire repository corpus and balances retrieval efficiency with semantic matching accuracy.

The tag co-occurrence graph expansion stage only queries neighbor tags based on the tags extracted from the Top-K retrieved repositories, so its online cost is limited by the number of retrieved repositories and retained graph neighbors. The final LLM-based generation stage is the most expensive online component because its latency is affected by the prompt length and the generated output length. Therefore, ReG-TG improves scalability mainly by moving repository encoding and graph construction to the offline stage, using ANN-based vector retrieval for the online retrieval stage, and restricting the expensive reranking and graph expansion operations to a small candidate set. Nevertheless, because the framework still includes retrieval, reranking, graph query, and LLM generation, its online inference latency is higher than that of pure classification methods. We have clarified this computational trade-off in the revised manuscript.

4. Experimental Setting

In this section, we provide a detailed description of the key components involved in the experiment of our study, including the real dataset from GitHub and the dataset preprocessing steps used for experiment, the benchmark methods and models used for comparison, the evaluation metrics used to quantify model performance, and the specific implementation details of the experiments, such as the experimental environment and key hyperparameters.

4.1. Experimental Datasets

To evaluate ReG-TG, we use two public GitHub topic datasets and one self-collected dataset. These datasets were selected to cover three complementary experimental settings: a large-scale sparse and long-tailed label space, a curated and standardized label space, and a self-collected open-world setting containing emerging topics. The first public dataset (Dataset Zest) was collected by Widyasari et al. and contains 21,273 repositories with 37,161 unique user-defined topics. Infrequent topics were retained, resulting in a highly sparse and long-tailed label space [6].

ZestXML is included as a baseline because it was identified as the strongest zero-shot extreme multi-label learning method among the approaches evaluated on this dataset [6,9].

The second public dataset (Dataset LEG) is the curated dataset used by LEGION. It was derived from the dataset of Widyasari et al. by retaining GitHub featured topics and contains 15,262 repositories with 665 unique labels [16].

In addition, we constructed a self-collected dataset (Dataset ReG) from representative public repositories on GitHub. We collected the raw dataset via the GitHub API and performed additional filtering based on the following criteria: (1) star count limit, only repositories with more than 5 stars were retained to remove low-quality or experimental projects; (2) text integrity, repositories lacking descriptions or with empty README files were removed; (3) tag validity, repositories without any topic tags were removed to ensure the existence of genuine ground truth for evaluation. We also perform simple data cleaning on these GitHub repositories. That is, since README documents are typically written in Markdown or HTML format, the HTML tags, image links, and other non-natural language information interspersed within them not only fail to provide effective semantic features but also interfere with the feature extraction. Therefore, we use regular expressions to filter the original text, removing all formatting characters, URL links, and non-text elements, and eliminating generic phrases such as “License” and “Contributors” that appear repeatedly across different repositories but lack distinctiveness. After these steps, we obtained a new dataset containing 18,773 repositories with 32,338 unique tags, which not only has high label sparsity but also involves a large number of emerging terms.

Due to the large amount of data, it is difficult to visualize the tag distributions of all three datasets. Therefore, we select Dataset ReG for visualization. As shown in Figure 2, the tag distribution of Dataset ReG exhibits typical long-tail characteristics (only showing the top 15 tags with the highest frequency), which suggests the relevance of using retrieval augmentation methods to handle sparse tags.

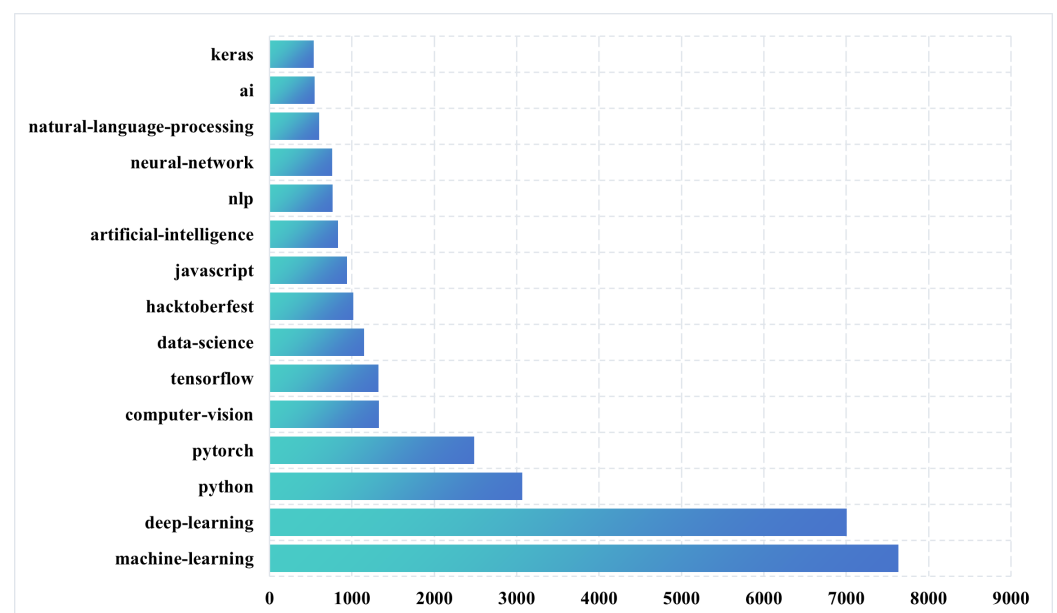


Figure 2. Feature information of the dataset, the vertical axis representing topics and the horizontal axis representing frequencies.

4.2. Comparison Benchmark

To comprehensively evaluate the effectiveness of ReG-TG, we select three representative methods as benchmarks according to their methodological relevance to repository tag recommendation. The selected baselines cover three major technical routes: traditional feature-based classification, extreme multi-label learning, and pre-trained-language-model-based topic recommendation. This selection allows us to compare ReG-TG with both

classical keyword-based methods and recent methods designed for large-scale sparse or long-tailed GitHub topic labels. These benchmark methods are briefly introduced below.

TF-IDF+LR is a traditional statistical method. It uses TF-IDF to extract text features and combines them with logistic regression to build a multi-label classifier. This method represents a traditional benchmark method for keyword matching. **ZestXML** is the current state-of-the-art method for GitHub tag recommendation tasks, a zero-shot extreme multi-label classification method specifically designed to handle large-scale sparse labels. **LEGION** is a recently proposed method based on fine-tuning the pre-trained model Bidirectional Encoder Representations from Transformers (BERT), introducing a distributional balance loss to optimize the prediction of long-tail labels.

