

When Retriever Meets Generator: A Joint Model for Code Comment Generation

Tien P. T. Le, Anh M. T. Bui, Huy N. D. Pham

Hanoi University of Science and Technology

Hanoi, Vietnam

tien.lpt207633@sis.hust.edu.vn, anhbtm@soict.hust.edu.vn

Alessio Bucaioni

Mälardalen University

Västerås, Sweden

alessio.bucaioni@mdu.se

Phuong T. Nguyen

University of L'Aquila

L'Aquila, Italy

phuong.nguyen@univaq.it

Abstract—Automatically generating concise, informative comments for source code can lighten documentation effort and accelerate program comprehension. Retrieval-augmented approaches first fetch code snippets with existing comments and then synthesize a new comment, yet retrieval and generation are typically optimized in isolation, allowing irrelevant neighbors to propagate noise downstream. To tackle the issue, we propose a novel approach named RAGSum with the aim of both effectiveness and efficiency in recommendations. RAGSum is built on top of fuse retrieval and generation using a single CodeT5 backbone. We report preliminary results on a unified retrieval-generation framework built on CodeT5. A contrastive pre-training phase shapes code embeddings for nearest-neighbor search; these weights then seed end-to-end training with a composite loss that (i) rewards accurate top-k retrieval; and (ii) minimizes comment-generation error. More importantly, a lightweight self-refinement loop is deployed to polish the final output. We evaluated the framework on three cross-language benchmarks (Java, Python, C), and compared it with three well-established baselines. The results show that our approach substantially outperforms the baselines with respect to the BLEU, METEOR, and ROUTE-L scores. These early findings indicate that tightly coupling retrieval and generation can raise the ceiling for comment automation and motivate forthcoming industrial replications and qualitative developer studies.

Index Terms—code comment generation, retrieval augmented generation, pre-trained language model

I. INTRODUCTION

Up-to-date, readable comments accelerate program comprehension, reduce onboarding time and maintenance cost [1]. Yet surveys show that 60–70% of developers routinely encounter missing or obsolete comments [2], and such mismatches raise the likelihood of defect-inducing changes by roughly 1.5x. Automating comment generation has therefore become an active line of inquiry at the intersection of software engineering and natural-language processing.

Early work tackled the problem with template rules and information-retrieval (IR) heuristics. Template systems extract salient tokens and stitch them into fixed linguistic patterns [3], [4]; IR systems locate code fragments similar to a query and reuse their comments [5], [6]. Although lightweight, these methods often misalign with the precise semantics of the target snippet. The advent of neural sequence-to-sequence models reframed comment generation as a machine-translation task from source code to natural language [7], [8]. Such models learned richer representations, but even the best variants struggled to

bridge the modality gap between programming languages and English, leading to generic or inaccurate summaries [9]. To mitigate these weaknesses, recent studies blended IR with neural generation. The model first retrieves code–comment exemplars and then conditions the decoder on that context [10]–[12]. Although this paradigm has shown promise, existing systems typically train retrieval and generation components separately, which can lead to irrelevant neighbors introducing noise into the generated comments and hinder overall performance. To address this issue, Li et al. [12] proposed EditSum that refines retrieved comments to better align with the semantics of the input code query. While EditSum captures essential keywords from the input code snippet during comment generation through its self-editing pipeline, the presence of irrelevant retrieved code can still degrade performance. A joint training approach for simultaneously optimizing the retriever and generator has been employed in JOINTCOM [13] and later, CMR-Sum [14], to enhance the retrieval of relevant comments. These approaches aim to achieve a balance between the two tasks’ performance by employing a shared learning framework. While JOINTCOM treated retrieval and generation as two separate models, sharing weights between them during training; CMR-Sum proposed an extractor that integrates generated and retrieved comments within a unified framework, aiming to align them using an attention mechanism. We argue that *though these approaches outperform earlier methods based on separate training paradigms, treating the retriever and generator as distinct tasks may still hinder the overall performance of comment generation*. As a motivating example, in Figure 1 we show the results of using JOINTCOM and CMR-Sum to generate comments for a given input code query. It is evident that compared to the ground-truth comment and the input code, the results generated by both JOINTCOM and CMR-Sum exhibit significant semantic inaccuracies.

To bridge such a gap, we *report preliminary results* on a novel model to fuse retrieval and generation within a single CodeT5 [15] backbone, with the aim of both effectiveness and efficiency in the final recommendations. First, an initial contrastive phase shapes the encoder for nearest-neighbour search. Second, a composite objective tunes both encoder and decoder end-to-end, rewarding accurate top-k retrieval and fluent, context-aware comments. Third, a lightweight self-refinement loop further polishes the output. To study

```

1 public void revert(final VirtualFile root, final List<FilePath> files)
2- throws VcsException {
3- for (List<String> paths : VcsFileUtil.chunkPaths(root, files)) {
4-   GitSimpleHandler handler = new GitSimpleHandler(
5     myProject, root, GitCommand.CHECKOUT
6   );
7   handler.addParameters(_STR);
8   handler.endOptions();
9   handler.addParameters(paths);
10  handler.run();
11 }
12 }

```

Retrieved comment

CMR-Sum: method to clear source and target folders . this ensures that source and target folder are ready to use for next junit test case .

