Empirical Software Engineering (2026) 31:84
https://doi.org/10.1007/s10664-025-10800-4

R^{2}ComSync: improving code-comment synchronization with in-context learning and reranking

Zhen Yang[1] · Hongyi Lin[1] · Xiao Yu[2] · Jacky Wai Keung[3] · Shuo Liu[3] · Pak Yuen Patrick Chan[3] · Yicheng Sun[3] · Fengji Zhang[3]

Received: 5 April 2025 / Accepted: 15 December 2025 / Published online: 25 February 2026

Abstract
Code-Comment Synchronization (CCS) aims to synchronize the comments with code changes in an automated fashion, thereby significantly reducing the workload of developers during software maintenance and evolution. While previous studies have proposed various solutions that have shown success, they often exhibit limitations, such as a lack of generalization ability or the need for extensive task-specific learning resources. This motivates us to investigate the potential of Large Language Models (LLMs) in this area. However, a pilot analysis proves that LLMs fall short of State-Of-The-Art (SOTA) CCS approaches because (1) they lack instructive demonstrations for In-Context Learning (ICL) and (2) many correct-prone candidates are not prioritized. To tackle the above challenges, we propose R^{2}ComSync, an ICL-based code-Comment Synchronization approach enhanced with Retrieval and Re-ranking. Specifically, R^{2}ComSync carries corresponding two novelties: (1) Ensemble hybrid retrieval. It equally considers the similarity in both code-comment semantics and change patterns when retrieval, thereby creating ICL prompts with effective examples. (2) Multi-turn re-ranking strategy. We derived three significant rules through large-scale CCS sample analysis. Given the inference results of LLMs, it progressively exploits three re-ranking rules to prioritize relatively correct-prone candidates. We evaluate R^{2}ComSync using five recent LLMs on three CCS datasets covering both Java and Python programming languages, and make comparisons with five SOTA approaches. Extensive experiments demonstrate the superior performance of R^{2}ComSync against other approaches. Moreover, both quantitative and qualitative analyses provide compelling evidence that the comments synchronized by our proposal exhibit significantly higher quality.

Keywords Code-comment synchronization · Large language models · In-context learning · Multi-turn Re-ranking

Communicated by: Foutse Khomh.

Extended author information available on the last page of the article

1 Introduction

Code comments are descriptions in natural language that record code functionalities or implementations, which are crucial to program comprehension and maintenance (Keyes 2002; Pascarella et al. 2019; Sridhara et al. 2010). However, with the increasing frequency of iterations of software products, developers often ignore or forget to synchronize comments when the corresponding code undergoes changes (Ratol and Robillard 2017; Liu et al. 2023). Such inconsistent or outdated comments (i.e., bad comments) hinder the program comprehension among developers and can lead to future bugs (Tan et al. 2007; Wen et al. 2019; Tan et al. 2007; Yao et al. 2024). In order to eliminate or even avoid bad comments, previous studies have proposed various Code-Comment Synchronization (CCS) approaches of three categories, including heuristic-based approaches (e.g., HEBCUP Lin et al. 2021), deep learning-based approaches (e.g., CUP Liu et al. 2020 and HatCUP Zhu et al. 2022), and two-phase-based approaches (e.g., CBS Yang et al. 2023 and Toper Lin et al. 2023). Despite the promising advances, each type of approach has its inherent limitations. For example, heuristic-based approaches try to introduce expert experience but result in a lack of generalizability (Yang et al. 2023; Lin et al. 2023). Deep learning-based approaches necessitate a vast amount of high-quality code-comment co-change historical data for training, which are extremely costly to construct (Ni et al. 2022; Nashid et al. 2023). Two-phase approaches combine the two kinds above and are constrained by their respective capability, making independent performance improvement difficult (Yang et al. 2023).

In recent years, Large Language Models (LLMs) have been applied in various code intelligence scenarios, such as code generation (Jiang et al. 2024; Li et al. 2024), code translation (Yang et al. 2024; Xue et al. 2025a, b), and program repair (Xia et al. 2023; Xue et al. 2024), owing to their powerful generalizability and exemption of task-specific retraining/finetuning during domain adaptation. As such, LLMs can naturally overcome the above limitations of current CCS approaches, but their adaptability in the CCS task has not been fully explored. Nonetheless, we conducted a pilot analysis in Section 3 and found that LLMs fall short of SOTA CCS approaches across diverse Programming Languages (PLs), even with several random examples for In-Context Learning (ICL). Through a careful investigation, we summarize two primary challenges that are imperative for LLMs to address during the adaptation process in the CCS field.

(1) Lacking instructive demonstrations for ICL-based code-comment synchronization The performance of ICL is heavily dependent on the quality of prompts, particularly those demonstrations (i.e., examples) that are crucial for guiding the inference process of LLMs (Liu et al. 2023; Yan et al. 2025). Our pilot analysis (shown in Section 3) found that although randomly selecting examples can yield better results than zero-shot learning, it still falls short of the performance of SOTA approaches. Thus, more effective examples that can provide precise instructions are needed. Previous studies in fields such as code completion (Lu et al. 2022; Zhang et al. 2023), program repair (Nashid et al. 2023), and assertion generation (Nashid et al. 2023) have achieved significant outcomes by retrieving semantically similar examples to compose the effective prompts. However, in the context of CCS, we argue that apart from the code-comment semantic similarity, similar code-comment change patterns should also be considered when retrieval, thereby providing specific and precise editing guidance on old comments.

(2) Given the diverse outputs of LLMs, many correct-prone candidates are not prioritized Although the performance of LLMs in the CCS field is poor, as shown in Section 3, we find that many correct candidates have been generated but not ranked first. This indicates that we should design an effective re-ranking strategy to prioritize correct-prone ones, thereby further boosting the CCS performance of LLMs.

To overcome the challenges above, we propose R^2 ComSync in this paper, an ICL-based code-Comment Synchronization approach enhanced with Retrieval and Re-ranking. (1) Towards curating instructive demonstrations, R^{2}ComSync first employs our proposed Ensemble Hybrid Retrieval (EHR) method to retrieve CCS demonstrations from both aspects of code-comment semantics and change patterns. Specifically, EHR adopts CodeBERT (Feng et al. 2020) to encode old code, old comments, and new code, obtaining code-comment semantics. Meanwhile, it exploits eleven manually crafted expert features proposed by Lin et al. (2023) to represent the code-comment change patterns. After obtaining two demonstration pools derived from the above two aspects, respectively, we then collect the top-(P/2) similar examples from each pool and concatenate them to construct a total of P demonstrations, thereby ensuring the balance between semantic and expert retrieval. After curating a series of CCS demonstrations that are similar in both code-comment semantics and change patterns, R^{2}ComSync incorporates those retrieved demonstrations into a hard prompt, a template we formulate particularly for the ICL-based CCS approach. Subsequently, R^{2}ComSync invokes an LLM (e.g., Llama3Dubey et al. 2024) to conduct the code-comment synchronization under the guidance of the specific prompts. (2) To prioritize correct-prone candidates, we conducted an in-depth analysis of CCS samples (as shown in Section 3), summarizing a series of re-ranking rules based on our observations of the relationship between code and comment co-changes. These rules are used to measure the likelihood of each candidate making mistakes and are categorized into three levels: weak, medium, and severe. According to the error-proneness level (i.e., from weak to severe), we perform three consecutive re-rankings in order. In this way, candidates who violate the higher-level rule will be ranked lower, thereby prioritizing the candidates who are relatively more correct-prone. As an engineering innovation, R^{2}ComSync incorporates previously proven effective thoughts, using them to make domain-specific innovations that improve CCS performance for LLMs.

We evaluated R^{2}ComSync on three CCS datasets: Liu, Panth, and Pai's datasets. The former two are of Java PLs, while the last one was crawled from popular Python projects. Extensive experiments demonstrate that R^{2}ComSync improves the CCS performance of different LLMs (i.e., Llama3-8, Llama3-70B, and GPT-3.5-Turbo) under test by 188.98%–694.08% in terms of Accuracy, 83.65%–337.92% in terms of Recall@5, and 40.50%–331.00% in terms of ESS Ratio across different datasets. As for the comparison between SOTA baselines, R^{2}ComSync with Llama3-70B performs the best overall. With the best setting on Liu's dataset, it outperforms SOTA baselines on average by 31.64% in terms of Accuracy, 13.16% in terms of Recall@5, and 37.98% in terms of ESS Ratio. As for Panth's dataset, R^{2}ComSync with Llama3-70B surpasses SOTA baselines on average by 142.10%, 89.59%, and 82.79% in terms of Accuracy, Recall@5, and ESS Ratio, respectively. Regarding the CCS performance on Python projects, R^{2}ComSync with Llama3-70B still maintains its superiority on average by 166.98%, 145.51%, and 127.23% in terms of each evaluation metric in order on Pai's dataset. Moreover, human evaluation in terms of similarity, naturalness, and sensitivity

further confirms the superiority of R^{2}ComSync in practical usage. Besides, we further conduct a series of ablation experiments to study the effectiveness of our proposed EHR method and MR strategy, extend R^{2}ComSync to Qwen2.5-1.5B and Qwen2.5-Coder-1.5B to examine its applicability on small LLMs, and qualitatively discuss the superiority and limitations of R^{2}ComSync with case studies.

The contributions of this paper can be summarized below:

- We conducted the first systematic comparison between LLMs and SOTA approaches in the field of code-comment synchronization, uncovering the weaknesses of LLMs and proposing improvement directions.
- We proposed R^{2}ComSync, including an Ensemble Hybrid Retrieval (EHR) method and a Multi-turn Re-ranking (MR) strategy. The former harnesses the retrieval superiority from both semantic and expert aspects, while the latter is derived from large-scale sample analysis with high explanability.
- We conduct extensive experiments on R^{2}ComSync both quantitatively and qualitatively. Besides, we compare R^{2}ComSync with five state-of-the-art CCS approaches, showing its powerful capability.
- We open source the replication package of R^{2}ComSync for facilitating its application and future research, which is available at YDS0823/ComSync (2025).

2 Background and Related Work

In this section, we introduce the background and related work concerning code-comment synchronization, code-comment inconsistency detection, in-context learning for software engineering, and post-processing of large language models.

2.1 Code-Comment Synchronization

Code-Comment Synchronization (CCS) is an emerging research field, which aims to synchronize comments with code changes (Yang et al. 2023). Formally, for each group of before- and after-change code snippets, namely c and $c^{'}$, along with their corresponding before- and after-change comments, namely x and y, CCS approaches are intended to devise a function that approximates f, i.e., $y = f(c, c^{'}, x)$, where c, $c^{'}$, x, and y are referred to as old code, new code, old comment, and new comment, respectively. In this paper, a CCS target is composed of a group of c, $c^{'}$, and x, whose y, namely reference comment, is unknown during the inference, while a CCS demonstration, as an example of a CCS target, consists of a group of c, $c^{'}$, x, and y. In literature, many studies have conducted various attempts to tackle this problem.

Liu et al. (2020, 2023) and Panthaplackel et al. (2020), for the first time, defined the CCS problem and proposed their own solutions concurrently. The former designed an LSTM-based neural network of the standard encoder-decoder paradigm, while the latter adopted a GRU-based component during their model construction and generated sequences of edits to modify old comments. Subsequently, Lin et al. (2021) proposed a heuristic-based approach to update tokens/sub-tokens in old comments according to the code changes. Later, Zhu et al. (2022) incorporated code structure changes into their neural network and achieved

further improvement in model performance. Two latest works were proposed by Yang et al. (2023) and Lin et al. (2023), respectively. Both of them put forward a two-phase approach, i.e., identifying categories of samples before synchronizing their comments via heuristic-based or deep learning-based approaches.

However, the approaches above either lack generalizability, require tremendous task-specific training data, or cannot independently improve performance. In contrast, this paper puts forward an ICL-based CCS approach, which leverages the generalizability of LLMs without task-specific fine-tuning/retraining to make reasonable inferences for the CCS task. Consequently, the limitations of current approaches can be avoided.