For a fair comparison, all baseline methods are evaluated under the same dataset split, input information, and evaluation metrics as ReG-TG. Specifically, each dataset is divided into the same training and test sets. For TF-IDF+LR, ZestXML, and LEGION, the training set is used to train the corresponding multi-label prediction model, and the held-out test set is used for evaluation. For ReG-TG, the same training set is used to build the retrieval index and label co-occurrence graph, while the same test set is used as target repositories for tag recommendation. All methods take repository textual information as input and are evaluated using Precision@5, Recall@5, and F1-score@5.

4.3. Evaluation Metrics

Given that GitHub topic recommendation is a multi-label recommendation task and the model output is a ranked label sequence, we use Precision@K, Recall@K, and F1-score@K as the core evaluation metrics. These metrics are selected because they are commonly used for top-K recommendation evaluation and are consistent with the practical usage scenario of repository tags, where users usually focus on the first several recommended labels. Precision@K measures the accuracy of the recommendation results, i.e., how many of the first K recommended labels are correct. Recall@K measures the coverage of the recommendation results, i.e., how many of the true labels are included in the first K recommended labels. F1-score@K is the harmonic mean of Precision and Recall, used to comprehensively evaluate the overall performance of the method. The formulas for each metric are as follows:

$$Precision@K = \frac{1}{|D|} \sum_{d \in D} \frac{|y_d \cap \hat{y}_d^K|}{K}$$

$$Recall@K = \frac{1}{|D|} \sum_{d \in D} \frac{|y_d \cap \hat{y}_d^K|}{|y_d|}$$

$$F1 - score@K = \frac{2 \times P@K \times R@K}{P@K + R@K}$$

where D is the total number of samples in the test set, y_d is the set of true labels for repository d , and $\hat{y}_d^K$ is the set of the Top K labels predicted by the method. In this experiment, since the average number of tags for most repositories is 5.3, we choose the value $K = 5$ to measure the overall performance of the ReG-TG.

4.4. Experiment Implementation

The implementation settings are chosen according to the functions of different modules in ReG-TG. In the label co-occurrence graph construction phase, we first count the tag frequency and tag-pair co-occurrence frequency based on the full training set, and then calculate Pointwise Mutual Information (PMI) as the edge weight of the tag co-occurrence graph. The PMI threshold τ_{PMI} is set to 0.5 to filter out weak association edges caused by

random co-occurrence, thereby retaining the core knowledge structure with domain universality and technical relevance. In the hybrid retrieval phase, the dense and sparse score fusion weights are set to 0.6 and 0.4, respectively, because dense retrieval is used to capture semantic similarity, while sparse retrieval is used to preserve exact matching of technical terms. This setting is used to balance semantic understanding and technical terminology matching accuracy. In the constraint reasoning phase, we selected DeepSeek-V3 as the final label generation engine because it is suitable for semantic understanding and structured generation when handling complex README texts. Unlike the traditional “train-valid-test” dataset partitioning for classification tasks, we divided the dataset into training and test sets. The training set accounts for 90% of the dataset; this data is not used for model training because this framework does not require model fine-tuning. Instead, it is vectorized and stored in the database as a reference case pool for the retrieval phase. The test set accounts for 10% of the dataset; these data are used as held-out target repositories to evaluate the method’s performance on repositories not included in the retrieval index. All experiments were conducted on a high-performance machine equipped with RTX 4090*4 GPUs. The software environment was based on PyTorch 2.1, Transformers 4.30+, and Python 3.10.

5. Experimental Results and Analysis

In this section, we evaluate the three hypotheses formulated in the Introduction to verify the effectiveness of the proposed method ReG-TG. Below, we elaborate on the approach and results corresponding to H1, H2, and H3 to facilitate experimental verification and results analysis.

H1: *ReG-TG achieves better repository tag recommendation performance than representative baseline methods on different GitHub topic datasets.*

Approach: To evaluate H1 and conduct a comprehensive evaluation, we experiment with the repository tagging task using three datasets with different features described in Section 4.1. We compare the performance of the proposed method ReG-TG with three representative baseline methods, namely TF-IDF+LR, ZestXML, and LEGION, covering traditional feature-based classification, zero-shot extreme multi-label learning, and pre-trained language model fine-tuning.

Results: Table 3 shows the performance comparison of various methods on Precision@5, Recall@5, F1-score@5, and HumanEval. Before computing the automatic metrics, both predicted tags and ground-truth GitHub topics were normalized by lowercasing, converting spaces and underscores to hyphens, removing invalid punctuation, deduplicating repeated tags, and merging common synonymous aliases according to GitHub topic naming conventions. Because GitHub topics are user-defined and may contain missing or noisy labels, we further conducted a human evaluation to complement the automatic metrics. For each dataset–method pair, three expert evaluators independently assessed the semantic relevance of Top-5 recommended tags on 100 randomly selected test repositories. A tag was considered acceptable if at least two evaluators judged it to be semantically relevant to the target repository. The inter-rater agreement measured by Fleiss’ kappa was 0.86, indicating high agreement among the evaluators.

(Observation 1) **The proposed ReG-TG framework consistently outperforms all baseline models in terms of automatic metrics and human semantic evaluation across the three datasets with distinct characteristics.** As shown in Table 3, on the ZestXML and LEGION datasets, which employ relatively fixed label taxonomies, ReG-TG achieves improvements of approximately 7.4% and 5.4% in F1-score, respectively, compared with the strongest baseline models. On the Dataset ReG, which contains a substantial number of emerging technical terms, the performance gain is even more pronounced, reaching 49.1%.

The HumanEval results further show that ReG-TG obtains the highest human acceptance rates on Dataset Zest, Dataset LEG, and Dataset ReG, reaching 0.698, 0.837, and 0.813, respectively. Compared with the best baseline method on each dataset, ReG-TG improves HumanEval by 3.7%, 4.2%, and 4.5%, respectively. The largest HumanEval gain appears on Dataset ReG, which is consistent with the open and long-tailed nature of this dataset and indicates that ReG-TG can generate semantically meaningful tags that may not always be fully covered by the original user-defined GitHub topics. These results demonstrate that the integration of retrieval-augmented generation and large language model (LLM) reasoning can significantly enhance the effectiveness and semantic acceptability of multi-label tag generation.