JOINTCOM: reboot throw localrepositoryexception if exit value = 66 or exit value != 0 also throw local repository exception if not exited normally

Ground truth

reverts the list of files we are passed .

Fig. 1. Example of retrieved comments by CMR-Sum and JOINTCOM.

RAGSum, we evaluated it on three cross-language benchmarks, i.e., Java, Python, C and compared it with three well-established baselines, i.e., CMR-Sum [14], JOINTCOM [13], and Llama-3.1-8B [16]. The experimental results showed that RAGSum gains significant improvements with respect to the baselines. These early findings indicate that tightly coupling retrieval and generation can raise the ceiling for comment automation and motivate forthcoming industrial replications and qualitative developer studies.

The main contributions of our work are as follows.

- **We developed RAGSum**, a practical approach to code comment generation on top of contrastive pre-training phase shapes code embeddings for nearest-neighbor search.
- **We conducted an empirical evaluation** using three real-world datasets to study RAGSum’s performance and compare it with three well-established baselines.
- **We published online a replication package** including the data curated and tool developed through this work to foster future research [17].

The paper is organized as follows. In Section II, we review the related work. Section III explains in detail the proposed approach. The empirical evaluation to study the performance of RAGSum is presented in Section IV. Afterward, in Section V, we report and analyze the experimental results. Finally, Section VI sketches future work, and concludes the paper.

II. RELATED WORK

Deep learning can automatically learn pattern features from large-scale datasets, several studies have explored deep learning-based methods for code summarization [18]. With the advantage of transformer, sequence-to-sequence (Seq2Seq) architectures bring significant improvements for generating summaries of code. Transformer-based models [19], [20] have enhanced the semantic understanding of comment generation. Several approach focused on leveraging AST as input of encoder-decoder model [21]. However, generation models often struggle with issues such as hallucination and limited access to external knowledge, which can hinder the accuracy and completeness of the generated summaries To address this limitations, Zhang et al. [22] introduced Rencos, a retrieval-based neural approach for source code summarization but lack

dynamic integration during generation. Another framework for comment generation – DECOM [23] with the multistage deliberation process which use the keywords from source code and the comment of retrieved sample to enhance the performance. However, these approaches treated the retriever and generator as separate components, training them in isolation and thereby limiting their potential synergy. Recent studies [13], [14] have proposed combining retrievers and generators to leverage their complementary strengths. Many research focus on the ability of LLMs in code comment generation. Recent research has concentrated on exploring various prompting techniques to better harness the potential of LLMs in this task [24] but the summaries produced by LLMs often differ significantly in expression from reference and tend to include more detailed information than those generated by traditional models [25].

III. PROPOSED APPROACH

In this paper, we introduce RAGSum—our proposed approach for Automated Code Comment Generation using Retrieval Augmented Generation, which can enhance the traditional RAG. The overall architecture of RAGSum is shown in Figure 2. RAGSum employs a Encoder-Decoder CodeT5 [15] model with joint fine-tuning to concurrently leverage the performance of Retriever and Generator for code comment generation. Our proposed approach consists of three key components: (i) Self-Supervised Training of Retriever; (ii) Retriever-Generator Joint Fine-tuning; and (iii) Self-Refinement Process.

A. Self-Supervised Training of Retriever

Recent research in code retrieval has underscored the value of self-supervised contrastive learning for effective code representation [11], [26]. Following this paradigm, we employ a contrastive learning approach to pre-train the encoder of the backbone CodeT5 that captures representations of both code snippets and comments. In particular, we introduce a multi-modal contrastive learning approach to jointly learn representations across the two modalities. Given a code query q_i and its corresponding comment c_i , the CodeT5 encoder first produces two representation vectors, which, for simplicity, are also denoted as q_i and c_i , respectively. We fine-tune the encoder to simultaneously enhance both code-to-code and code-to-comment retrieval performance, using *in-batch negatives* technique [27]. As such, for each training instance (q_i, c_i) in a training batch $\mathcal{B}$, two contrastive loss functions will be computed as follows.

$$\mathcal{L}_{q2q} = -\log \frac{e^{\text{sim}(q_i, q_i^+)/\tau}}{e^{\text{sim}(q_i, q_i^+)/\tau} + \sum_{j \in \mathcal{B}} e^{\text{sim}(q_i, q_j^-)/\tau}} \quad (1)$$

$$\mathcal{L}_{q2c} = -\log \frac{e^{\text{sim}(q_i, c_i)/\tau}}{e^{\text{sim}(q_i, c_i)/\tau} + \sum_{j \in \mathcal{B}, j \neq i} e^{\text{sim}(q_i, c_j)/\tau}} \quad (2)$$