2.2 Code-Comment Inconsistency Detection

Code-Comment Inconsistency Detection (CCID) is a closely related code task to CCS mentioned above, which aims to detect whether a code snippet can be accurately described by its associated comment (Tan et al. 2007). As such, CCID is a preceding task before synchronizing comments to be consistent with their code, which has been widely studied in the past several decades (Tan et al. 2007; Panthaplackel et al. 2021; Wen et al. 2019; Rabbi and Siddik 2020; Xu et al. 2024; Zhang et al. 2024; Zhang 2024). Tan et al. (2007) first formulated the CCID task and proposed iComment with the combination of machine learning and program analysis techniques. Subsequently, Rabbi and Siddik (2020) designed a siamese recurrent network to analyze code and comments respectively, thereby computing their similarity for inconsistency detection purposes. Afterwards, to overcome characterization noise and labeling errors in CCS datasets, Xu et al. (2024) put forward MCCL that first removes noise from CCS datasets with confidence learning and then detects inconsistent code comments, thereby enhancing the model's learning ability. Very recently, CCID tools have been incorporated with LLM techniques. For example, Zhang et al. (2024); Zhang (2024) combined LLM-driven techniques and program analysis to identify inconsistencies in code comments, where the former was used for design constraints extraction while the latter was applied for verifying the implementation of the above extracted constraints. Their proposed approach, namely RustC^4, has been successfully examined on 12 large-scale real-world Rust projects.

2.3 In-Context Learning for Software Engineering

In-Context Learning (ICL) is an emerging learning paradigm that does not conduct parameter updates and performs the inference on LLMs directly according to the given demonstrations (Dong et al. 2024). Inspired by its extensive application and promising results in the NLP field (Wei et al. 2022; Ram et al. 2023; Chen et al. 2023; Min et al. 2022), researchers adapt ICL techniques to code-related tasks. For example, Li et al. (2024) adopted ICL with coding preliminaries to study automatic code generation. Nashid et al. (2023) explored the influence of demonstration retrieval on ICL-based assertion generation and program repair. Zhang et al. (2023) proposed an iterative retrieval-generation framework to enhance the ICL capability of LLMs in the field of code completion.

Nonetheless, current studies using ICL did not consider task-specific adaptation during their demonstration retrieval and neglected the importance of post-processing for LLM's outputs. Therefore, we propose an ensemble hybrid retrieval method and multi-turn re-ranking strategy to augment the ICL ability of LLMs toward the CCS field. The former,

tailoring to the CCS task, improves the previous retrieval methods, while the latter is used for optimizing model outputs. Both methods benefit LLMs in generating new comments of high quality and significantly improve their performance.

2.4 Large Language Models with Post-Processing

Large Language Models (LLMs) have demonstrated significant success in various natural language processing tasks. The diversity in their outputs enables the use of post-processing techniques to ensure the relevance and accuracy of the answers provided by the models.

Chen et al. (2022) utilize the same pretrained language model to automatically generate test cases for code samples, reducing human effort and increasing the coverage of testing scenarios. Subsequently, CodeT executes the generated test cases on code samples and employs a dual execution protocol, which considers both the consistency of the output with the generated test cases and the consistency of the output with other code samples, selecting the most suitable solution from multiple samples generated by the pretrained model. Zhang et al. (2023). introduce a collaborative framework inspired by real-world programming practices. First, an encoder model generates a series of candidate programs. Then, a reviewer model reranks these candidates by analyzing their alignment with the input instructions. The reviewer provides a complementary perspective to the encoder's likelihood-based approach. Jain et al. (2022) enhance LLMs through post-processing steps based on program analysis and synthesis techniques. These techniques understand program syntax and semantics, leverage user feedback, and improve over time with usage. Yan et al. (2024) employ a constrained regression method to minimally adjust relevance scores generated by LLMs, aligning them with pairwise ranking preferences. The adjustments prioritize maintaining original relevance estimates while ensuring consistency with ranking predictions.

Our method requires no additional model training. We cleverly reorder generated samples based on three reranking rules derived from analyzed datasets, which have proven to significantly improve model performance

3 Motivation

Although Fan et al. (2024) have conducted an empirical study on diverse LLMs on code-comment synchronization, they did not make comparisons with the State-Of-The-Art (SOTA) methods in this field. Thus, it is unknown whether LLMs are effective tools in handling the code-comment synchronization task in practice compared with existing approaches. As such, we propose the first Research Question (RQ), namely **RQ1: How do LLMs perform against the state-of-the-art CCS approaches?**

Experimental design To address RQ1, we evaluate three LLMs, i.e., Llama3-8B, Llama3-70B, and GPT-3.5-Turbo, because they are the SOTA LLMs at the time we conduct experiments and have been widely investigated in other code comprehension studies (Yu et al. 2024; Xue et al. 2024; Sun et al. 2025; Fan et al. 2024). Besides, we measure their effectiveness according to the evaluation metrics mentioned in Section 5.3 on all three CCS datasets introduced in Section 5.1. Except for Toper and HatCUP, all SOTA approaches and studied LLMs are experimented on the same testing set across all three datasets, where the former

are rerun with their publicly available replication packages. Regarding Toper and HatCUP, we include them in Liu's dataset solely for comparison based on the reported experimental results in their papers. The reason is that some of their key components are not open-source. Therefore, we cannot rerun these two baselines to obtain the corresponding results on Panth and Pai's datasets. As for the implementation of LLMs, we follow the procedures in Section 5.4. All experiments are repeated five times. Following the above setting, we try to make a fair and reliable comparison between SOTA approaches and the studied LLMs.

Results Table 1 presents a comparative analysis between LLMs and the SOTA approaches. As for the zero-shot learning setting, only Llama3-70B obtains a slight superiority by 3.85% against the best-performing SOTA approach (i.e., CBS) in terms of ESS ratio on Pai's dataset. For most other occasions, the results clearly indicate that LLMs struggle to compete with existing baselines. For example, the best-performing SOTA approach outperforms LLMs of this setting by 49.00%–463.67% in terms of Accuracy, by 26.81%–181.02% in terms of Recall@5, and by 32.12%–383.10% in terms of ESS ratio on Liu's dataset. As for Panth's dataset, the best-performing SOTA approach surpasses LLMs by 75.11%–893.28%, 53.43%–535.25%, and 89.99%–547.22% in terms of each metric in order. Regarding Pai's dataset, SOTA approaches still keep their superiority by 15.20%–453.15% in terms of Accuracy and by 23.76%–183.86% in terms of Recall@5. When we turn to the two-random-shots learning setting, LLMs' performance indeed improves a lot compared with the zero-shot learning setting. But it still falls short of the performance of SOTA baselines on most occasions, as shown in Table 1. This phenomenon illustrates that the CCS performance improvement of LLMs is limited without meaningful and relevant ICL examples. We need a retrieval module that provides effective samples for the LLMs to learn from context, thereby improving the accuracy of generated comments.

Additionally, we have observed that many of the candidates generated by the LLMs are correct but not ranked first. This explains why the performance on Recall@5 is significantly higher than on Accuracy, both in zero-shot and two-shot learning settings. Besides, as the performance of LLMs improves, more correct answers appear within their candidate items, potentially lifting the upper bound of LLMs' CCS performance and making it possible to outperform SOTA baselines. Therefore, a re-ranking strategy is needed to prioritize the correct answers.

Along this line of thought, the first and second authors conducted an analysis to find out some features of correct synchronizations. The analysis began from the authors' prior developing experience and small-scale observation of the code-comment co-change behaviours. Subsequently, they verified these experiences and intuitive observations by large-scale analysis on Liu's training set (a total of 80,591 samples). First, the authors found that when a function name is changed in the code and mentioned in the corresponding old comment, 86.9% of the training samples also change the function name in the new comment. For example, Fig. 1(a) conforms to this finding exactly, where its function name in the old code is "getConversationPanel" while changing to "getLastIncomingMsgTimestamp" in the new code. Since function names normally convey the core intent of code snippets, it can be seen that the elements (e.g., "Conversion","Panel", "Last", and "Msg") involved in the function names are also changed (e.g., "conversion","panel", "last", and "message") in comments. Next, the authors found that the proportion of sub-tokens appearing in the new comments

Table 1 Performances of Baselines and LLMs

Dataset	Approach	Accuracy%↑	Recall@5%↑	ESS Ratio%↑
Liu's Dataset	CUP	$21.63_{\pm 1.672}$	$31.79_{\pm 1.478}$	$28.64_{\pm 1.512}$
	HEBCUP	$26.68_{\pm 0.226}$	$27.67_{\pm 0.226}$	$27.72_{\pm 0.025}$
	CBS	$28.53_{\pm 0.270}$	$35.76_{\pm 1.338}$	$31.61_{\pm 0.418}$
	HatCUP	24.30	35.20	31.33
	Toper	**30.10**	33.04	**34.59**
	LLM with Zero-shot Learning			
	GPT3.5-turbo	$15.05_{\pm 0.261}$	$19.43_{\pm 0.304}$	$18.89_{\pm 0.304}$
	Llama3-8B	$5.34_{\pm 0.462}$	$12.59_{\pm 0.462}$	$7.16_{\pm 0.558}$
	Llama3-70B	$20.20_{\pm 0.897}$	$27.90_{\pm 0.924}$	$26.18_{\pm 0.924}$
	LLM with Two-Random-shots Learning			
	GPT3.5-turbo	$15.93_{\pm 0.261}$	$25.95_{\pm 0.304}$	$22.01_{\pm 0.608}$
	Llama3-8B	$15.13_{\pm 1.049}$	$24.18_{\pm 0.666}$	$19.75_{\pm 0.840}$
	Llama3-70B	$20.92_{\pm 0.769}$	$27.63_{\pm 0.256}$	$29.62_{\pm 1.675}$
Panth's Dataset	CUP	$7.56_{\pm 0.981}$	$17.34_{\pm 0.494}$	$21.20_{\pm 1.481}$
	HEBCUP	$8.90_{\pm 0.109}$	$9.24_{\pm 0.000}$	$19.54_{\pm 0.323}$
	CBS	$11.82_{\pm 0.657}$	$17.66_{\pm 0.420}$	$25.63_{\pm 0.832}$
	LLM with Zero-shot Learning			
	GPT3.5-turbo	$1.59_{\pm 0.495}$	$3.31_{\pm 0.495}$	$3.96_{\pm 0.495}$
	Llama3-8B	$1.19_{\pm 0.374}$	$2.78_{\pm 0.857}$	$6.35_{\pm 0.561}$
	Llama3-70B	$6.75_{\pm 0.857}$	$11.51_{\pm 0.561}$	$13.49_{\pm 0.000}$
	LLM with Two-Random-shots Learning			
	GPT3.5-turbo	$9.66_{\pm 0.495}$	$13.62_{\pm 0.495}$	$15.61_{\pm 0.495}$
	Llama3-8B	$5.16_{\pm 0.324}$	$10.45_{\pm 0.748}$	$12.83_{\pm 1.042}$
	Llama3-70B	**$15.61_{\pm 1.349}$**	**$21.69_{\pm 1.309}$**	**$27.38_{\pm 0.648}$**
Pai's Dataset	CUP	$3.93_{\pm 0.434}$	$9.33_{\pm 1.545}$	$8.60_{\pm 1.322}$
	HEBCUP	$3.00_{\pm 0.000}$	$3.00_{\pm 0.000}$	$6.53_{\pm 0.186}$
	CBS	**$6.14_{\pm 0.506}$**	**$10.73_{\pm 1.254}$**	$12.2_{\pm 1.304}$
	LLM with Zero-shot Learning			
	GPT3.5-turbo	$5.33_{\pm 0.272}$	$8.67_{\pm 0.272}$	$11.33_{\pm 0.272}$
	Llama3-8B	$1.11_{\pm 0.314}$	$3.78_{\pm 0.416}$	$9.11_{\pm 1.030}$
	Llama3-70B	$3.44_{\pm 0.629}$	$6.33_{\pm 0.000}$	$12.67_{\pm 0.272}$
	LLM with Two-Random-shots Learning			
	GPT3.5-turbo	$5.33_{\pm 0.272}$	$9.00_{\pm 0.272}$	$10.29_{\pm 0.272}$
	Llama3-8B	$3.00_{\pm 0.720}$	$6.78_{\pm 0.567}$	$10.56_{\pm 0.416}$
	Llama3-70B	$5.44_{\pm 0.416}$	$8.78_{\pm 0.157}$	**$13.56_{\pm 1.370}$**