Table 3. Performance comparison with representative baselines. Best results are in bold; the italicized Diff denotes ReG-TG’s relative gain over the best baseline.

Datasets	Methods	Precision@5	Recall@5	F1-Score@5	HumanEval
Dataset Zest	TF-IDF+LR	0.225	0.183	0.202	0.512
	ZestXML	0.356	0.315	0.334	0.673
	LEGION	0.321	0.289	0.304	0.645
	ReG-TG	0.379	0.341	0.359	0.698
	<i>Diff</i>	(+6.4%)	(+8.2%)	(+7.4%)	(+3.7%)
Dataset LEG	TF-IDF+LR	0.451	0.402	0.425	0.642
	ZestXML	0.523	0.481	0.501	0.731
	LEGION	0.568	0.535	0.551	0.803
	ReG-TG	0.594	0.569	0.581	0.837
	<i>Diff</i>	(+4.6%)	(+6.4%)	(+5.4%)	(+4.2%)
Dataset ReG	TF-IDF+LR	0.186	0.123	0.149	0.523
	ZestXML	0.289	0.225	0.253	0.694
	LEGION	0.301	0.246	0.271	0.778
	ReG-TG	0.423	0.386	0.404	0.813
	<i>Diff</i>	(+40.5%)	(+56.9%)	(+49.1%)	(+4.5%)

Figure 3 further visualizes the cross-dataset pattern reported in Table 3. On Dataset Zest and Dataset LEG, where the label spaces are relatively sparse or curated, ReG-TG obtains stable improvements over the strongest baselines. On Dataset ReG, which contains more open, sparse, and emerging tags, the improvement becomes much larger. Together with the statistical results in Table 4, this visualization supports H1 by showing that ReG-TG not only achieves higher average performance but also maintains consistent advantages across different repository tag distributions.

Table 4. Wilcoxon signed-rank test and Cliff’s delta effect size for H1 model comparisons.

Comparison	Metric	<i>p</i> -Value	Cliff’s δ / Effect Size
ReG-TG vs. TF-IDF+LR	F1-score@5	<0.001	0.61 / large
ReG-TG vs. ZestXML	F1-score@5	<0.01	0.43 / medium
ReG-TG vs. LEGION	F1-score@5	<0.01	0.39 / medium

To further examine whether the observed improvements are statistically reliable, we conducted the Wilcoxon signed-rank test on the per-repository F1-score@5 results between ReG-TG and each baseline method. As shown in Table 4, ReG-TG achieves statistically significant improvements over TF-IDF+LR, ZestXML, and LEGION, with all *p*-values below 0.01. In addition, Cliff’s delta effect size analysis shows a large effect compared with TF-IDF+LR and medium effects compared with ZestXML and LEGION, indicating that the performance gains are not only numerically higher but also statistically meaningful.

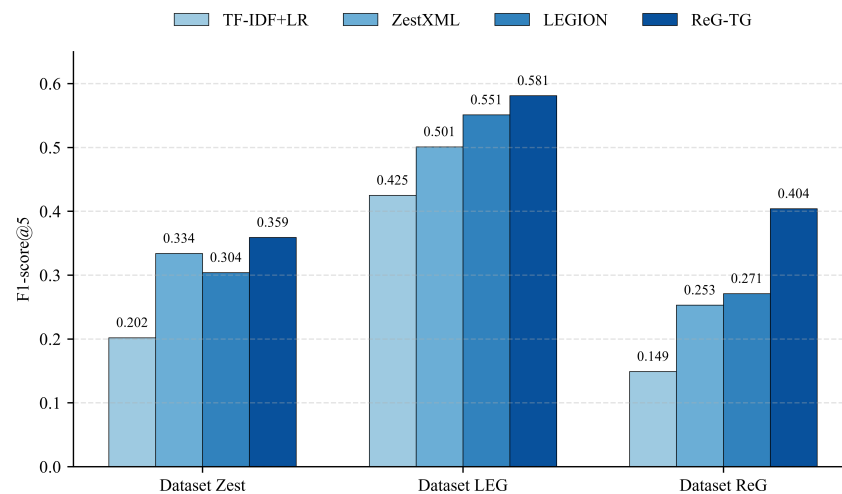


Figure 3. F1-score@5 comparison of ReG-TG and baseline methods across different datasets.

The ZestXML dataset is characterized by an extremely large label space and a considerable number of non-standardized and noisy labels. Experimental results indicate that although traditional ZestXML-based methods, which rely on sparse feature learning, outperform BERT-based models on this dataset, ReG-TG still achieves superior performance. This improvement can be attributed primarily to the semantic retrieval and reasoning-based filtering mechanisms incorporated into the ReG-TG framework. Conventional statistical models largely depend on keyword matching and are therefore susceptible to synonymy and polysemy. In contrast, ReG-TG employs BGE-M3 to generate dense semantic representations for repository text fragments, enabling the retrieval of relevant repositories based on semantic similarity rather than lexical overlap. Furthermore, by incorporating chain-of-thought reasoning into the prompting process, the LLM is able to assess the contextual relevance of candidate tags and filter out certain non-technical or noisy labels (e.g., test and demo), thereby improving prediction accuracy.

The labels in the LEGION dataset are manually curated and exhibit a high degree of standardization. Under this setting, the LEGION baseline model, which is built upon BERT fine-tuning, achieves strong performance, highlighting the effectiveness of pre-trained language models in contextual understanding. The additional gains achieved by ReG-TG are primarily reflected in improved recall. In a controlled label taxonomy, one of the key challenges lies in maintaining consistency with the predefined label granularity and naming conventions. ReG-TG addresses this issue by retrieving the Top-5 most similar repositories and providing their existing labels as demonstration examples for the LLM. This few-shot learning paradigm offers explicit guidance regarding label naming conventions and granularity, encouraging the model to generate standardized tags that are consistent with the reference repositories. Consequently, prediction errors caused by synonymous variants (e.g., generating `deep_learning` instead of the standardized label `deep-learning`) can be effectively reduced.