In Equation 1, q_i^+ denotes the representation vector of the positive code query of q_i . In the self-supervised learning setting, the input code query q_i is passed through the encoder twice to produce two representation vectors, q_i and q_i^+ . The

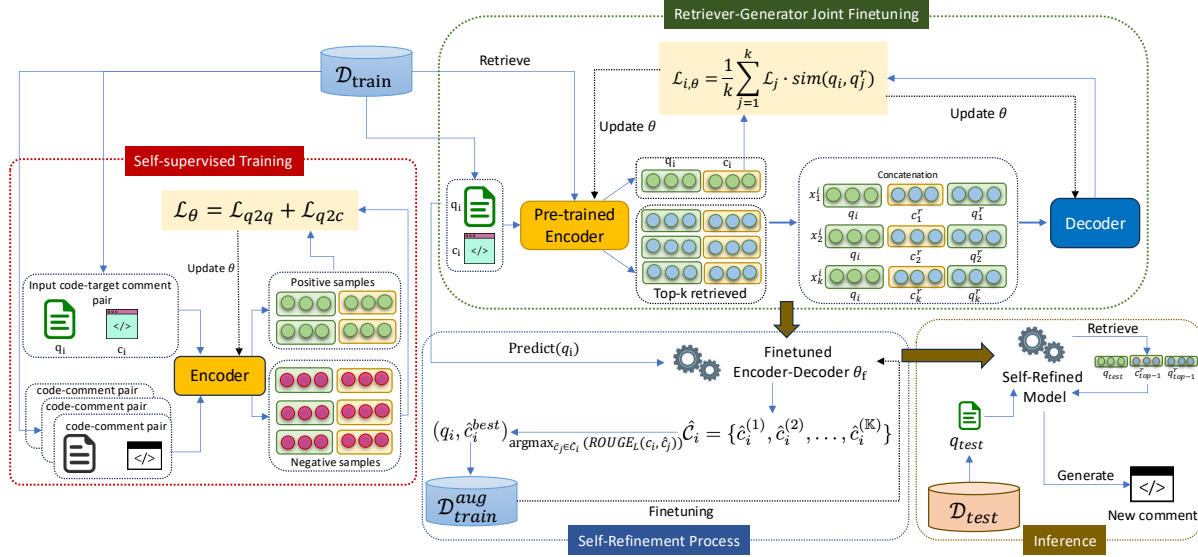


Fig. 2. The overall architecture of RAGSum.

model is trained to minimize the distance between these two representations while maximizing the distance to other code queries in the batch $\mathcal{B}$, which serve as negative samples (q_i^-). $\text{sim}(\cdot)$ denotes the cosine similarity of two vectors. τ is the temperature from contrastive learning [11]. The second loss function, $\mathcal{L}_{q2c}$, aims to minimize the similarity between the input code q_i and its corresponding comment c_i , while maximizing the similarity gap with comments from negative samples in the batch. By pulling semantically aligned code-comment pairs closer and pushing apart misaligned ones, this objective helps the model more effectively distinguish between relevant and irrelevant pairs, thereby enhancing its understanding of the semantic relationship between code and comments. This self-supervised training phase aims to enhance the encoder's performance on the retrieval task and strengthens the ability of the proposed approach to capture semantic representations, enabling it to generate comments that more accurately reflect the underlying logic of the code.

B. Retriever-Generator Joint Finetuning

We employ the Encoder-Decoder architecture of the CodeT5 backbone to generate summaries for a given code query. The pre-trained encoder from the previous phase is used to embed code snippets and comments from the training dataset. Subsequently, we jointly fine-tune both this encoder (for Retriever) and the decoder (for Generator) in a unified training process. For each code-comment pair (q_i, c_i) in the training set $\mathcal{T}$, the Retriever selects the top- k most relevant code-comment pairs by ranking the cosine similarity between q_i and all other code snippets $q_j \in \mathcal{T}, j \neq i$, resulting in a retrieval set $\mathcal{R}_i = \{(q_1^r, c_1^r), \dots, (q_k^r, c_k^r)\}$. The input query q_i is then concatenated with each retrieval exemplar to form k augmented input sequence: $x_j^i = q_i \oplus c_j^r \oplus q_j^r$. $\{x_j^i\}_{j=1}^k$ are

then fed into the Decoder to estimate $p(c_i|x_j^i)$ using the Cross Entropy loss function:

$$\mathcal{L}_j = - \sum_{t=0}^{|c_i|} \log(p(c_i^t | c_{i,<t}, x_j^i))$$

In the joint finetuning process, the Retriever will be updated based on the feedback from Generator by taking into account the contribution of each retrieved result. The joint loss function $\mathcal{L}_i$ of the input code query q_i is then computed as in Equation 3.

$$\mathcal{L}_i = \frac{1}{k} \sum_{j=1}^k \mathcal{L}_j \cdot \nu_j \quad (3)$$

where $\nu_j = \text{sim}(q_i, q_j^r)$ represents the contribution of the retrieved exemplar q_j^r to the generation of the comment c_i of q_i .