Note: The results are presented in the form of $mean_{\pm std}$, where *mean* is the average result of 5 experiments, and *std* is the corresponding standard deviation. The best-performing results are highlighted in **bold**. For LLMs outperforming all SOTA approaches in terms of certain metrics, the results are highlighted in a light cyan background. "↑" denotes that the higher, the better. The tables in the follow-up sections all follow this convention

but not in the old comments mostly accounts for 0 to 0.4 of the new comments' sub-tokens. This phenomenon occurs as frequently as 92.07% of the total analyzed CCS samples. The detailed analysis can be seen in Figs. 2(a), and 1(b) is a typical example that satisfies the

Project: jitsi/jitsi Commit ID: 035465c7a871c5a27fe681adaadebdf96a86b7e4	ID: JetBrains_jdk8u_jaxp-34-6323

- public ChatConversationPanel getConversationPanel() {
+ public Date getLastIncomingMsgTimestamp() {
- return conversationPanel;
+ return this.conversationPanel.getLastIncomingMsgTimestamp();
}
	- public Hashtable getBuiltInTypes() {
	+ public Map<String, DatatypeValidator> getBuiltInTypes() {
	- Hashtable toReturn = (Hashtable) fBuiltInTypes.clone();
	+ final HashMap<String, DatatypeValidator> toReturn = new HashMap<>(fBuiltInTypes);
	- Enumeration xml11Keys = fXml11BuiltInTypes.keys();
	- while (xml11Keys.hasMoreElements()) {
	- Object key = xml11Keys.nextElement();
	- toReturn.put(key, fXml11BuiltInTypes.get(key));
	- }
	+ toReturn.putAll(xml11BuiltInTypes);
	return toReturn;
	}

Old Comment: Returns the panel that contains the conversation.	**Old Comment:** a ~~hashtable~~ which contains all datatypes.
Reference Comment: Returns the time of the last received message.	**Reference Comment:** a map which contains all datatypes.
(a)	(b)

Fig. 1 CCS Samples in Liu's Training Set And Panth's Traning Set

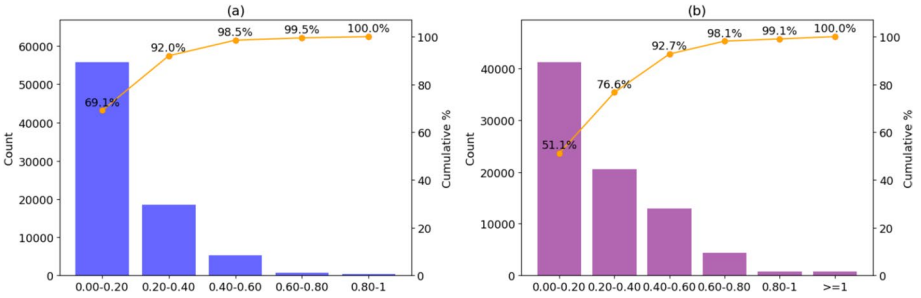

Fig. 2 Sample Analysis of Liu's Training Set

above finding. Its new comment only has one sub-token (i.e., "map") that does not appear in its old comment, accounting for 0.14(1/7) of sub-tokens in its new comments. Finally, the authors found that the ratio of the edit distance between the new and old comments to the number of sub-tokens in the corresponding old comments mostly falls within the range of 0 to 0.6, with a sample proportion of 92.72%. For example, the edit distance between old and new comments in Fig. 1(b) is 1 (i.e., edit from "hashtable" to "map"), accounting for 0.14(1/7) of the sub-tokens in its old comment. The detailed analysis can be seen in Fig. 2(b).

These three findings are derived from a significant amount of the CCS samples in Liu's training set, indicating that we can leverage these findings as our re-ranking rules to further filter the candidate samples generated by the LLMs, making the synchronization more in line with user expectations. As such, the main idea of our re-ranking strategy is established, with the detailed methodology illustrated in Section 4.3. It should be noted that we did not verify these rules in other datasets (e.g., Panth and Pai's) because we plan to conduct experiments (see Section 6) to examine their generalities across other datasets or programming languages, thereby demonstrating their practical value.

RQ1: LLMs struggle to match state-of-the-art baselines in the code-comment synchronization task, but retrieving relevant demonstrations and re-ranking candidate new comments potentially enhance their CCS performance.

4 Methodology

Based on the above analysis in RQ1, we propose R^{2}ComSync with exclusively designed retrieval and re-ranking methods. The overview of R^{2}ComSync is illustrated in Fig. 3, which totally consists of four phases: ❶ demonstration retrieval, where R^{2}ComSync employs the EHR method to retrieve similar CCS demonstrations for each CCS target from the demonstration pool (i.e., the training set). The EHR method takes into account both the similarities of code-comment semantics and change patterns, where the former is extracted via CodeBERT (Feng et al. 2020), while the latter is derived from eleven CCS expert features manually constructed by Lin et al. (2023); ❷ prompt construction aims to create prompts with retrieved demonstrations based on our exclusively designed template; ❸ model invocation proceeds on LLMs (e.g., Llama3), which takes prompts as inputs for ICL and generates a series of candidate new comments; and ❹ re-ranking phase exploits our proposed multi-turn re-ranking strategy to prioritize correct-prone candidates, from which we export the final outputs for evaluation. The following subsections elaborate on each phase in detail.

4.1 Demonstration Retrieval

The demonstration retrieval phase is carried out with the EHR method, retrieving a total of P examples from two demonstration pools, including both code-comment semantic and change-pattern aspects. To achieve this, we design two steps, namely (1) CodeBERT retrieval and (2) Expert retrieval, taking into account both aspects accordingly. To balance the weights of the two retrievals, we take the top-(P/2) similar examples from each demonstration pool and concatenate these examples as the retrieval results. Detailed procedures of both CodeBERT retrieval and Expert retrieval are elaborated below.

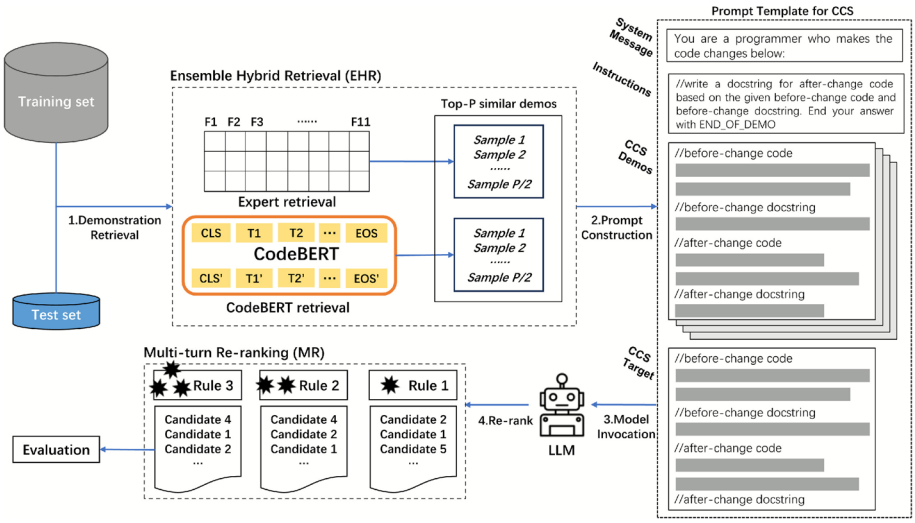

Fig. 3 The Overview of R^{2}ComSync

4.1.1 CodeBERT Retrieval

In the community, code-related semantic feature extraction mainly relies on pre-trained LLMs for their powerful representation capability (Ni et al. 2022; Inala et al. 2022; Zhang et al. 2022). CodeBERT, as a prominent LLM for code representation, has demonstrated its effectiveness across a range of software engineering tasks (Yang et al. 2023; Nashid et al. 2023; Lu et al. 2026), making it a good option for extracting code-comment semantic features in this paper. To be specific, we encode the old code, old comment, and new code of each sample by CodeBERT separately, owing to its encoding length limitation. Following the workaround of previous studies (Ni et al. 2022; Kenton and Toutanova 2019), we add the token "[CLS]" and the token "[EOS]" at the beginning and ending position of each input (i.e., the old code, old comment, and new code), and gather word embedding of token "[CLS]" as each of their contextual-aware representation. Afterwards, we sum the embeddings of each input and refer to the result as the code-comment semantic features. Following the above method, we vectorize all samples from the training set as a semantic demonstration pool. When retrieving demonstrations, we compute cosine similarity between the above samples and those in the test set in pairs, thereby finding the top-(P/2) most semantically similar examples for each testing sample. A more explicit illustration is shown in Fig. 4.

4.1.2 Expert Retrieval

Here, we make use of manually crafted expert features proposed by Lin et al. (2023) for representing code-comment change patterns owing to their successful application in the CCS classification task (Lin et al. 2023). These features (as shown in Table 2) are derived from large-scale empirical analysis based on the complexity of code changes, the extent of code changes in old comments, and the contextual information of code changes. However, it should be noted that the derivation algorithm of one of the features (i.e., TE under the context dimension, specifically the **T**ype of the **E**xpression where the changed token locates) is not open source, which makes cross-dataset extension difficult. Hence, we drop the feature TE in the follow-up experiments and retain eleven features in total, depicting code-comment change patterns from three main dimensions, including complexity, involvement, and context. Similarly, we perform vectorized representation based on the above features on the

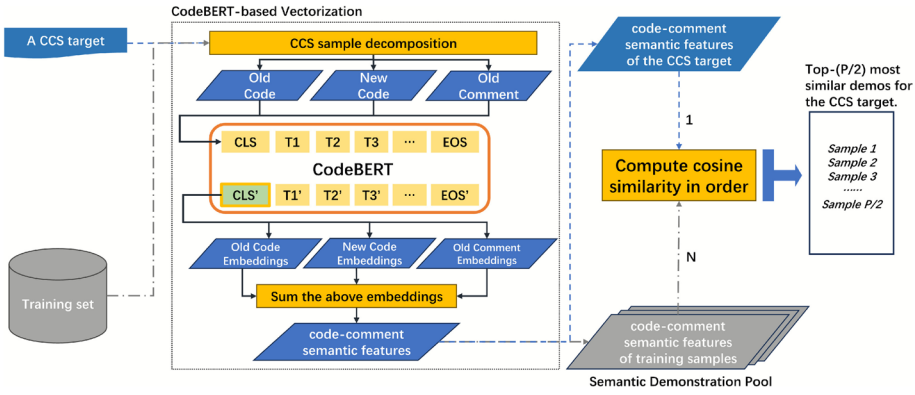

Fig. 4 The Illustration of CoderBERT Retrieval

Table 2 Expert Features that Imply Code-Comment Change Patterns

Dim	Features	Definition
Complexity	NMS	Number of Modified Sub-tokens.
	NMT	Number of Modified Tokens.
	NML	Number of Modified Lines.
	NMC	Number of Modified Chunks.
	NNTRP	Number of Non-redundant Token Replacement Pairs.
	NNSRP	Number of Non-redundant Sub-token Replacement Pairs.
Involvement	NTOD	Number of Tokens which Occur in the old comment but Disappear after the code change.
	NSOD	Number of Sub-tokens which Occur in the old comment but Disappear after the code change.
Context	TS_1	the Type of the Statement where the FIRST changed token locates.
	TS_2	the Type of the Statement where the SECOND changed token locates.
	TS_3	the Type of the Statement where the THIRD changed token locates.

training sets to construct the demonstration pool. Then, we calculate the cosine similarity to find the top-(P/2) similar examples from the above demonstration pool for each CCS target in the testing set.