The largest performance gap between ReG-TG and the baseline methods is observed on the self-constructed Dataset ReG. This dataset contains a large number of emerging technical terms that appeared between 2023 and 2024, such as LLM and LangChain. In this study, these emerging terms should be understood as tags that are poorly covered or absent in the supervised label space of classification-based baselines, rather than tags that are completely absent from all available repository data. Baseline approaches, including TF-IDF+LR, ZestXML, and LEGION, are discriminative models whose prediction space is restricted by the label set observed during training. Consequently, they have limited ability to recommend tags outside this predefined label space, resulting in relatively low

recall. In contrast, ReG-TG adopts a retrieval-augmented generation architecture whose knowledge source extends beyond model parameters to include an external vector index. When repositories related to emerging technologies are present in the retrieval corpus, ReG-TG can retrieve them as contextual evidence and guide the LLM to generate candidate tags that are consistent with the target repository's README. This capability enables ReG-TG to better adapt to open-world and time-sensitive tag recommendation scenarios.

H2: *The complete ReG-TG framework achieves better recommendation performance than its variants with key components removed.*

Approach: To evaluate H2 and quantitatively analyze the contribution of each component in the ReG-TG framework, we conduct systematic ablation studies on Dataset ReG. We choose Dataset ReG because it contains a more open, sparse, and long-tailed label space with many emerging technical terms, making it more suitable for examining the effects of retrieval augmentation, hybrid retrieval, graph enhancement, and structured reasoning. Specifically, by progressively removing key components from the ReG-TG framework, we construct five variant models: (1) w/o Retrieval, where the retrieval-augmented mechanism is removed to assess the impact of incorporating external knowledge; (2) w/o Hybrid Retrieval, where the sparse retrieval component is excluded, retaining only dense vector-based coarse-grained retrieval to analyze the contribution of the hybrid retrieval strategy; (3) w/o Reranker, where the reranking module is eliminated and the Top-5 results from hybrid retrieval are directly used as reference samples to experiment to evaluate the effect of reranking; (4) w/o Graph Enhancement, where the label co-occurrence graph enhancement module is removed to examine its influence on final performance; and (5) w/o CoT Instruction, where the structured chain-of-thought prompting is excluded from the prompt template to investigate its contribution to the overall effectiveness of the ReG-TG framework.

Results: Table 5 shows the performance comparison of the full model and different variants on Precision@5, Recall@5, and F1-score@5.

Table 5. Ablation study results of the full model and different variants on Dataset ReG, bold values mean better.

Model Variants	Precision@5	Recall@5	F1-Score@5	Degradation
ReG-TG	0.423	0.386	0.404	—
w/o Reranker	0.415	0.378	0.396	−2.0%
w/o CoT Inst.	0.409	0.371	0.389	−3.7%
w/o Hybrid Ret.	0.391	0.359	0.374	−7.4%
w/o Graph & Clean	0.398	0.364	0.380	−5.9%
w/o Retrieval	0.375	0.342	0.358	−11.4%

(Observation 2) Each component of ReG-TG contributes to performance improvements, with hybrid retrieval contributing the most. The results presented in Table 5 on Dataset ReG reveal that removing the retrieval module (w/o Retrieval) still yields a relatively strong F1-score of 0.358. However, compared with the full ReG-TG framework, the performance decreases by an absolute F1-score of 0.046. Although this improvement appears less dramatic than the gains observed over the baseline models in the previous section, it objectively reflects the strong intrinsic capability of the DeepSeek-V3 large language model. Benefiting from extensive pre-training on large-scale source code and technical documentation, the model is already capable of understanding README files and inferring many common technical tags (e.g., machine-learning and react). The primary contribution of the retrieval mechanism lies in overcoming the limitations imposed by the model's

training cutoff date and reducing uncertainty in tagging styles. By retrieving up-to-date reference repositories, the model can identify emerging technical terms that were absent from its training data and align its predictions with the tag granularity used by similar repositories (e.g., whether to assign a broad tag such as AI or a more specific tag such as CNN). Consequently, retrieval enables the transition from generally correct predictions to domain-specific and precise tag generation.

Figure 4 provides a visual summary of the module-level differences reported in Table 5. The full ReG-TG framework achieves the highest F1-score@5, while each ablated variant leads to a performance decline. The largest drop occurs when the retrieval module is removed, which is consistent with the preceding analysis that external repository evidence is the central source of domain-specific and up-to-date tag knowledge. The remaining performance drops further indicate that hybrid retrieval, reranking, graph enhancement, and CoT instruction play complementary roles in improving retrieval precision, filtering noisy context, expanding implicit tag evidence, and structuring LLM generation.

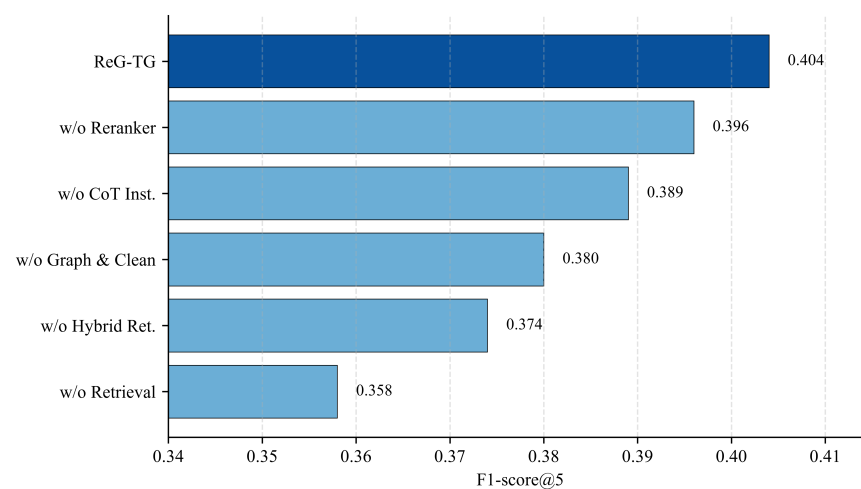


Figure 4. Ablation effect of different ReG-TG components on Dataset ReG.

Compared with the full model, removing the hybrid retrieval component (w/o Hybrid Retrieval), which relies solely on dense retrieval, results in a 7.4% performance degradation. This finding indicates that semantic vector retrieval alone has inherent limitations when applied to software repositories. While dense embeddings are effective at capturing high-level semantic relationships, they are prone to semantic drift when distinguishing between precise technical abbreviations or closely related library names (e.g., PyTorch versus Torch). By incorporating sparse retrieval signals, the hybrid retrieval strategy strengthens exact lexical matching for domain-specific terminology, thereby improving retrieval precision and overall tagging performance.