C. Self-Refinement Process

Auto-regressive sequence generation models commonly suffer from exposure bias problem and hallucination between training and inference phase [28]. To tackle this, we introduce a lightweight post-generation refinement module that improves the faithfulness and fluency of generated comments. For each training example, given an input q_i and its corresponding ground-truth comment c_i , we generate $\mathbb{K}$ candidate comments using the joint fine-tuned model $\mathbb{M}$ resulting from the second phase.

$$\hat{\mathcal{C}}_i = \{\hat{c}_i^{(1)}, \hat{c}_i^{(2)}, \dots, \hat{c}_i^{(\mathbb{K})}\} \text{ where } \hat{c}_i^{(j)} = \mathcal{M}(q_i)$$

We compute the ROUGE-L score between each candidate comment and the reference comment c_i . The candidate with the highest score is selected to build the augmented dataset $\mathcal{D}_{\text{aug}} = \{(q_i, \hat{c}_i^{\text{best}})\}$, which is then used to further fine-tune the joint retrieval-generation model. This allows the model to

leverage self-generated, high-quality comments that are most semantically aligned with the ground truth.

D. Inference

During inference phase, an input code query q_t is only concatenated with the highest retrieval score exemplar q_r^t, c_r^t to generate the comment.

$$c_t = \mathcal{M}_{refined}(q_t \oplus c_r^t \oplus q_t^r)$$

IV. EVALUATION

We evaluate RAGSum through a series of experiments on established code summarization benchmarks.

A. Research Questions

RQ₁: *How effective is the Retriever component of RAGSum in retrieving relevant results compared to the baselines?* This research question examines the effectiveness of the retriever component in comparison to baseline approaches. We evaluate the relevance of the comments retrieved by three different retrievers—RAGSum, JOINTCOM, and CMR-Sum—by comparing them to the reference comment of the input code.

RQ₂: *How effective is RAGSum compared to the baselines?* We compare the efficiency of our approach to baselines. For a fair comparison and consistency in evaluation, we reproduce JOINTCOM and CMR-Sum with the same experimental setting as provided in the original studies [13], [14].

- CMR-Sum [14] introduced a joint retriever-generator framework for code summarization, where the retriever and generator are finetuned independently. An extractor is then used to align the retrieved code with the generated comment, refining the final output.
- JOINTCOM [13] also employed a joint retriever-generator paradigm for comment generation, but treated the retriever and generator as separate models, sharing weights between them during training.
- Llama-3.1-8B [16] is a Large Language Model (LLM) developed by Meta AI. Due to resource constraints, we use the 8B-parameter version for inference. In our experiments, the LLM serves as the generator in the RAG framework, with one-shot and few-shot exemplars retrieved using CodeT5 embeddings.

RQ₃: *How does each component of RAGSum contribute to its overall performance?* We propose strategies to increase model performance, including training of encoder, retriever-generator integration, and a self-refinement mechanism. This RQ ascertains how each individual component contributes to the overall performance.

B. Benchmark Datasets

We evaluate our approach on JCSD, PCSD and CCSD—the most popular benchmark datasets for code summarization. Specifically, the Java dataset [8] comprises pairs of source code and corresponding comments from well-known GitHub repositories, the Python dataset initially gathered by Baron et

TABLE I
STATISTIC OF THE DATASETS.

Dataset	JCSD	PCSD	CCSD
Training set	69,708	55,538	84,315
Validation set	8,714	18,505	4,432
Testing set	6,489	18,142	4,203

al. [29], the dataset JCSD and PCSD was preprocessed by Lu et al. [13] to remove duplication. The C Code dataset (CCSD) was crawled by Liu et al. [27] with 95k function-summary pairs. The statistics of datasets are shown in Table I.

C. Evaluation metrics

Following prior work [13], [14], [30], we evaluate RAGSum using BLEU [31], ROUGE-L [32], METEOR [33], and CIDER [34]. Corpus-level BLEU captures overall performance while Sentence-level BLEU evaluates individual predictions. ROUGE-L evaluates the similarity between generated and reference texts using the longest common subsequence. METEOR offers improvements over traditional metrics by considering linguistic aspects such as synonymy, stemming, and word order. CIDER computes the relevance of key information.¹

D. Implementation Details

RAGSum is implemented using the pre-trained CodeT5-base model ², which has 12 encoder and decoder layers with a hidden size of 768. The batch size is set to 24 with a default learning rate 5×10^{-5} for fine-tuning and 1×10^{-5} for self-improvement stage, temperature τ is set to 0.2. The model is pretrained for 1 epoch on the retrieval task, followed by 10 fine-tuning epochs and 5 self-refinement epochs. We use a beam size of 10 and generate 10 candidate comments during the self-refinement process.