4.2 Prompt Construction and Model Invocation

As shown in the right-hand side of Fig. 3, we design an exclusive prompt template for R^2 ComSync in phase 2, which comprises a system message to designate roles, an instruction that delivers commands, and a series of placeholders (gray boxes in the prompt template) to be filled by CCS demonstrations and a CCS target. Each demonstration is separated by a delimiter "END_OF_DEMO". Towards the few-shot learning setup, given each CCS target in the testing set, we fill its corresponding retrieved demonstrations and itself in the template, while for the zero-shot learning setup, demonstrations will be ignored. A specific prompt example containing two retrieved demonstrations is listed in our released repository (YDS0823/ComSync 2025) for reference. Subsequently, in phase 3, prompts are fed into a plugged LLM (here, we use Llama3 Dubey et al. 2024 and GPT-3.5 series) to execute the code-comment synchronization. We select the 70B parameter version of LlaMa3 as the main model for this experiment and also replicate our experiments on other LLMs, such as LlaMa3-8B and GPT-3.5-Turbo, to verify the results.

4.3 Multi-Turn Re-ranking

For each CCS target, given a series of candidate new comments generated from LLMs, we conduct the Multi-turn Re-ranking (MR) strategy to prioritize correct-prone ones. In detail, we design three heuristic rules, ranging from weak to severe, based on our pilot analysis in Section 3. We execute the above rules from weak to severe for each candidate and place them in the last position if they violate the rules. Since the candidate order can

be changed after each execution of one of the above rules, the worst candidate that violates the most rules will be ranked last. In contrast, the relatively correct-prone candidates can be prioritized passively. If multiple candidates violate the same rule simultaneously, they will be ranked last together with their original relative order. A MR strategy example and its illustration are shown in Fig. 10 of Section 7.2. Below, we introduce the specific mechanism of each heuristic rule.

- **Rule 1 (Weak):** If the function name is changed and the changed contents can be found in the old comment, the corresponding changes should also appear in the candidate new comment as well. **Explanation:** Typically, a function name contains the main idea of a code snippet (Gao et al. 2019), and comment updates should be sensitive to its changes. This phenomenon accounts for 86.9% of the CCS samples as testified in Section 3. To this end, once the function name has been changed, but the corresponding contents in the old comment were not updated, the candidate new comment is likely to be incorrect. Since this rule only covers instances with function name changes and normally concerns mistakes of only several sub-tokens, we label it as a weak level. A typical example can be found in Fig. 10 of Section 7.2, which is a CCS sample in Liu's dataset. When Rule 1 is executed as the first rule, the third candidate (i.e., "Is the counter valid for usage? ") is ranked in the last position, because the token "valid" in the old comment is not replaced by "active" as the code change occurs in their function names. Since other candidates do not violate Rule 1, the candidates' order after executing Rule 1 is: candidate 1, 2, 4, and 3.
- **Rule 2 (Medium):** The number of sub-tokens in a candidate new comment but not in the corresponding old comment (N_{diff}) should not exceed a predefined threshold (σ) relative to the number of sub-tokens in the candidate new comment ($N_{cm'_s}$). Formally, Rule 2 is defined as below:

$$N_{diff}/N_{cm'_s} < \sigma, \tag{1}$$

- **Explanation:** As we analyzed in Section 3, comment updating normally occurs with limited ranges. For example, new comments of 92.07% of CCS samples produce less than 40% of new sub-tokens against old comments. Thus, constraining the ratio of such sub-tokens is reasonable. Since this rule only considers differences in sub-token values between old comments and candidate new comments, we label it as medium level. Figure 10 of Section 7.2 can also be a typical example to illustrate Rule 2. As can be seen, the value of $N_{diff}/N_{cm'_s}$ for candidates 1-4 are 0.2(1/5)<0.35[1], 0.14(1/7)<0.35, 0(0/7)<0.35, and 0.56(5/9)>0.35. Hence, candidate 4 violates Rule 2 here and should be ranked last based on the order in the previous round. As such, the candidates' order after executing Rule 2 becomes: candidate 1, 2, 3, 4.

- **Rule 3 (Severe):** The ratio between the comment edit distance (ED_{cm}) and the number of sub-tokens in the corresponding old comment (N_{cm_s}) should not exceed the predefined threshold ϵ, where ED_{cm} is computed by the word-level edit distance (Navarro 2001) between old comments and candidate new comments. Formally, Rule 3 is defined as below:

[1] Rule 2 (σ) is set to 0.35 for CCS samples of Liu's dataset as mentioned in Section 5.4.

$$ED_{cm}/N_{cm_s} < \epsilon, \tag{2}$$

- **Explanation:** As revealed in Section 3, if the threshold of Rule 3 is 0.6, 92.72% of CCS samples in the training set are satisfied, illustrating that we should impose restrictions on edit distance between old comments and candidates. Because the randomness during the decoding process of LLMs may potentially cause the generated sub-tokens to be out of order (Xu et al. 2022) or irrelevant to the CCS target, leading to a high value on Rule 3. Since the computation of edit distance considers differences in both the orders and values of sub-tokens, we label this rule as a severe level. Similarly, we continue taking the example in Fig. 10 for illustration. The value of ED_{cm}/N_{cm_s} for candidates 1-4 are 0.429(3/7)>0.25[2], 0.14(1/7)<0.25, 0(0/7)<0.25, and 0.714(5/7)>0.25. Hence, candidates 1 and 4 both violate Rule 3 here and should be ranked last again, while keeping their original relative orders. As such, the candidates' order after executing Rule 3 becomes: candidate 2, 3, 1, 4, which is the final order (With MR Strategy) shown in Fig. 10.

5 Experiment Preparation

This section presents the dataset, baselines for comparison, evaluation metrics, and implementation details of LLMs and R^{2}ComSync.

5.1 Dataset

For experiments, we use two Java datasets and a Python dataset to assess code-comment synchronization, thereby examining the effectiveness of each approach comprehensively. The first dataset is extracted from 1,496 popular Java code repositories hosted on GitHub (GitHub 2023), which has been widely used in this field (Liu et al. 2020; Lin et al. 2021; Zhu et al. 2022; Yang et al. 2023; Lin et al. 2023) and was successively polished by Liu et al. (2020) and Lin et al. (2021). The dataset contains a total of 80,591 training samples, 8,827 validation samples, and 9,204 testing samples. As it was first used by Liu et al. (2020) for code-comment synchronization, we refer to it as Liu's dataset in this work. The second dataset, extracted by Panthaplackel et al. (2020) from popular open-source Java projects, contains a total of 5,791 training samples, 712 validation samples, and 736 testing samples. To distinguish from Liu's dataset, we refer to it as Panth's dataset. Furthermore, to investigate whether our method can be extended to other programming languages, we introduce a third dataset crawled from Python projects. This dataset is adapted from the CoDocBench proposed by Pai et al. (2025), which originally contains 4,573 Python samples. Since the original dataset includes some samples with excessively long comments, which make existing SOTA approaches (e.g., CUP) hard to converge during training, we remove samples with comments exceeding the median length. From the remaining data, we randomly select 300 samples as the testing set and retain 3,013 samples as the training set. Consistent with the naming convention of the previous datasets, we refer to this dataset as Pai's dataset.

For each of the datasets above, all samples consist of two code-comment pairs of the old and new versions for CCS purposes. Table 3 presents their detailed characteristics, where

[2] Rule 3 (ϵ) is set to 0.25 for CCS samples of Liu's dataset as mentioned in Section 5.4.

Table 3 Statistics of Datasets

Dataset		Type	Length			NCS		
(PL)			Mean	Std Dev	Median	Mean	Std Dev	Median
Liu's Dataset (Java)	Train	Old Code	91.22	84.42	60.0	34.73	57.49	11.0
		New Code	91.26	84.98	58.0			
		Old Comment	14.43	7.34	13.0	3.81	3.01	3.0
		New Comment	14.47	7.63	13.0			
	Validation	Old Code	89.60	82.61	59.0	32.73	54.66	12.0
		New Code	88.77	82.66	57.0			
		Old Comment	15.16	7.57	14.0	3.83	3.03	3.0
		New Comment	15.21	7.83	14.0			
	Test	Old Code	97.61	86.92	64.0	36.85	63.73	12.0
		New Code	97.44	87.94	63.0			
		Old Comment	15.37	7.52	14.0	3.97	3.11	3.0
		New Comment	15.29	7.77	14.0			
Panth's Dataset (Java)	Train	Old Code	84.19	124.18	45.0	43.31	109.2	11.0
		New Code	88.67	125.44	48.0			
		Old Comment	10.33	9.18	8.0	2.42	3.37	1.0
		New Comment	11.30	9.61	9.0			
	Validation	Old Code	88.13	121.75	47.0	39.19	77.44	12.0
		New Code	86.63	119.36	51.0			
		Old Comment	10.81	9.15	8.0	2.60	4.24	1.0
		New Comment	11.64	9.65	9.0			
	Test	Old Code	97.73	158.3	48.0	48.4	102.10	14.0
		New Code	97.16	147.35	53.0			
		Old Comment	10.76	8.30	8.0	2.48	3.38	1.0
		New Comment	11.40	8.84	9.0			
Pai's Dataset (Python)	Train	Old Code	156.05	183.33	97.00	5.31	5.97	4.00
		New Code	166.38	188.29	108.00			
		Old Comment	43.89	32.91	37.00	2.82	2.83	2.0
		New Comment	56.80	50.08	45.0			
	Test	Old Code	165.70	184.17	112.0	5.72	6.64	4.0
		New Code	171.88	192.27	112.0			
		Old Comment	49.58	34.18	44.50	2.59	2.61	1.0
		New Comment	60.52	48.22	51.0			

*Note:***Std Dev** stands for standard deviation, which reflects the degree of data dispersion

both their code and comments are tokenized into sequences and further decomposed into sub-tokens based on CamelCase and snake_case naming conventions. The metric Length indicates the count of sub-tokens in each sequence, while the Number of Changed Sub-tokens (NCS) quantifies the modified sub-tokens between the old and new versions of code (or comments). To compromise the limited computational resources while ensuring the validity of our experiments, we sample 368 data points from the testing set of Liu's dataset with a 95% confidence level for the performance evaluation. In contrast, the other two datasets are kept intact for experiments. The training sets of all three datasets are utilized as retrieval databases.

5.2 Baselines

We comprehensively compare the performance of R^2ComSync with the five state-of-the-art approaches, namely CUP (Liu et al. 2020), HEBCUP (Lin et al. 2021), HatCUP (Zhu et al. 2022), CBS (Yang et al. 2023), and Toper (Lin et al. 2023). Their detailed information is listed below.

(1) **CUP** is an LSTM-based encoder-decoder structured neural network, constructed by layers of embedding, co-attention, modeling, and pointer generator (See et al. 2017). By effectively modeling the connections between comments and code changes, CUP has obtained promising results in the CCS field.
(2) **HEBCUP** is the first heuristic-based CCS approach, leveraging a series of fine-designed heuristic rules to conduct token/sub-token level updates on old comments. However, it is only useful for code-indicative samples, whose changed tokens/sub-tokens in comments can also be found in corresponding code changes.
(3) **HatCUP** takes code structure change information into account. Combining data flow dependencies and code change graphs, HatCUP introduces a structure-guided attention mechanism to analyze hybrid information. Besides, it generates edit actions for old comments, instead of complete new comments, leading to its updates being more accurate.
(4) **CBS** is a two-phase approach in the CCS field, which first predicts the proneness of each sample, i.e., heuristic-prone or non-heuristic-prone, then allocates samples to CUP or HEBCUP for code-comment synchronization according to their classification results.
(5) **Toper** is another two-phase approach concurrently proposed with CBS. Inside Toper, an Abstract Syntax Tree (AST) path-based comment updater is proposed to deal with non-code-indicative samples, while code-indicative samples are allocated to HEBCUP to handle. The whole mechanism is also operated via the process of classifying samples first, followed by synchronizing.