Eliminating the reranking module (w/o Reranker) leads to a 2.0% decrease in F1-score@5, suggesting that the outputs of hybrid retrieval alone are insufficient for constructing high-quality contextual evidence. Without the Cross-Encoder reranker, the initial Top-50 retrieved repositories contain not only relevant examples but also a substantial amount of noise. The reranker serves as a denoising filter by removing false neighbors that share similar keywords but differ significantly in their underlying technology stacks. Since the quality of the retrieved context directly influences the upper bound of LLM generation performance, reranking plays a crucial role in selecting semantically relevant reference repositories and improving tag prediction accuracy.

Removing chain-of-thought reasoning and instruction guidance (w/o CoT & Instruction), which requires the model to generate tags directly without explicit reasoning steps or

structured instructions, causes a 3.7% decline in F1-score@5. Although DeepSeek-V3 possesses strong implicit reasoning capabilities, explicit step-by-step reasoning is particularly beneficial when processing long and complex documents. By encouraging the model to first compare, analyze, and summarize information before generating tags, chain-of-thought prompting introduces an intermediate reasoning process that reduces the likelihood of overlooking important information, especially content appearing later in the document. This effect is particularly pronounced for repositories that integrate multiple technology stacks.

Finally, removing the graph enhancement and rule-based cleaning module (w/o Graph & Clean) results in an approximately 5.9% reduction in F1-score@5. The primary contribution of this module lies in improving prediction completeness. Large language models tend to generate only the most salient tags, whereas graph enhancement leverages global tag co-occurrence information to encourage the generation of implicit higher-level concepts (e.g., inferring backend from flask), thereby improving recall. In addition, the rule-based cleaning component prevents the inclusion of meaningless or noisy tags such as test. Regardless of the strength of the underlying language model, it remains difficult to completely avoid blindly imitating noisy labels from retrieved examples. Therefore, rule-based constraints serve as a final safeguard to ensure output quality.

H3: *ReG-TG achieves more stable recommendation performance under long-tailed tag distributions than representative baseline methods.*

Approach: To evaluate H3 and reveal the potential of the ReG-TG framework in handling complex application scenarios, considering the significant power-law distribution characteristics of open-source repository tag dataset, this paper constructs a new evaluation dataset based on the dataset used. Specifically, we employ a hierarchical evaluation strategy, dividing the test set tags into three mutually exclusive subsets based on their frequency of occurrence in the training set, i.e., head tags (frequency > 50), body tags (10 < frequency ≤ 50), and tail tags (frequency ≤ 10). Based on this, we conduct comparative experiments on this dataset wusing representative baseline methods to evaluate their performance under long-tailed label distributions.

Results: Table 6 shows the performance analysis of different label frequency groups.

Table 6. Performance analysis on different label frequency groups, bold values mean better.

Datasets	Methods	F1-Score@5	Degradation
Head Tags (freq > 50)	ZestXML	0.555	–
	LEGION	0.581	–
	ReG-TG (Ours)	0.603	–
Body Tags (10 < freq ≤ 50)	ZestXML	0.321	–
	LEGION	0.356	–
	ReG-TG (Ours)	0.482	–
Tail Tags (freq ≤ 10)	ZestXML	0.123	↓77.7%
	LEGION	0.155	↓73.4%
	ReG-TG (Ours)	0.357	↓40.8%

(Observation 3) ReG-TG exhibits relatively good stability when processing long-tailed distributed data. As shown in Table 6, in the Head region, where sufficient training samples are available (e.g., popular tags such as javascript and machine-learning), all models achieve relatively high F1 scores. ReG-TG attains an F1-score@5 of 0.603 in this region, slightly outperforming LEGION (0.581) and ZestXML (0.555). This result indicates that, when handling common high-frequency tags, the retrieval mechanism of ReG-TG can consistently retrieve a large number of high-quality reference repositories, while the large

language model itself has already been exposed to these widely used technical terms during pre-training. Consequently, the performance gap among different models remains relatively small. Under such conditions, prediction performance is primarily constrained by issues related to tag standardization and synonym normalization rather than knowledge scarcity.

Figure 5 complements Table 6 by visualizing how the performance gap changes across label-frequency groups. In the Head region, the three methods show relatively close performance because frequent tags provide sufficient training evidence. However, in the Body and Tail regions, ReG-TG shows a much clearer advantage over ZestXML and LEGION. This pattern supports H3 and is consistent with the retrieval-augmented design of ReG-TG: even when a label has limited training frequency, similar repositories and graph-expanded tag evidence can provide additional contextual support for LLM-based tag generation.

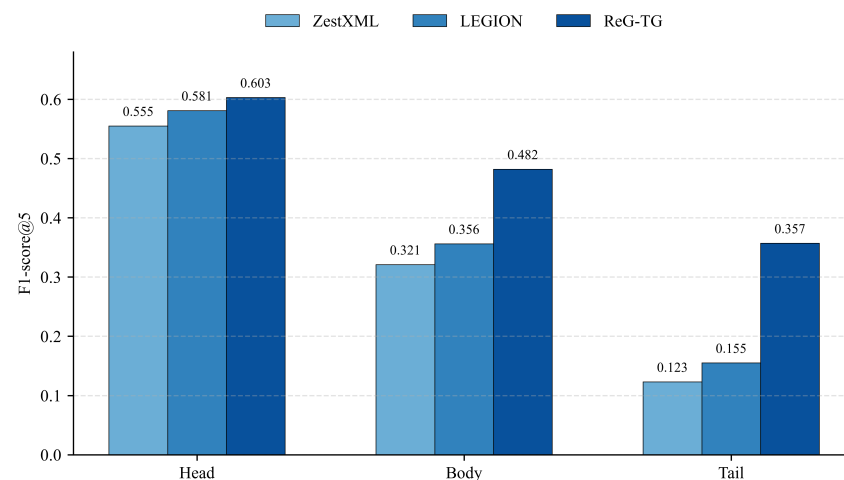


Figure 5. F1-score@5 comparison across Head, Body, and Tail tag groups.