V. RESULTS AND DISCUSSION

A. RQ₁: *How effective is the Retriever component of RAGSum in retrieving relevant results compared to the baselines?*

We employed the retriever components of RAGSum, JOINTCOM, and CMR-Sum to retrieve the most relevant code corresponding to each input code in the test set, and then calculated the ROUGE-L score between the retrieved comment and the ground truth comment. Figure 3 shows the ROUGE-L score distributions across three datasets. Overall, RAGSum consistently attains a higher median score compared to both JOINTCOM and CMR-Sum.

¹Due to space limitations, we omit the details of these metrics.

²Salesforce/codet5-base

TABLE II
COMPARISON OF RAGSum WITH THE BASELINES.

Approach	JCSD					PCSD					CCSD				
	C-B	S-B	RL	M	C	C-B	S-B	RL	M	C	C-B	S-B	RL	M	C
Llama3.1RAG 1-shot [16]	15.08	14.61	35.26	18.33	1.69	11.66	7.08	21.28	14.09	0.96	16.46	12.09	34.01	19.27	1.76
Llama3.1RAG n-shot [16]	15.15	14.51	36.3	18.97	1.74	20.89	13.28	35.96	24.13	1.85	19.65	13.38	37.57	20.55	2.04
CMR-Sum [14]	23.53	23.24	46.59	20.5	2.7	28.89	22.41	52.42	23.94	2.9	21.72	15.85	40.96	19.8	2.45
JOINTCOM [13]	26.09	26.53	50.22	22.02	2.99	27.89	21.05	52.6	23.84	2.82	26.32	19.99	46.15	22.82	2.91
RAGSum	27.16	27.94	51.54	22.71	3.13	33.0	26.11	56.15	26.53	3.28	27.95	21.36	47.35	23.76	3.03

TABLE III
ABLATION STUDY.

Approach	JCSD					PCSD					CCSD				
	C-B	S-B	RL	M	C	C-B	S-B	RL	M	C	C-B	S-B	RL	M	C
Only Generator	13.33	14.16	41.3	15.63	1.88	21.42	15.36	48.64	21.3	2.28	18.28	12.71	39.13	18.83	2.16
RAGSum <i>w/o</i> combined	24.02	24.03	48.37	20.92	2.77	28.69	22.07	52.64	24.24	2.88	23.1	17.23	42.89	21.13	2.59
RAGSum <i>w/o</i> pretrained + SR	27.12	27.25	50.59	22.47	3.05	31.69	24.8	55.04	25.73	3.16	27.19	20.84	46.75	23.27	2.96
RAGSum <i>w/o</i> SR	27.06	27.8	51.24	22.62	3.1	32.57	25.66	55.65	26.22	3.23	27.76	21.26	47.18	23.54	3.02
RAGSum	27.16	27.94	51.54	22.71	3.13	33.0	26.11	56.15	26.53	3.28	27.95	21.36	47.35	23.76	3.03

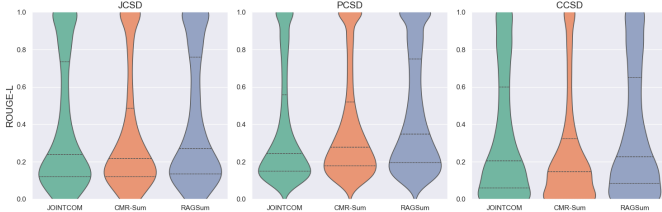


Fig. 3. Distribution of Retrieved Comments and Targets Across Methods

The upper quartile of RAGSum extends beyond 0.75, while JOINTCOM and CMR-Sum are remain below this level on JCSD dataset. The PCSD reveals a clearly distinction, where RAGSum achieves the highest median approximately 0.37, followed by CMR-Sum with around 0.3 and JOINTCOM with 0.25. The distribution scores of RAGSum generally shifts upward, indicating that the top-1 returned comments are more relevant to the target comments.

Answer to RQ₁: The retriever of RAGSum is more effective and robust than the baseline methods in fetching relevant information.

B. RQ₂: How effective is RAGSum compared to the baselines?

We compare our approach to the baselines on three datasets, with results summarized in Table II. The metrics include **C-B** (Corpus-BLEU), **S-B** (Sentence-BLEU), **RL** (ROUGE-L), **M** (METEOR), and **C** (CIDEr). In Java dataset, compared to the best baselines JOINTCOM, RAGSum increases 4.1%, 5.31%, 2.63%, 3.13% and 4.68% in terms of C-BLEU, S-BLEU, ROUGE-L, METEOR, and CIDEr, respectively. With PCSD, our approach significantly outperforms, RAGSum achieves 33.0, 26.11, 56.15, 26.53 and 3.28 points with improvements of 14.23%, 16.51%, 7.12%, 10.82%, and 13.1%, respectively, compared to CMR-Sum. These gains reflect the enhanced

alignment between the retriever and generator, which enables more accurate and semantically relevant summary generation. For the C dataset, the performance of RAGSum remains competitive. While the margins over JOINTCOM are narrower due to the inherently lower redundancy and more complex structure of C programs, RAGSum still achieves a gain of 2.6% in ROUGE-L and a 6.19% boost in C-BLEU, suggesting its strong generalization even under challenging conditions.