5.3 Evaluation Metrics

Following the previous studies in the CCS field (Liu et al. 2020; Lin et al. 2021; Zhu et al. 2022; Yang et al. 2023; Lin et al. 2023), we adopt Accuracy, Recall@5, and ESS Ratio to evaluate the performance of each approach.

Accuracy refers to the ratio of CCS targets where *correct synchronization* can be generated on the first attempt among all examined cases. Particularly, *correct synchronization* denotes that the generated new comments are exactly identical to their corresponding reference comments.

Recall@5 is a similar metric to Accuracy, but allows CCS approaches to carry out five attempts for each case examined. If any one of its attempts reaches the *correct synchronization*, it considers the approach can successfully handle this case.

ESS ratio evaluates the ratio of samples whose word-level edit distances are reduced after the code-comment synchronization, where these samples are referred to as Effective

Synchronized Samples (ESSs). This metric was first proposed in CBS (Yang et al. 2023) and can be formally defined as below:

$$ESS\ Ratio = \frac{N_{ED\_diff<0}}{N}. \tag{3}$$

where $N_{ED\_diff<0}$ denotes that, the number of samples whose edit distances are reduced (i.e., $edit\_distance(\hat{y}^{(k)}, y^{(k)}) - edit\_distance(x^{(k)}, y^{(k)}) < 0$).

5.4 Implementaion Details

For the implementation of large language models (LLMs), we invoke GPT-3.5 using the constructed prompts via the OpenAI API (OpenAI-CodeX 2023). Meanwhile, we deploy the Llama3 series models locally through Huggingface (Hugging Face 2025) and utilize vLLM (vLLM 2025) to accelerate the inference process, fixing the GPU memory utilization rate to 95%. All locally deployed LLMs are run on four NVIDIA A40 GPUs, each with 48 GB of memory. Regarding the hyper-parameter settings, we adopt nuclear sampling (Holtzman et al. 2026) for LLMs with *temperature*=0.8, *sampling_number*=10, and *top-p*=0.95. For R^{2}ComSync, we adopt Llama3-8B/70B and GPT-3.5-Turbo as the backbone models for experiments, using the same hyper-parameter settings as above. During the retrieval phase, we retrieve the top-2, 4, 6, 8, 10 similar demonstrations as examples (i.e., shot numbers P) and select the appropriate number of shots based on each model's performance. In the re-ranking phase, we test various combinations of σ and ϵ, and based on our exploration in Section 6.3, we finally set the thresholds for Rule 2 (σ) and Rule 3 (ϵ) to 0.35 and 0.25 for Liu's dataset, 0.35 and 0.55 for Panth's dataset, and 0.35 and 0.2 for Pai's dataset, respectively.

6 Evaluation

The following section introduces four research questions to investigate the performance of R^{2}ComSync as follows:

- **RQ2:** How effective is the R^{2}ComSync compared with the state-of-the-art baselines?
- **RQ3:** What is the contribution of the retrieval module of R^{2}ComSync.
- **RQ4:** What is the contribution of the re-ranking module of R^{2}ComSync.
- **RQ5:** Does R^{2}ComSync apply to small models?

6.1 Effectiveness of R^{2}ComSync (RQ2)

Objective Various well-designed CCS approaches have continuously emerged in recent years, yet the LLM-based approach, e.g., R^{2}ComSync, to the best of our knowledge, is proposed for the first time. Therefore, we conduct the following experiments to investigate its effectiveness and make comparisons with state-of-the-art baselines.

Experimental design Experiments towards RQ2 begin with an automatic evaluation using the metrics mentioned in Section 5.3. The experimental settings are almost the same as RQ1,

except we augment each LLM with our proposed R^{2}ComSync framework to make comparisons with SOTA approaches again. Since the ultimate goal of CCS is to reduce developers' efforts in maintaining comments when code changes, we also conduct a human evaluation as a complementary experiment. Specifically, we consider three practical aspects: similarity (whether the generated new comment is similar to the reference comment), naturalness (whether the generated new comment conforms to grammar and is fluent), and sensitivity (whether the generated new comment is keenly aware of the code changes). The score for each aspect is an integer ranging from 0 to 4 (i.e., from bad to good). Similar to previous studies (Li et al. 2024; Gao et al. 2020; Hu et al. 2020), for each dataset, we randomly select 50 samples from the testing set and divide them evenly into five groups. For each group, we list the generated new comments of each approach and their corresponding CCS targets in random order through questionnaires. Subsequently, we invite ten developers from the industrial community with 3-5 years of programming experience in Java and Python. We then ask them to work in pairs to anonymously carry out the evaluation, where the final scores are averaged based on each pair's results. Evaluators are allowed to search the Internet for unfamiliar concepts.

Results Table 4 demonstrates the CCS performance of R^{2}ComSync embedded with various LLMs. As can be seen, compared with zero-shot learning and two-shot learning across

Table 4 Code-Comment Synchronization Results of Each Approach

Dataset	Approach	Accuracy%↑	Recall@5%↑	ESS Ratio%↑
Liu's Dataset	CUP	$21.63_{\pm 1.672}$(*5.21e-3*)	$31.79_{\pm 1.478}$(*5.96e-3*)	$28.64_{\pm 1.512}$(*6.09e-3*)
	HEBCUP	$26.68_{\pm 0.226}$(*5.21e-3*)	$27.67_{\pm 0.226}$(*5.71e-3*)	$27.72_{\pm 0.025}$(*3.75e-3*)
	CBS	$28.53_{\pm 0.270}$(*4.97e-3*)	$35.76_{\pm 1.338}$(*2.96e-2*)	$31.61_{\pm 0.418}$(*5.96e-3*)
	HatCUP	24.30	35.20	31.33
	Toper	30.10	33.04	34.59
	R²ComSync Scenarios			
	GPT3.5-turbo	$29.89_{\pm 0.443}$	$33.79_{\pm 0.256}$	$36.59_{\pm 0.256}$
	Llama3-8B	$26.68_{\pm 0.993}$	$30.87_{\pm 0.981}$	$36.25_{\pm 0.1.054}$
	Llama3-70B	**$34.08_{\pm 0.133}$**	**$36.79_{\pm 0.474}$**	**$42.28_{\pm 0.504}$**
Panth's Dataset	CUP	$7.56_{\pm 0.981}$(*5.96e-3*)	$17.34_{\pm 0.494}$(*5.96e-3*)	$21.20_{\pm 1.481}$(*5.96e-3*)
	HEBCUP	$8.90_{\pm 0.109}$(*5.71e-3*)	$9.24_{\pm 0.000}$(*3.65e-3*)	$19.54_{\pm 0.323}$(*5.96e-3*)
	CBS	$11.82_{\pm 0.657}$(*5.83e-3*)	$17.66_{\pm 0.420}$(*5.83e-3*)	$25.63_{\pm 0.832}$(*5.96e-3*)
	R²ComSync Scenarios			
	GPT3.5-turbo	$16.14_{\pm 0.187}$	$17.99_{\pm 0.187}$	$26.32_{\pm 0.990}$
	Llama3-8B	$12.38_{\pm 1.105}$	$15.24_{\pm 1.111}$	$21.11_{\pm 1.983}$
	Llama3-70B	**$22.06_{\pm 0.538}$**	**$25.56_{\pm 0.645}$**	**$39.92_{\pm 1.053}$**
Pai's Dataset	CUP	$3.93_{\pm 0.434}$(*5.71e-3*)	$9.33_{\pm 1.545}$(*5.96e-3*)	$8.60_{\pm 1.322}$(*5.96e-3*)
	HEBCUP	$3.00_{\pm 0.000}$(*3.54e-3*)	$3.00_{\pm 0.000}$(*3.75e-3*)	$6.53_{\pm 0.186}$(*5.33e-3*)
	CBS	$6.14_{\pm 0.506}$(*5.58e-3*)	$10.73_{\pm 1.254}$(*6.09e-3*)	$12.2_{\pm 1.304}$(*5.83e-3*)
	R²ComSync Scenarios			
	GPT3.5-turbo	$8.89_{\pm 0.567}$	$10.22_{\pm 0.831}$	$12.00_{\pm 0.981}$
	Llama3-8B	$7.80_{\pm 0.909}$	$11.13_{\pm 0.542}$	$12.00_{\pm 0.981}$
	Llama3-70B	**$10.67_{\pm 0.558}$**	**$13.80_{\pm 0.618}$**	**$19.40_{\pm 0.827}$**

Note: *p*-value represents the hypothetical test value between each baseline and R2ComSync(Llama3-70B), the values below 0.05 are *italicized*. The best-performing results are highlighted in **bold**

diverse datasets, LLMs obtain significant improvements, demonstrating that employing R^2 ComSync can effectively harness the LLMs' capabilities for CCS tasks. Besides, Table 4 also showcases the SOTA approaches' performances. Apparently, R^{2}ComSync with different LLMs demonstrates significant superiority over current SOTA approaches overall, especially in Panth and Pai's datasets. Even the worst LLM, namely Llama3-8B, surpasses almost all SOTA approaches after being equipped with the R^{2}ComSync framework. In particular, R^{2}ComSync with Llama3-70B performs the best across all datasets. For example, on Liu's dataset, R^{2}ComSync with Llama3-70B outperforms SOTA approaches by 31.64% in terms of Accuracy, 13.16% in Recall@5, and 37.98% in ESS Ratio on average. Besides, it also surpasses existing approaches by 142.1%, 89.59%, and 82.79% in terms of each metric on Panth's dataset. Regarding the CCS for Python programs on Pai's dataset, R^2 ComSync with Llama3-70B still maintains its substantial superiority by 166.98%, 145.51%, and 127.23% in terms of each metric in order. The above analysis demonstrates that R^2 ComSync possesses a powerful CCS capability for the overall samples and can generate a great amount of *correct synchronization*s regardless of its first attempt or five attempts. Moreover, experiments across both Java and Python projects prove that R^{2}ComSync can be extended and generalized to different programming languages. To examine the significance of the improvement, we conduct Wilcoxon Signed-Rank Tests (WSRT) (Wilcoxon 1945) with a confidence level of 95% between R^{2}ComSync(Llama3-70B) and every SOTA approach with respect to each metric, where the p-values are listed behind their respective scores under each metric. Subsequently, we further conduct Cliff's Delta analysis (Aarts et al. 2014) to measure the effect size in pairs. Results demonstrate that the superiority of R^2 ComSync with Llama3-70B is statistically significant and consistently shows a large effect size.