The most significant differences emerge in the Tail region. All models experience performance degradation to varying degrees. The F1 scores of ZestXML and LEGION decrease to 12.34% and 15.45%, respectively. Calculating the performance decay from the Head to the Tail region reveals that both baseline models suffer performance drops exceeding 70%. This observation highlights a fundamental limitation of parameterized models when dealing with low-resource categories. Because tail labels contribute only weak gradient signals during training, these models tend to prioritize optimizing the overall loss by focusing on high-frequency labels, leading to the gradual forgetting of long-tail knowledge and an increased tendency to overfit frequent categories.

In contrast, ReG-TG achieves an F1-score of 0.357 in the Tail region, corresponding to a performance decay rate of 40.8% from Head to Tail. Although its performance is still affected by data sparsity, the degradation is substantially smaller than that of the baseline models. As a result, ReG-TG maintains a reasonable level of effectiveness in predicting low-frequency labels.

These experimental results demonstrate that ReG-TG exhibits superior robustness when handling long-tailed data distributions. The observed advantage primarily stems from differences in the underlying learning mechanisms. Baseline approaches, including ZestXML and LEGION, are parameterized models that rely on sufficient training samples to update model parameters and learn reliable representations for specific labels. For infrequently occurring tail labels, it becomes difficult for these models to establish effective feature-to-label mappings.

By contrast, the retrieval-augmented mechanism adopted by ReG-TG can be viewed as a non-parametric knowledge acquisition approach. Its predictions do not rely solely on knowledge encoded within model parameters; instead, they are supported by retrieving similar examples from an external knowledge base. The experimental findings suggest that even for sparse labels that appear only a few times in the training data, the model can successfully transfer such knowledge to the target repository as long as relevant examples are present in the retrieved Top-5 contextual references. This capability effectively mitigates the prediction failures caused by insufficient training samples and enhances the model's ability to generalize to long-tail labels.

6. Discussion

6.1. Effect of Retrieval Quantity

In the ReG-TG framework, the hyperparameter K determines how many “reference cases” are provided for LLM as context for inference. Choosing this parameter is not simply a matter of “the larger the better,” but rather involves a trade-off between information density and noise interference. Since more reference cases may introduce more noise, while fewer reference cases may lead to insufficient contextual information. Therefore, to explore the impact of hyperparameter K on the performance of the ReG-TG framework and determine the optimal balance between “information gain” and “noise introduction,” we conduct a parameter sensitivity experiment by varying the value of K . We set the value set of K to $\{1, 3, 5, 7, 10\}$ and recorded the changes in Precision@5, Recall@5, and F1-score@5 under different numbers of retrievals. The experimental results are shown in Table 7.

Table 7. Effect of retrieval quantity K on performance, bold values mean better.

Retrieval Quantity (K)	Precision@5	Recall@5	F1-Score@5
1	0.351	0.295	0.320
3	0.402	0.361	0.381
5	0.423	0.386	0.404
7	0.419	0.381	0.399
10	0.406	0.372	0.388

As the number of retrievals K increases, the model's F1-score@ K exhibits a trend of first rising and then falling. Figure 6 provides a more intuitive view of this non-monotonic trend. Precision@5, Recall@5, and F1-score@5 all increase when K grows from 1 to 5, indicating that additional high-quality reference repositories enrich the contextual evidence available to the LLM. However, when K further increases from 5 to 10, all three metrics slightly decrease, suggesting that additional retrieved repositories may introduce weakly relevant examples and noisy labels into the prompt. This visualization reinforces the discussion that retrieval quantity should balance information gain and noise interference rather than simply be maximized. When K is small, increasing the number of reference repositories significantly improves model performance. The F1-score rapidly increases from 0.320 at $K = 1$ to 0.381 at $K = 3$. This indicates that a single reference example is often insufficient to capture the complex technical stack characteristics of the target repository and is also more vulnerable to annotation bias in individual samples. Incorporating more reference examples effectively enriches contextual information, thereby improving recall in tag generation. When K increases to 5, the model achieves its best performance, with an F1-score@ K of 0.404. At this point, precision and recall reach an optimal balance. This suggests that five high-quality reference repositories are sufficient to provide representative label distributions and naming conventions, while not yet introducing excessive noise. When K continues to increase beyond 5, performance no longer improves and instead

shows a slight decline. At $K = 10$, the F1-score drops to 0.388. This phenomenon indicates that blindly increasing the number of retrieved neighbors leads to diminishing returns.

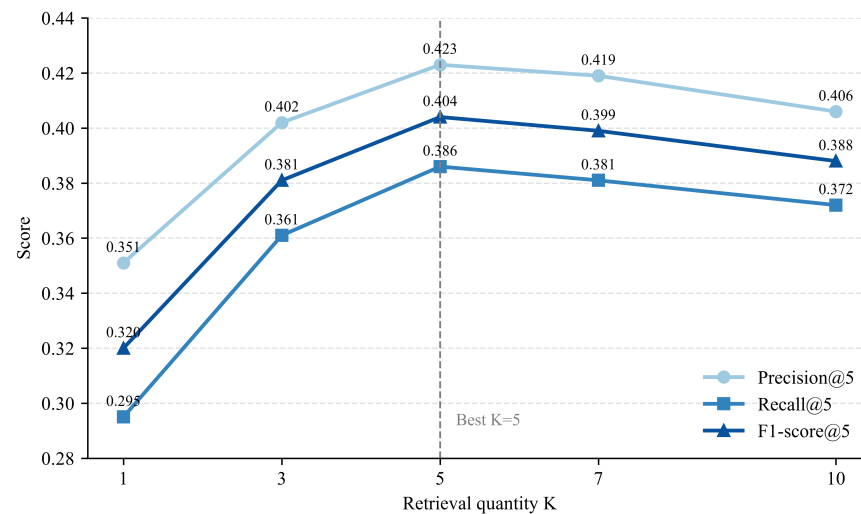


Figure 6. Sensitivity analysis of retrieval quantity K .