Moreover, across three benchmarks, RAGSum consistently outperforms LLaMa-3.1-8B in both 1-shot and n-shot configurations. Notably, RAGSum leverages relevant knowledge to generate more context-aware comments. Overall, the results in Table II demonstrate the superior performance and strong generalization capabilities of RAGSum across diverse programming languages. The combination of joint fine-tuning, encoder pretraining, and self-improvement enables RAGSum to effectively model the structural and semantic complexity of code, setting a new state-of-the-art in comment generation.

Answer to RQ₂: On the three given datasets, RAGSum substantially outperforms the considered baselines with respect to all the evaluation metrics.

C. RQ₃: How does each component of RAGSum contribute to its overall performance?

We conduct an ablation study to assess the contribution of key components through four settings: (1) **Only Generator** fine-tuning CodeT5 without relevant code-comment pairs; (2) **RAGSum_{w/o} combined** using retriever and generator independently; (3) **RAGSum_{w/o} pretrained + SR** removing both pre-trained encoder and self-refinement process; (4) **RAGSum_{w/o} SR** excluding self-refinement. Results are shown in Table III. Notably, RAGSum_{w/o} combined exhibits a significant decrease of 13.07% for Java, 15.02% for Python, and 20.99%

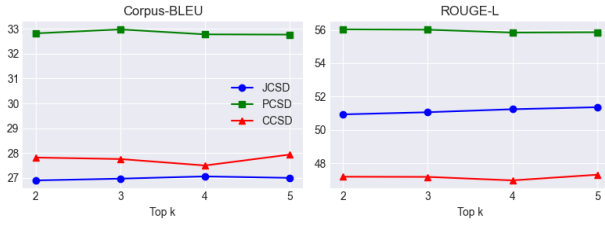


Fig. 4. Top- k Impact Scores

for C in the Corpus-BLEU metric, primarily due to the absence of the joint fine-tuning strategy, which is essential for effectively aligning the retriever and generator components. It can be observed that excluding both encoder pretraining and the self-improvement mechanism consistently degrades the model’s performance across all metrics and programming languages. Further analysis shows that removing the self-refinement mechanism results in a performance degradation across all metrics. For instance, in PCS, Corpus-BLEU falls from 33.0 to 32.57, and in CCS, it drops from 27.95 to 27.76.

For a more in-depth analysis of the effectiveness of top k relevant code comment pairs during fine-tuning, Fig. 4 presents a comparative analysis of k values, ranging from 2 to 5 based on two metrics C-BLEU and ROUGE-L. The optimal performance is achieved with a top- k value of 4 for JCS, while a value of 3 is the preferred choice for both PCS and CCS.

Answer to RQ₃: Each component of RAGSum contributes significantly to its overall performance, enhancing different aspects of the model.

D. Qualitative analysis

We present two examples demonstrating the superior efficiency of our retrieval method compared to existing baselines. This highlights the impact of relevant code comment on the quality of generated comment. Figure 5 shows the example in Java code, RAGSum retrieves comment which is the most relevant to target, provide meaningful context for generation. In this case, the comments are generated by CMR-Sum and JOINTCOM lack the information contained in the code snippet. In Fig 6, CMR-Sum retrieves the same comment with RAGSum, but the generated comment by CMR-Sum fails to align closely with target, because our approach employs a joint modeling mechanism that better integrated retrieval and generation. JOINTCOM’s retriever performance is limited in Fig 6, leading to a generated comment that lacks sufficient information from the given code. In both cases, the output generated by LLama-3.1-8B leverages the relevant code and comments, but the result remains misaligned with the reference. Compared to LLM, the comments produced by LLM often deviate from reference and lack alignment with ground truth. However, the strength of LLMs should not be underestimated. LLMs play an important role in code summarization task. It is necessary to fine-tune LLMs and

```
public boolean registerUser(String username,
    String password, String uri)
    throws DataAccessException {
    if (checkUsernameExists(username)) {return false;}
    storeUser(username, password, uri);
    return true;}

Ground Truth: attempts to register the username , password combination . checks
if username not already exist . returns true if successful , false otherwise .

Retrieved comment:
CMR-Sum: method to register a new user, user will automatically be added to
the default user level new users will be automatically added to the organization
with the id specified in the configuration value default domain id .
JOINTCOM: authenticates the given username , password combination . hash of
password is matched against the hash value stored for password field
RAG with LLama-3.1-8B: authenticates the given username , password combina-
tion . hash of password is matched against the hash value stored for password
field.
RAGSum: attempts to register the uid , key and username combination. returns
true if successful , false otherwise .

Prediction:
CMR-Sum: attempts to register the user with the specified username and password
JOINTCOM: register a user with the database .
RAG with LLama-3.1-8B: Registers a new user with the given username and
password combination.
RAGSum: attempts to register the username and password combination .returns
true if successful , false otherwise .
```