Furthermore, Table 5 demonstrates the human evaluation results. To validate the effectiveness of the assessment, we compute Cohen's kappa coefficient (Chmura Kraemer et al. 2002) of agreement for the scores assigned by the two evaluators, finding that it consistently exceeds 0.55 across all evaluation metrics and datasets. Therefore, the evaluation process is above moderate consistency. As can be seen, compared with the results of automatic evaluation (Fig. 4), R^{2}ComSync demonstrates a more consistent superiority against SOTA approaches. Even using a relatively less capable model such as Llama3-8B, it still achieves higher scores than SOTA baselines in most scenarios. Taking the best-performing LLM, i.e., Llama3-70B, and SOTA approaches to conduct WSRT among 50 pairs of evaluation results on each dataset, the hypothetical tests concerning the improvements are all statistically significant (p-value$<$0.05) as shown in Table 5. Moreover, the Cliff's Delta analysis also reveals that their effect sizes are all non-negligible. This proves the practical value of R^{2}ComSync, showing that R^{2}ComSync can generate comments that developers prefer, not just in terms of correctness (i.e., similarity and sensitivity), but also in terms of naturalness. We attribute the outperformance in human evaluation to using LLMs as R^{2}ComSync's backbone, as they were pretrained on massive human-written corpora, providing a solid foundation for generating human-preference text. Another worth noting phenomenon is that GPT-3.5-Turbo has closed the distance with Llama3-70B when equipped with R^{2}ComSync in the human evaluation. Particularly, in Panth and Pai's datasets, it even slightly surpasses Llama3-70B across all evaluation metrics. Although the WSRT has revealed that the performance

Table 5 Human Evaluation: Code-Comment Synchronization Results of Each Approach

Dataset	Approaches	Similarity↑	Naturalness↑	Sensitivity↑
Liu's Dataset	CUP	$2.11_{\pm 0.11}$ (2.30e-5)	$3.42_{\pm 0.06}$ (3.84e-5)	$1.92_{\pm 0.14}$ (1.04e-5)
	HEBCUP	$2.11_{\pm 0.01}$ (3.55e-5)	$3.13_{\pm 0.09}$ (3.87e-8)	$1.99_{\pm 0.17}$ (2.71e-5)
	HatCUP	$1.89_{\pm 0.03}$ (4.46e-7)	$2.75_{\pm 0.13}$ (3.55e-15)	$1.82_{\pm 0.14}$ (1.38e-6)
	CBS	$2.08_{\pm 0.06}$ (1.96e-5)	$3.25_{\pm 0.05}$ (2.28e-6)	$1.90_{\pm 0.12}$ (9.08e-6)
	Toper	$1.74_{\pm 0.02}$ (9.67e-8)	$2.34_{\pm 0.16}$ (1.60e-16)	$1.59_{\pm 0.15}$ (6.41e-8)
	R²ComSync Scenarios			
	Llama3-8B	$2.34_{\pm 0.10}$	$3.36_{\pm 0.04}$	$2.26_{\pm 0.14}$
	GPT3.5-turbo	$2.90_{\pm 0.08}$	$3.84_{\pm 0.08}$	$2.84_{\pm 0.12}$
	Llama3-70B	$\mathbf{3.04_{\pm 0.04}}$	$\mathbf{3.85_{\pm 0.05}}$	$\mathbf{2.99_{\pm 0.05}}$
Panth's Dataset	CUP	$1.82_{\pm 0.18}$ (1.37e-3)	$3.62_{\pm 0.10}$ (3.86e-2)	$1.60_{\pm 0.26}$ (2.03e-4)
	HEBCUP	$1.82_{\pm 0.16}$ (1.72e-3)	$3.28_{\pm 0.04}$ (6.70e-4)	$1.58_{\pm 0.26}$ (3.76e-4)
	CBS	$2.13_{\pm 0.15}$ (3.62e-2)	$3.56_{\pm 0.08}$ (2.20e-2)	$1.95_{\pm 0.21}$ (1.10e-2)
	R²ComSync Scenarios			
	Llama3-8B	$1.86_{\pm 0.14}$	$3.06_{\pm 0.00}$	$1.96_{\pm 0.16}$
	GPT3.5-turbo	$\mathbf{2.87_{\pm 0.03}}$	$\mathbf{3.79_{0.01}}$	$\mathbf{2.97_{\pm 0.05}}$
	Llama3-70B	$2.66_{\pm 0.04}$	$\mathbf{3.79_{\pm 0.03}}$	$2.67_{\pm 0.17}$
Pai's Dataset	CUP	$1.88_{\pm 0.13}$ (1.39e-12)	$2.37_{\pm 0.15}$ (3.12e-15)	$0.79_{\pm 0.33}$ (7.13e-14)
	HEBCUP	$2.31_{\pm 0.13}$ (9.19e-11)	$2.81_{\pm 0.19}$ (2.19e-14)	$1.19_{\pm 0.33}$ (3.49e-12)
	CBS	$2.12_{\pm 0.24}$ (1.22e-11)	$2.78_{\pm 0.24}$ (1.22e-14)	$1.04_{\pm 0.14}$ (6.70e-14)
	R²ComSync Scenarios			
	Llama3-8B	$2.84_{\pm 0.12}$	$3.32_{\pm 0.12}$	$2.47_{\pm 0.11}$
	GPT3.5-turbo	$\mathbf{3.44_{\pm 0.06}}$	$\mathbf{3.83_{\pm 0.07}}$	$\mathbf{3.19_{\pm 0.01}}$
	Llama3-70B	$3.32_{\pm 0.12}$	$3.74_{\pm 0.12}$	$3.05_{\pm 0.09}$

The best-performing results are highlighted in **bold**

differences between them are not significant, it still illustrates that R²ComSync with GPT-3.5-Turbo tends to generate new comments that are more satisfying to human beings than similar to ground truths.

▶▶RQ2: R²ComSync significantly lifts LLMs' performance in the CCS task, and most studied LLMs outperform SOTA approaches after equipping with R²ComSync, especially in the human evaluations.

6.2 Contributions of Ensemble Hybrid Retrieval (RQ3)

Objective Although we have verified the effectiveness of R²ComSync as a whole in Section 6.1, we still need to quantify the contribution of the Ensemble Hybrid Retrieval (EHR) method, thereby proving our motivation for retrieving CCS demonstrations from perspectives of both code-comment semantic and change patterns is worthwhile.

Experimental design To carry out the ablation study for the EHR module, we first remove the MR strategy to present the performance of our proposal equipped with EHR solely. Since the EHR module is a composite method of CodeBERT retrieval and expert retrieval,

we compare the EHR module with the above two retrieval methods, respectively. Besides, we also include the random selection of demonstrations for comparison. To eliminate the performance discrepancies across different shot numbers (P), we feed LLMs with $P = \{2, 4, 6, 8, 10\}$ in order and then average their performances to evaluate the contribution of the EHR module and make comparisons with other retrieval methods. It should be noted that experiments are repeated five times for each setting of P. Subsequently, based on the above experimental results with different shot numbers, we investigate the performance variation. All the above experiments are conducted across all datasets with all LLMs under test.

Results First, we compared the impact of different retrieval methods on LLMs' performance, where the specific experimental results are shown in Table 6. As can be seen, in Liu and Pai's datasets, EHR performs the best across almost three studied LLMs on most occasions. On average, EHR outperforms CodeBERT retrieval by 9.58% in terms of Accuracy, by 8.44% in terms of Recall@5, and by 1.72% in terms of ESS Ratio across all studied LLMs on Pai's dataset. As for expert retrieval, EHR still maintains its superiority by 26.27%, 16.50%, and 10.42% in each metric, respectively. When it turns to random retrieval, EHR manifests much more pronounced advantages by 56.51%, 30.56%, and 17.70% on each metric in order. Similar trends occur in Liu's dataset, where EHR dominates among all retrieval methods, CodeBERT retrieval performs the second best, and expert retrieval surpasses the random one. We attribute this to the fact that CCS samples of Liu and Pai's dataset are relatively longer (See Section 5.1 for details), especially Pai's dataset, and expert features solely may have difficulties in fully capturing their diverse characteristics. Besides, the longest code and comment length also induce the absolute worst performance on Pai's dataset across all studied LLMs.

As for the experimental results on Panth's dataset, the expert retrieval method achieves the best performance, outperforming other retrieval methods, including EHR, by 13.74%–72.66% in terms of Accuracy, by 9.05%–41.14% in terms of Recall@5, and by 5.50%–28.25% in terms of ESS Ratio across all studied LLMs on average. A possible explanation is as follows: The sample distribution of Panth's dataset exhibits significant discrepancies, as shown in Table 3, with a much higher standard deviation on both code and comment lengths, as well as their corresponding NCSs. Thus, it can be regarded as a "high-variability dataset". CodeBERT relies on alignment between the data distribution and its pretraining data in semantic similarity tasks (Zhou et al. 2021). A high standard deviation may disrupt this alignment, leading to performance degradation and resulting in lower-quality retrieval samples for EHR, thereby affecting its performance. On the contrary, expert features are more precise in sample representation for shorter ones, as these patterns are easier and explicit. Besides, they need not proactively learn crucial features among samples, leading to their surpassing in Panth's dataset. Even though expert retrieval performs the best in Panth's dataset, our proposed EHR consistently performs second-best across almost all LLMs and evaluation metrics, demonstrating its effectiveness. Although CCS samples in diverse datasets exhibit varying proneness to different retrieval methods, they all perform worst with random retrieval, demonstrating that relevant examples are crucial for LLM-based code-comment synchronization. For practitioners, it may be beneficial to consider different mixing ratios for EHR depending on the degree of dispersion in the sample distribution. For

Table 6 Experimental Results Concerning the Influence of EHR

Dataset	Models	Methods	Accuracy%↑	Recall@5%↑	ESS Ratio%↑
Liu's Dataset	GPT3.5-turbo	Random	$18.61_{\pm 0.287}$	$26.82_{\pm 0.304}$	$24.85_{\pm 0.469}$
		Expert	$20.21_{\pm 1.793}$	$27.81_{\pm 1.112}$	$26.38_{\pm 1.275}$
		CodeBERT	$23.78_{\pm 0.501}$	$31.16_{\pm 0.392}$	$29.38_{\pm 1.109}$
		EHR	$\mathbf{25.78_{\pm 0.720}}$	$\mathbf{32.07_{\pm 0.390}}$	$\mathbf{31.54_{\pm 0.650}}$
	Llama3-8B	Random	$7.45_{\pm 0.550}$	$15.38_{\pm 0.594}$	$10.22_{\pm 0.706}$
		Expert	$11.36_{\pm 0.827}$	$19.43_{\pm 0.863}$	$15.74_{\pm 1.132}$
		CodeBERT	$\mathbf{16.26_{\pm 0.703}}$	$\mathbf{25.48_{\pm 0.699}}$	$\mathbf{21.02_{\pm 0.877}}$
		EHR	$15.62_{\pm 0.922}$	$24.67_{\pm 0.809}$	$20.09_{\pm 1.033}$
	Llama3-70B	Random	$25.54_{\pm 0.805}$	$31.03_{\pm 0.301}$	$32.74_{\pm 1.021}$
		Expert	$27.67_{\pm 0.605}$	$32.83_{\pm 0.435}$	$34.97_{\pm 0.807}$
		CodeBERT	$29.37_{\pm 1.060}$	$34.32_{\pm 0.828}$	$37.67_{\pm 1.346}$
		EHR	$\mathbf{30.14_{\pm 0.609}}$	$\mathbf{34.77_{\pm 0.507}}$	$\mathbf{38.37_{\pm 0.876}}$
Panth's Dataset	GPT3.5-turbo	Random	$10.61_{\pm 0.470}$	$15.53_{\pm 0.470}$	$18.19_{\pm 0.470}$
		Expert	$\mathbf{13.99_{\pm 0.737}}$	$\mathbf{18.84_{\pm 0.518}}$	$\mathbf{23.02_{\pm 1.002}}$
		CodeBERT	$11.51_{\pm 0.494}$	$16.24_{\pm 0.494}$	$20.00_{\pm 0.626}$
		EHR	$12.59_{\pm 0.614}$	$16.75_{\pm 0.415}$	$20.56_{\pm 1.282}$
	Llama3-8B	Random	$2.04_{\pm 0.214}$	$5.95_{\pm 1.022}$	$9.23_{\pm 0.816}$
		Expert	$\mathbf{5.35_{\pm 0.871}}$	$\mathbf{10.92_{\pm 0.872}}$	$\mathbf{13.05_{\pm 1.206}}$
		CodeBERT	$4.32_{\pm 0.738}$	$9.40_{\pm 0.760}$	$12.29_{\pm 1.258}$
		EHR	$4.35_{\pm 0.693}$	$9.83_{\pm 0.775}$	$12.48_{\pm 0.999}$
	Llama3-70B	Random	$15.66_{\pm 1.183}$	$20.51_{\pm 0.682}$	$28.36_{\pm 1.366}$
		Expert	$\mathbf{19.40_{\pm 0.717}}$	$\mathbf{24.32_{\pm 0.549}}$	$33.13_{\pm 1.046}$
		CodeBERT	$16.94_{\pm 0.862}$	$22.16_{\pm 0.626}$	$30.63_{\pm 1.456}$
		EHR	$18.11_{\pm 0.717}$	$23.48_{\pm 0.549}$	$\mathbf{33.14_{\pm 1.046}}$
Pai's Dataset	GPT3.5-turbo	Random	$5.53_{\pm 0.272}$	$8.60_{\pm 0.272}$	$10.53_{\pm 0.272}$
		Expert	$5.69_{\pm 0.470}$	$9.20_{\pm 0.496}$	$12.20_{\pm 0.565}$
		CodeBERT	$\mathbf{6.73_{\pm 0.758}}$	$\mathbf{9.84_{\pm 0.396}}$	$11.58_{\pm 0.587}$
		EHR	$6.38_{\pm 0.360}$	$9.56_{\pm 0.466}$	$\mathbf{12.31_{\pm 0.981}}$
	Llama3-8B	Random	$1.78_{\pm 0.447}$	$4.73_{\pm 0.517}$	$9.40_{\pm 0.796}$
		Expert	$2.29_{\pm 0.630}$	$5.76_{\pm 0.668}$	$9.59_{\pm 0.673}$
		CodeBERT	$\mathbf{3.24_{\pm 0.776}}$	$7.25_{\pm 0.731}$	$\mathbf{11.60_{\pm 0.884}}$
		EHR	$3.23_{\pm 0.814}$	$\mathbf{7.33_{\pm 0.724}}$	$10.81_{\pm 1.538}$
	Llama3-70B	Random	$4.76_{\pm 0.666}$	$9.31_{\pm 0.380}$	$14.18_{\pm 1.118}$
		Expert	$5.93_{\pm 0.578}$	$9.88_{\pm 0.434}$	$14.97_{\pm 1.109}$
		CodeBERT	$5.55_{\pm 0.435}$	$9.20_{\pm 0.640}$	$16.67_{\pm 0.872}$
		EHR	$\mathbf{7.45_{\pm 0.636}}$	$\mathbf{11.69_{\pm 0.553}}$	$\mathbf{17.61_{\pm 0.940}}$

Note: Each model's best-performing results are highlighted in **bold**, and their second-best-performing results are highlighted with underlines

datasets with more dispersed sample distributions or shorter samples, a larger weight can be assigned to expert retrieval; otherwise, a larger weight can be assigned to CodeBERT's retrieval results.