The observed trend reflects the trade-off between relevance dilution and noise interference introduced by different values of K . In the retrieval ranking, lower-ranked candidates (e.g., ranks 6 to 10) typically exhibit lower semantic similarity to the target repository compared with the top 5 results. When K becomes too large, these weakly relevant or marginal samples are included in the prompt context. Such examples may contain inconsistent or conflicting labels; for instance, the target repository may use the Vue framework, while the 10th neighbor may be based on React. Although both belong to frontend development, their associated labels may be mutually incompatible. When processing excessively long and noisy contexts, the attention mechanism of large language models may become distracted, leading to suboptimal focus on key features. In addition, the presence of too many reference labels increases uncertainty during generation, resulting in reduced precision. Based on these experimental results, this study concludes that $K = 5$ represents the optimal trade-off between information richness and noise interference under the current dataset and model configuration.

6.2. Inference Efficiency and Practical Deployment

We further analyze the inference efficiency of ReG-TG from the perspective of practical deployment. In our implementation, repository encoding, vector index construction, and tag co-occurrence graph construction are completed offline. Therefore, the online inference stage mainly consists of hybrid retrieval, reranking, graph-enhanced tag expansion, prompt construction, and LLM-based tag generation. We measured the end-to-end inference time on 100 randomly selected repositories from Dataset ReG. The average latency for processing a single repository is approximately 5.8 s. Among different modules, hybrid dense-sparse retrieval and graph-based tag expansion are relatively lightweight, usually requiring less than 0.3 s in total. The reranking stage over the Top-50 candidates takes about 0.9 s on average, while the LLM generation stage is the dominant cost, accounting for approximately 4.5 s per repository. This indicates that the main efficiency bottleneck of ReG-TG lies in LLM-based structured generation rather than retrieval or graph expansion.

From a deployment perspective, ReG-TG is more suitable for offline or semi-online repository tag recommendation scenarios, such as periodically recommending tags for newly created repositories, updating repository metadata, or assisting maintainers during repository release. Since the retrieval index and tag graph can be precomputed and

incrementally updated, the online cost can be further reduced by caching repository embeddings, reusing retrieved reference results for similar repositories, and limiting the number of reranked candidates. In latency-sensitive scenarios, the LLM generation module can also be replaced by a smaller instruction-tuned model or a distilled tag generator, while retaining the same retrieval and graph-enhancement components. Therefore, although ReG-TG has higher inference latency than pure classification models, its modular design provides a practical trade-off between recommendation quality and deployment efficiency.

6.3. Case Study and Error Analysis

To further illustrate how the components of ReG-TG contribute to the final recommendation results and to analyze its limitations, we discuss one successful case and one failed case.

In the successful case, the target repository is a Python project that implements Transformer- and BERT-based models for text classification. The README contains clear technical signals such as Transformer, BERT, PyTorch, and text classification. During hybrid retrieval, the dense retrieval branch recalls repositories with similar neural text classification functions, while the sparse retrieval branch preserves exact matches of key technical terms such as BERT and PyTorch. The BGE-Reranker then filters the Top-50 candidates and keeps the most semantically relevant repositories as reference examples. These retrieved repositories provide standard GitHub-style topic labels such as pytorch, transformer, and text-classification. Based on these reference tags, the PMI-based tag co-occurrence graph further expands related concepts such as natural-language-processing, bert, and deep-learning. Finally, the structured prompt provides the LLM with three complementary evidence sources: the target README content, the retrieved repository examples, and the graph-enhanced tag candidates. The final generated tags include pytorch, transformer, bert, natural-language-processing, and text-classification. This case shows that hybrid retrieval provides concrete local evidence, reranking improves the quality of the reference context, graph enhancement supplements implicit higher-level concepts, and structured LLM reasoning integrates these signals into a normalized tag list.

In the failed case, the target repository is a Vue-based frontend project for building reusable web UI components. Its README contains broad terms such as component, frontend, UI, and JavaScript, which are also common in React component libraries. As a result, the hybrid retrieval module may retrieve some React-related repositories because they are semantically close and share many frontend keywords. Although the reranker removes many irrelevant candidates, a few semantically similar but framework-incompatible repositories may still remain in the Top-K context. The graph enhancement module can expand general frontend-related concepts such as javascript, frontend, and web-development, but it cannot always distinguish mutually exclusive framework-specific tags such as vue and react. Consequently, ReG-TG may correctly generate vue, frontend, and javascript but incorrectly include react in the final tag list. This failure indicates that ReG-TG is still sensitive to noisy retrieval context and fine-grained distinctions between closely related technology ecosystems. Future work could incorporate stricter entity-level constraints, framework-specific conflict detection, or retrieval confidence calibration to reduce such errors.

7. Conclusions

This paper presents ReG-TG, a novel framework for open-source repository tag recommendation that integrates multi-granularity hybrid retrieval, knowledge graph augmentation, and chain-of-thought reasoning with large language models. ReG-TG achieves accurate recall of high-quality reference samples through a dual-path retrieval mechanism that combines dense semantic representations with sparse symbolic signals, further en-

hanced by a re-ranking strategy. To address the challenges of long-tail and sparse tags, ReG-TG leverages tag co-occurrence graphs and semantic neighborhood expansion to effectively inject complementary domain knowledge. In addition, structured chain-of-thought prompting is introduced to guide the large language model toward generating standardized, comprehensive, and interpretable tag sets. Experimental results on multiple datasets demonstrate that ReG-TG consistently outperforms representative baseline methods in terms of Precision@K, Recall@K, and F1-score@K. Moreover, the framework exhibits strong robustness and generalization under long-tail distributions. ReG-TG provides a scalable and effective solution for repository tag recommendation in open-world scenarios and offers new insights into the application of retrieval-augmented generation paradigms in software engineering tasks.

Author Contributions: M.W. and S.W.: conceptualization; methodology; software; validation; formal analysis; investigation; resources; data curation; writing—original draft preparation; writing—review and editing; visualization. W.L. and Y.Z.: software; validation; data curation; writing—review and editing; visualization; supervision; project administration; funding acquisition. All authors have read and agreed to the published version of the manuscript.