Fig. 5. Example of Java code.

```
def demo(print_times=True, print_grammar=False,
    print_trees=True, trace=2, numparses=5):
    grammar = demo_grammar()
    if print_grammar:
        print(grammar)
    print("sentence")
    tokens = sent.split()
    print(tokens)
    earley = earley_chart_parser(grammar, trace=trace)
    t = time.clock()
    chart = earley.chart_parse(tokens)
    parses = list(chart.parses(grammar.start()))
    t = time.clock() - t
    ...

Ground Truth: a demonstration of the earley parsers .

Retrieved comment:
CMR-Sum: a demonstration of the recursive descent parser .
JOINTCOM: helper function for integer factorization .
RAG with LLama-3.1-8B: check that interpret sents is compatible with legacy
grammars that use a lowercase sem feature .
RAGSum: a demonstration of the recursive descent parser .

Prediction:
CMR-Sum: a demonstration of the grammar .
JOINTCOM: generates a demo chart for the given grammar .
RAG with LLama-3.1-8B : check that earley chart parser is compatible with
legacy grammars that use a lowercase sem feature.
RAGSum: a demonstration of the earley chart parser .
```

Fig. 6. Example of Python code.

combine them with external tools to better adapt to domain-specific task.

E. Threats to Validity

a) *Internal validity:* We used the most popular metrics for evaluating code summarization but it may have some limitations. These metrics may not fully capture semantic equivalence, potentially underestimating the quality. Therefore, it is necessary to evaluate generated summaries from additional perspectives, such as human evaluation [35].

b) *External validity:* Potential threat to validity lies in the variation of results and performance of our approach with different coding styles, programming languages and levels of complexity. To mitigate this, we selected three widely

used datasets with different programming languages, aiming to capture a broad range of code characteristics.

VI. CONCLUSION AND FUTURE WORK

In this paper, we proposed RAGSum for automated code comment generation that effectively leverages the existing joint fine-tuning retriever and generator. Empirical evaluation of benchmark datasets showed that RAGSum significantly improved baselines in code summarization. For future work, we plan to explore more dynamic retrieval mechanisms, investigate the scalability of RAGSum to large-scale codebases, and extend our approach to support multilingual codebases and more diverse programming paradigms.

ACKNOWLEDGMENT

This work is supported by the Swedish Agency for Innovation Systems through the project “Secure: Developing Predictable and Secure IoT for Autonomous Systems” (2023-01899), and by the Key Digital Technologies Joint Undertaking through the project “MATISSE: Model-based engineering of digital twins for early verification and validation of industrial systems” (101140216).