Next, based on the above experimental results, we further analyze the impact of different shot numbers (P) on the LLMs' performance, where the experimental results are shown in Fig. 5. As can be seen, for GPT-3.5-Turbo and Llama3-70B, with more CCS demonstrations, they tend to perform gradually better, but this tendency cannot last forever; it drops

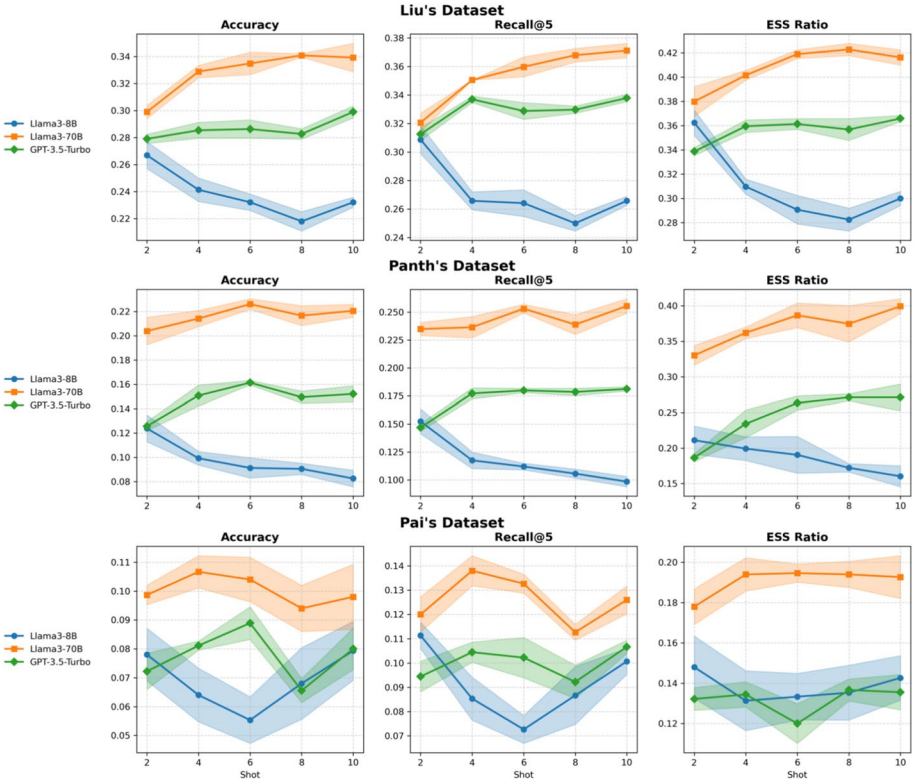

Fig. 5 LLMs' Performance Variation among Different Shot Numbers across three different datasets

at a certain shot number, which is also the same in other in-context learning scenarios (Xue et al. 2024; Li et al. 2024; Dong et al. 2024). For Liu and Panth's dataset, their peak values tend to be lagged compared to those in Pai's dataset, as the latter has much longer code and comment lengths on average. This makes it harder for the above LLMs to leverage the relatively more complex contextual information, given the same shot numbers. Nonetheless, as for Llama3-8B, it performs a declining trend overall, regardless of any CCS dataset. Owing to the fact that LLMs' performance scales following a power law with respect to parameter count, data volume, and computational resources, enabling more effective utilization of additional contextual information (Kaplan et al. 2020). Hence, Llama3-8B apparently has difficulty in harnessing the given contextual information as the number of shots increases, due to its smaller parameter size. Therefore, selecting an appropriate shot number based on diverse LLMs and datasets can enhance the performance of code-comment synchronization.

RQ3: The EHR method substantially enhances LLMs' performance on the CCS task and either performs the best or second best among different retrieval methods. Moreover, for different LLMs and datasets, selecting an appropriate shot number is also crucial.

6.3 Contributions of Multi-Turn Re-ranking Strategy (RQ4)

Objective Similar to RQ3, to examine whether the Multi-turn Re-ranking (MR) strategy can effectively prioritize correct-prone candidates, this subsection quantifies its contribution in R^{2}ComSync. Besides, we also analyze the threshold settings of Rules 2 (i.e., σ) and 3 (i.e., ϵ) in the MR strategy, thereby investigating its influence even further.

Experimental design To verify the effectiveness of the MR strategy, we carry out another group of ablation experiments. Specifically, we first list R^{2}ComSync without MR strategy as the baseline (i.e., W/O MR), then adopt Rule 1 to re-rank candidates generated by LLMs (i.e., MR-1), thereby quantifying the contribution of Rule 1. Afterwards, we further impose Rules 2 (i.e., MR-1-2) and 3 (i.e., MR-1-2-3) on it successively to complete the ablation experiment. For different models and datasets, we refer to Fig. 5 to select the shot with the best performance for re-ranking experiments. Since the experiments for each shot number are repeated five times, their follow-up re-ranking results are also recorded, respectively, as shown in Table 7. Regarding the study on threshold settings for Rule 2 (σ) and 3 (ϵ), we investigate the performance of Llama3-70B using various threshold combinations across the three datasets. Specifically, based on the findings in Section 3, we adjust (σ) from 0.2 to 0.4 and (ϵ) from 0.2 to 0.6 with a step size of 0.05. Accordingly, we exhaustively carry out 5*9*3=135 groups of experiments to observe the influence of each pair of threshold combinations on R^{2}ComSync.

Results Table 7 demonstrates the results of ablation experiments concerning the MR strategy. Firstly, compared with R^{2}ComSync without MR strategy (i.e., W/O MR), using the rules we proposed consecutively to re-rank the candidate values can lead to significant improvements by 15.71%-46.65% in terms of Accuracy, 3.48%-9.85% in terms of Recall@5, and 8.60%-20.29% in terms of ESS Ratio across three datasets, indicating that re-ranking the candidates generated by LLM is necessary for the CCS field. Besides, as can be seen, both Rule 2 and Rule 3 can almost progressively boost CCS performance, proving that the effectiveness of these rules, which were summarized from Liu's datasets, can be generalized to other CCS scenarios in practice. However, Rule 1 seems to be only effective in Liu's dataset, demonstrating its overfitting to our analyzed samples from Liu's dataset. Besides, Rule 1's contribution is relatively limited compared to Rules 2 and 3, further indicating its weak level in the MR strategy.

Afterwards, we further analyze the threshold setting in Rules 2 and 3, where the experimental results are shown in Fig. 6. It can be observed that on Liu's dataset, as σ and ϵ increase, the performance of R^{2}ComSync in terms of Accuracy generally decreases, while Recall@5 and ESS Ratio initially increase and then decline. In contrast, on Panth's dataset, all three metrics exhibit an upward trend as both thresholds increase. Nonetheless, as for Pai's dataset, R^{2}ComSync's both Accuracy and ESS Ratio scores manifest a trend of declining first, then increasing, but the former is mainly controlled by ϵ while the latter is almost solely affected by σ. An interesting finding is on Recall@5, where it is consistently stable across diverse combinations of ϵ and σ. A potential explanation is that Pai's dataset has the relatively longest comments, which leads to the extreme difficulty in generating exactly *correct synchronization*, given the limited attempts. Hence, even though we continuously

Table 7 Results Concerning the Influence of Multi-turn Re-ranking

Dataset	Model	Components	Accuracy%↑	Recall@5%↑	ESS Ratio%↑
Liu's Dataset	GPT-3.5-Turbo	W/O MR	$27.45_{\pm 0.587}$	$33.24_{\pm 0.558}$	$33.61_{\pm 0.462}$
		MR-1	$28.08_{\pm 0.779}$	$33.24_{\pm 0.558}$	$34.24_{\pm 0.444}$
		MR-1-2	$29.17_{\pm 0.462}$	$33.51_{\pm 0.339}$	$36.14_{\pm 0.222}$
		MR-1-2-3	$\mathbf{29.89_{\pm 0.444}}$	$\mathbf{33.79_{\pm 0.256}}$	$\mathbf{36.59_{\pm 0.256}}$
	Llama3-8B	W/O MR	$19.95_{\pm 0.966}$	$29.46_{\pm 0.935}$	$26.96_{\pm 1.347}$
		MR-1	$20.65_{\pm 0.749}$	$29.51_{\pm 0.919}$	$27.61_{\pm 1.095}$
		MR-1-2	$24.46_{\pm 1.100}$	$30.82_{\pm 0.817}$	$34.18_{\pm 1.713}$
		MR-1-2-3	$\mathbf{26.68_{\pm 0.993}}$	$\mathbf{30.87_{\pm 0.981}}$	$\mathbf{36.15_{\pm 1.054}}$
	Llama3-70B	W/O MR	$30.60_{\pm 0.504}$	$35.92_{\pm 0.554}$	$38.70_{\pm 0.741}$
		MR-1	$30.71_{\pm 0.570}$	$35.87_{\pm 0.595}$	$38.64_{\pm 0.916}$
		MR-1-2	$32.17_{\pm 0.657}$	$36.47_{\pm 0.605}$	$40.76_{\pm 0.666}$
		MR-1-2-3	$\mathbf{34.08_{\pm 0.133}}$	$\mathbf{36.79_{\pm 0.474}}$	$\mathbf{42.28_{\pm 0.504}}$
Panth's Dataset	GPT-3.5-Turbo	W/O MR	$13.36_{\pm 0.374}$	$17.59_{\pm 0.374}$	$22.22_{\pm 1.620}$
		MR-1	$13.36_{\pm 0.374}$	$17.59_{\pm 0.374}$	$22.22_{\pm 1.620}$
		MR-1-2	$15.21_{\pm 0.815}$	$17.86_{\pm 0.324}$	$24.74_{\pm 0.935}$
		MR-1-2-3	$\mathbf{16.14_{\pm 0.187}}$	$\mathbf{17.99_{\pm 0.187}}$	$\mathbf{26.32_{\pm 0.990}}$
	Llama3-8B	W/O MR	$7.70_{\pm 0.692}$	$13.65_{\pm 0.736}$	$16.90_{\pm 1.193}$
		MR-1	$7.70_{\pm 0.692}$	$13.65_{\pm 0.736}$	$16.90_{\pm 1.193}$
		MR-1-2	$11.59_{\pm 1.077}$	$15.24_{\pm 1.111}$	$21.03_{\pm 2.085}$
		MR-1-2-3	$\mathbf{12.38_{\pm 1.105}}$	$\mathbf{15.24_{\pm 1.111}}$	$\mathbf{21.11_{\pm 1.983}}$
	Llama3-70B	W/O MR	$18.25_{\pm 0.905}$	$24.29_{\pm 0.985}$	$33.97_{\pm 1.397}$
		MR-1	$18.25_{\pm 0.905}$	$23.49_{\pm 0.985}$	$34.52_{\pm 1.397}$
		MR-1-2	$20.79_{\pm 0.855}$	$23.89_{\pm 0.952}$	$36.98_{\pm 2.236}$
		MR-1-2-3	$\mathbf{22.06_{\pm 0.817}}$	$\mathbf{25.56_{\pm 0.884}}$	$\mathbf{39.92_{\pm 2.567}}$
Pai's Dataset	GPT-3.5-Turbo	W/O MR	$6.33_{\pm 0.272}$	$9.89_{\pm 0.567}$	$11.78_{\pm 1.110}$
		MR-1	$6.33_{\pm 0.272}$	$9.89_{\pm 0.567}$	$11.78_{\pm 1.100}$
		MR-1-2	$7.56_{\pm 0.685}$	$10.11_{\pm 0.875}$	$\mathbf{12.22_{\pm 0.875}}$
		MR-1-2-3	$\mathbf{8.89_{\pm 0.567}}$	$\mathbf{10.22_{\pm 0.831}}$	$12.00_{\pm 0.981}$
	Llama3-8B	W/O MR	$4.40_{\pm 0.800}$	$9.40_{\pm 0.389}$	$12.33_{\pm 0.023}$
		MR-1	$4.40_{\pm 0.800}$	$9.40_{\pm 0.389}$	$12.33_{\pm 2.280}$
		MR-1-2	$6.13_{\pm 0.686}$	$10.40_{\pm 0.611}$	$13.87_{\pm 1.916}$
		MR-1-2-3	$\mathbf{7.80_{\pm 0.909}}$	$\mathbf{11.13_{\pm 0.542}}$	$\mathbf{14.80_{\pm 1.543}}$
	Llama3-70B	W/O MR	$8.73_{\pm 0.646}$	$12.80_{\pm 0.452}$	$18.67_{\pm 0.869}$
		MR-1	$8.73_{\pm 0.646}$	$12.80_{\pm 0.452}$	$18.67_{\pm 0.869}$
		MR-1-2	$9.80_{\pm 0.581}$	$13.40_{\pm 0.646}$	$19.33_{\pm 0.471}$
		MR-1-2-3	$\mathbf{10.67_{\pm 0.558}}$	$\mathbf{13.80_{\pm 0.618}}$	$\mathbf{19.40_{\pm 0.827}}$