Funding: This research is supported by the Shenzhen Science and Technology Program under grant No. JCYJ20240813152507009, the Basic and Applied Basic Research Foundation of Guangdong Province under grant No. 2026A1515010518, and the Fundamental Research Funds for the Central Universities under grant No. NJ2025054, the Foundation of the Key Laboratory of Safety-Critical Software (Nanjing University of Aeronautics and Astronautics), the Collaborative Innovation Center of Novel Software Technology and Industrialization and the High Performance Computing Platform of Nanjing University of Aeronautics and Astronautics.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. GitHub. 2018. Available online: <https://github.com/> (accessed on 15 July 2026).
2. SourceForge. 2018. Available online: <https://sourceforge.net/> (accessed on 15 July 2026).
3. GitHub. Classifying Your Repository with Topics. 2025. Available online: <https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/classifying-your-repository-with-topics> (accessed on 3 June 2026).
4. GitHub. Topic Suggestions for Millions of Repositories. 2017. Available online: <https://github.blog/engineering/user-experience/topics/> (accessed on 15 July 2026).
5. Izadi, M.; Heydarnoori, A.; Gousios, G. Topic Recommendation for Software Repositories Using Multi-Label Classification Algorithms. *Empir. Softw. Eng.* **2021**, *26*, 93. [CrossRef]
6. Widayarsi, R.; Zhao, Z.; Le Cong, T.; Kang, H.J.; Lo, D. Topic Recommendation for GitHub Repositories: How Far Can Extreme Multi-Label Learning Go? In *Proceedings of the 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*; IEEE: Piscataway, NJ, USA, 2023; pp. 167–178.
7. Di Sipio, C.; Rubel, R.; Di Ruscio, D.; Nguyen, P.T. A Multinomial Naïve Bayesian (MNB) Network to Automatically Recommend Topics for GitHub Repositories. In *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering*, Trondheim, Norway, 15–17 April 2020; pp. 71–80.
8. LeClair, A.; Eberhart, Z.; McMillan, C. Adapting Neural Text Classification for Improved Software Categorization. In *Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*; IEEE: Piscataway, NJ, USA, 2018; pp. 461–472.
9. Gupta, N.; Bohra, S.; Prabhu, Y.; Purohit, S.; Varma, M. Generalized Zero-Shot Extreme Multi-Label Learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, Virtual, 14–18 August 2021; pp. 527–535.

10. Cai, X.; Zhu, J.; Shen, B.; Chen, Y. Greta: Graph-Based Tag Assignment for GitHub Repositories. In *Proceedings of the 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*; IEEE: Piscataway, NJ, USA, 2016; Volume 1, pp. 63–72.
11. Izadi, M.; Nejati, M.; Heydarnoori, A. Semantically-Enhanced Topic Recommendation Systems for Software Projects. *Empir. Softw. Eng.* **2023**, *28*, 50. [[CrossRef](#)]
12. Brown, T.B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language Models Are Few-Shot Learners. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 1877–1901.
13. Zanartu, F.; Treude, C.; Cartaxo, B.; Borges, H.S.; Moura, P.; Wagner, M.; Pinto, G. Automatically Categorising GitHub Repositories by Application Domain. *arXiv* **2022**, arXiv:2208.00269.
14. Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; Yih, W.t.; Rocktäschel, T.; et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 9459–9474.
15. Karpukhin, V.; Oguz, B.; Min, S.; Lewis, P.; Wu, L.; Edunov, S.; Chen, D.; Yih, W.t. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Online, 16–20 November 2020; pp. 6769–6781. [[CrossRef](#)]
16. Dang, Y.T.; Le-Cong, T.; Nguyen, P.T.; Bui, A.M.T.; Nguyen, P.T.; Le, B.; Huynh, Q.T. LEGION: Harnessing Pre-Trained Language Models for GitHub Topic Recommendations with Distribution-Balance Loss. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, Salerno, Italy, 18–21 June 2024; pp. 181–190. [[CrossRef](#)]
17. Tian, K.; Revelle, M.; Poshyvanyk, D. Using Latent Dirichlet Allocation for Automatic Categorization of Software. In *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories*; IEEE: Piscataway, NJ, USA, 2009; pp. 163–166.
18. Linares-Vásquez, M.; McMillan, C.; Poshyvanyk, D.; Grechanik, M. On Using Machine Learning to Automatically Classify Software Applications into Domain Categories. *Empir. Softw. Eng.* **2014**, *19*, 582–618. [[CrossRef](#)]
19. Wang, T.; Wang, H.; Yin, G.; Ling, C.X.; Li, X.; Zou, P. Mining Software Profile across Multiple Repositories for Hierarchical Categorization. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance*; IEEE: Piscataway, NJ, USA, 2013; pp. 240–249.
20. Sharma, A.; Thung, F.; Kochhar, P.S.; Sulistya, A.; Lo, D. Cataloging GitHub Repositories. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, Karlskrona, Sweden, 15–16 June 2017; pp. 314–319.
21. Nguyen, A.T.; Nguyen, T.N. Automatic Categorization with Deep Neural Network for Open-Source Java Projects. In *Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*; IEEE: Piscataway, NJ, USA, 2017; pp. 164–166.
22. Rokon, M.O.F.; Yan, P.; Islam, R.; Faloutsos, M. Repo2Vec: A Comprehensive Embedding Approach for Determining Repository Similarity. In *Proceedings of the 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*; IEEE: Piscataway, NJ, USA, 2021; pp. 355–365.
23. Qian, Y.; Zhang, Y.; Wen, Q.; Ye, Y.; Zhang, C. Rep2Vec: Repository Embedding via Heterogeneous Graph Adversarial Contrastive Learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, Washington, DC, USA, 14–18 August 2022; pp. 1390–1400.
24. Zhang, Y.; Xu, F.F.; Li, S.; Meng, Y.; Wang, X.; Li, Q.; Han, J. HiGitClass: Keyword-Driven Hierarchical Classification of GitHub Repositories. In *Proceedings of the 2019 IEEE International Conference on Data Mining (ICDM)*; IEEE: Piscataway, NJ, USA, 2019; pp. 876–885.
25. Zhang, Y.; Meng, Y.; Huang, J.; Xu, F.F.; Wang, X.; Han, J. Minimally Supervised Categorization of Text with Metadata. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, Virtual, 25–30 July 2020; pp. 1231–1240.
26. Zhang, Y.; Chen, X.; Meng, Y.; Han, J. Hierarchical Metadata-Aware Document Categorization under Weak Supervision. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, Online, 8–12 March 2021; pp. 770–778.
27. Chen, J.; Xiao, S.; Zhang, P.; Luo, K.; Lian, D.; Liu, Z. M3-Embedding: Multi-Linguality, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. In *Proceedings of the Findings of the Association for Computational Linguistics: ACL 2024*, Virtual, 11–16 August 2024; pp. 2318–2335.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.