REFERENCES

- [1] R. Xie, W. Ye, J. Sun, and S. Zhang, “Exploiting method names to improve code summarization: A deliberation multi-task learning approach,” in *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. IEEE, 2021, pp. 138–148.
- [2] T. Ahmed and P. Devanbu, “Few-shot training llms for project-specific code-summarization,” in *Proceedings of the 37th IEEE/ACM international conference on automated software engineering*, 2022, pp. 1–5.
- [3] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker, “Towards automatically generating summary comments for java methods,” in *Proceedings of the 25th IEEE/ACM international conference on Automated software engineering*, 2010, pp. 43–52.
- [4] P. W. McBurney and C. McMillan, “Automatic source code summarization of context for java methods,” *IEEE Transactions on Software Engineering*, vol. 42, no. 2, pp. 103–119, 2015.
- [5] E. Wong, J. Yang, and L. Tan, “Autocomment: Mining question and answer sites for automatic comment generation,” in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2013, pp. 562–567.
- [6] E. Wong, T. Liu, and L. Tan, “Clocom: Mining existing source code for automatic comment generation,” in *2015 IEEE 22nd International conference on software analysis, evolution, and reengineering (SANER)*. IEEE, 2015, pp. 380–389.
- [7] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, “Summarizing source code using a neural attention model,” in *54th Annual Meeting of the Association for Computational Linguistics 2016*. Association for Computational Linguistics, 2016, pp. 2073–2083.
- [8] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, “Deep code comment generation,” in *Proceedings of the 26th conference on program comprehension*, 2018, pp. 200–210.
- [9] A. LeClair, S. Jiang, and C. McMillan, “A neural model for generating natural language summaries of program subroutines,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 795–806.
- [10] W. Ye, R. Xie, J. Zhang, T. Hu, X. Wang, and S. Zhang, “Leveraging code generation to improve code retrieval and summarization via dual learning,” in *Proceedings of The Web Conference 2020*, 2020, pp. 2309–2319.
- [11] M. R. Parvez, W. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, “Retrieval augmented code generation and summarization,” in *Findings of the Association for Computational Linguistics: EMNLP 2021*, 2021, pp. 2719–2734.
- [12] J. A. Li, Y. Li, G. Li, X. Hu, X. Xia, and Z. Jin, “Editsum: A retrieve-and-edit framework for source code summarization,” in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 155–166.
- [13] H. Lu and Z. Liu, “Improving retrieval-augmented code comment generation by retrieving for generation,” in *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2024, pp. 350–362.
- [14] L. Li, B. Liang, L. Chen, and X. Zhang, “Cross-modal retrieval-enhanced code summarization based on joint learning for retrieval and generation,” *Information and Software Technology*, vol. 175, p. 107527, 2024.
- [15] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, “Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation,” in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021, pp. 8696–8708.
- [16] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, “The llama 3 herd of models,” *arXiv preprint arXiv:2407.21783*, 2024.
- [17] A. Authors, “When Retriever Meets Generator: A Joint Model for Code Comment Generation.” Figshare, May 2025. [Online]. Available: <https://figshare.com/s/9654cd9dfabe9332b0e3>
- [18] Y. Wan, Z. Zhao, M. Yang, G. Xu, H. Ying, J. Wu, and P. S. Yu, “Improving automatic source code summarization via deep reinforcement learning,” in *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*, 2018, pp. 397–407.
- [19] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, “Codebert: A pre-trained model for programming and natural languages,” *arXiv preprint arXiv:2002.08155*, 2020.
- [20] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu *et al.*, “Graphcodebert: Pre-training code representations with data flow,” *arXiv preprint arXiv:2009.08366*, 2020.
- [21] E. Shi, Y. Wang, L. Du, H. Zhang, S. Han, D. Zhang, and H. Sun, “CAST: Enhancing code summarization with hierarchical splitting and reconstruction of abstract syntax trees,” in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, Eds. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 4053–4062. [Online]. Available: <https://aclanthology.org/2021.emnlp-main.332/>
- [22] J. Zhang, X. Wang, H. Zhang, H. Sun, and X. Liu, “Retrieval-based neural source code summarization,” in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 1385–1397.
- [23] F. Mu, X. Chen, L. Shi, S. Wang, and Q. Wang, “Automatic comment generation via multi-pass deliberation,” in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–12.
- [24] W. Sun, Y. Miao, Y. Li, H. Zhang, C. Fang, Y. Liu, G. Deng, Y. Liu, and Z. Chen, “Source code summarization in the era of large language models,” *arXiv preprint arXiv:2407.07959*, 2024.
- [25] W. Sun, C. Fang, Y. You, Y. Miao, Y. Liu, Y. Li, G. Deng, S. Huang, Y. Chen, Q. Zhang *et al.*, “Automatic code summarization via chatgpt: How far are we?” *arXiv preprint arXiv:2305.12865*, 2023.
- [26] E. Shi, Y. Wang, W. Gu, L. Du, H. Zhang, S. Han, D. Zhang, and H. Sun, “Cocosoda: Effective contrastive learning for code search,” in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 2198–2210.
- [27] S. Liu, Y. Chen, X. Xie, J. Siow, and Y. Liu, “Retrieval-augmented generation for code summarization via hybrid gnn,” *arXiv preprint arXiv:2006.05405*, 2020.
- [28] H. Q. To, N. D. Q. Bui, J. Guo, and T. N. Nguyen, “Better language models of code through self-improvement,” 2023.
- [29] A. V. M. Barone and R. Sennrich, “A parallel corpus of python functions and documentation strings for automated code documentation and code generation,” *arXiv preprint arXiv:1707.02275*, 2017.
- [30] Y. Gao and C. Lyu, “M2ts: Multi-scale multi-modal approach based on transformer for source code summarization,” in *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, 2022, pp. 24–35.
- [31] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “Bleu: a method for automatic evaluation of machine translation,” in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.
- [32] C.-Y. Lin, “Rouge: A package for automatic evaluation of summaries,” in *Text summarization branches out*, 2004, pp. 74–81.
- [33] S. Banerjee and A. Lavie, “Meteor: An automatic metric for mt evaluation with improved correlation with human judgments,” in *Proceedings*

of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization, 2005, pp. 65–72.

- [34] R. Vedantam, C. Lawrence Zitnick, and D. Parikh, “Cider: Consensus-based image description evaluation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 4566–4575.
- [35] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, “Deep code comment generation with hybrid lexical and syntactical information,” *Empirical Software Engineering*, vol. 25, pp. 2179–2217, 2020.