The best-performing results are highlighted in **bold**

change their candidate orders by altering re-ranking hyper-parameters, the Recall@5 keep almost unchanged. The thresholds used in previous RQs are marked with red dots in Fig. 6, and these settings have achieved or are close to optimal. It is important to note that adjustments for σ and ϵ will significantly affect the CCS performance of R^{2}ComSync in most scenarios. Therefore, we recommend that practitioners use our settings as a baseline and further tune the values of σ and ϵ according to the specific circumstances to achieve better performances.

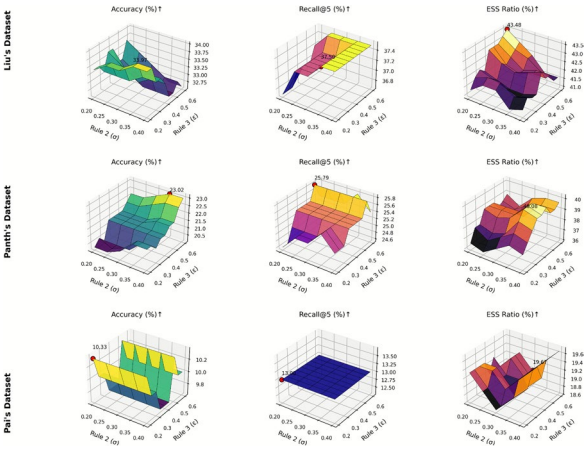

Fig. 6 Sensitivity Analysis of Llama3-70B's Performance Metrics on Rule 2 (σ) and Rule 3 (ϵ)

▶▶**RQ4:** The MR strategy is effective in prioritizing correct-prone candidates. Towards the threshold configuration, we recommend practitioners use our setting as an initial configuration and tune it further according to their specific demands.

6.4 Applicability of R^{2}ComSync to Small Models (RQ5)

Objective In RQ2, we have investigated R^{2}ComSync's effectiveness on a series of LLMs with over 8B parameters. However, considering the resource limitations and restrictions of using these large LLMs in practice, it is imperative to examine whether R^{2}ComSync is still applicable on small LLMs.

Experimental design Considering that the very recently released Qwen2.5 series contains powerful LLMs with diverse parameter sizes and both code and general versions, we select two of this series (i.e., Qwen2.5-1.5B Qwen et al. 2025 and Qwen2.5-Coder-1.5B Hui et al. 2024) for experiments, thereby exploring the effectiveness of R^{2}ComSync on small models. In addition, Qwen2.5-1.5B is a general LLM, while Qwen2.5-Coder-1.5B is a code LLM. Hence, we can further study performance discrepancies between code and general LLMs fairly on the CCS task. For implementation, we download their model parameters from Huggingface to deploy them locally, and follow Section 5.4 to configure hyper-parameters for LLMs. Considering the shot number (P) setting also affects the performance of R^{2}ComSync, we conduct multiple experiments with P=$\{2, 4, 6, 8, 10\}$ and average their performance for analysis. Comparative experiments among R^{2}ComSync and its ablated versions are conducted across all three datasets, covering both Java and Python programming languages. As shown in Fig. 7, we use "Base" to denote small LLMs without the equipment of any of our proposals, using "EHR" denotes small LLMs equipped with the EHR method for ICL, and using "EHR+MR (R^{2}ComSync)" denotes small LLMs equipped with our whole proposal.

Results As can be seen from Fig. 7, our proposal, i.e., R^{2}ComSync, is still effective for lifting small LLMs' performance in the CCS task. From the perspective of R^{2}ComSync as a whole,

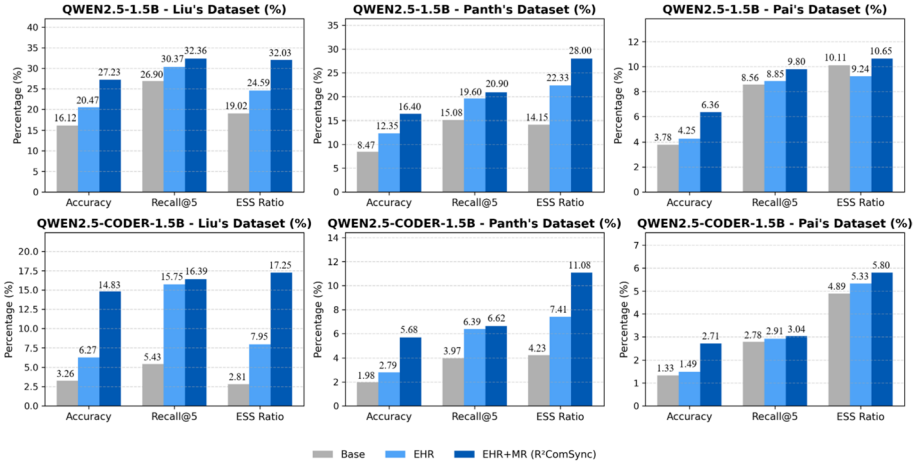

Fig. 7 Performance of R^{2}ComSync on Small Models

it improves Qwen2.5-1.5B by 76.77% in terms of Accuracy, 24.27% in terms of Recall@5, and 57.32% in terms of ESS Ratio across all studied datasets on average. Besides, it also, on average, lifts Qwen2.5-1.5B-Coder by 213.73%, 91.95%, and 231.31% in terms of each evaluation metric in order. When small LLMs are only equipped with the EHR method, their performance improvement against their Base situation is still substantial across almost all datasets and evaluation metrics. To be specific, the EHR method improves Qwen2.5-1.5B by 28.40% in terms of Accuracy, 15.05% in terms of Recall@5, and 26.24% in terms of ESS Ratio across all three datasets on average. As for Qwen2.5-1.5B-Coder, the improvement becomes more significant, achieving a 48.76% increase in Accuracy, 85.39% in Recall@5, and 90.02% in ESS Ratio on average across all datasets. The above analysis demonstrates the effectiveness of each of the modules we proposed on small LLMs. Horizontally comparing Qwen2.5-1.5B and Qwen2.5-1.5B-Coder, we find that the former consistently maintains a pronounced advantage. One potential explanation is that Qwen2.5-1.5B was pre-trained on a much larger corpus of 18 trillion tokens, including the code and math corpora used in Qwen2.5-1.5B-Coder (Qwen et al. 2025). In contrast, the pre-training data for Qwen2.5-1.5B-Coder is only 5.2 trillion, and it uses relatively outdated data filtering techniques, which makes its capability for supervised fine-tuning less effective (Hui et al. 2024).

RQ5: R^{2}ComSync is still applicable to small models and can significantly improve their CCS capabilities as well.

7 Discussion

7.1 Cost Analysis

Table 8 showcases the GPU memory and time cost of each studied approach for each dataset per trial, where they are further partitioned into “Train”, “Test”, and “Sample”. “Train” and

Table 8 Time/GPU Memory Cost Analysis of Different Approaches on Each Dataset per Trial

Model		Liu's Dataset (Train/Test/Sample)	Panth's Dataset (Train/Test/Sample)	Pai's Dataset (Train/Test/Sample)
CUP	Time(s)	1461.20/13.50/0.037	1440.00/18.03/0.049	1426.90/22.10/0.074
	Memory(GB)	46.5/0.642/0.642	46.2/0.608/0.608	47.1/0.660/0.660
HEBCUP	Time(s)	—/0.21/0.0006	—/0.80/0.001	—/0.42/0.001
	Memory(GB)	—/—/—	—/—/—	—/—/—
CBS	Time(s)	1619.70/8.08/0.022	1459.00/15.19/0.021	1444.00/18.06/0.060
	Memory(GB)	46.5/0.642/0.642	46.2/0.608/0.608	47.1/0.660/0.660
R^{2}ComSync Scenarios				
Llama3-70B	Time(s)	—/2313.27/6.286	—/4582.74/6.227	—/3704.97/12.350
	Memory(GB)	—/145.352/145.352	—/144.847/144.847	—/149.938/149.938
Llama3-8B	Time(s)	—/334.85/0.910	—/696.28/0.946	—/670.60/2.235
	Memory(GB)	—/17.075/17.075	—/16.958/16.958	—/18.023/18.023
GPT-3.5-Turbo	Time(s)	—/453.60/1.233	—/882.83/1.199	—/523.20/1.744
	Memory(GB)	—/—/—	—/—/—	—/—/—
Qwen2.5-1.5B	Time(s)	—/130.77/0.355	—/223.13/0.303—	—/230.81/0.769
	Memory(GB)	—/9.271/9.271	/8.643/8.643	—/14.915/14.915
Qwen2.5-Coder-1.5B	Time(s)	—/159.59/0.434	—/299.19/0.407—	—/255.09/0.850
	Memory(GB)	—/9.271/9.271	/8.643/8.643	—/14.915/14.915

Note: The reported GPU memory usage represents the theoretical value because we employ vLLM for inference acceleration, using a fixed memory allocation of $48 \times 4 \times 0.95$ GB for LLMs

"Test" denote the total cost during the training and testing stage, respectively. "Sample" denotes the cost of running each sample during the testing stage, thereby providing a reference for professionals for their practical usage. It should be noted that the testing stage follows a sample-by-sample inference; thus, the total GPU memory consumption in "Test" and "Sample" is the same. Besides, we also exclude HatCUP and Toper, due to their non-reproducibility, as mentioned in Section 3. As can be seen, HEBCUP, as a heuristic-based approach without GPU support, consumes minimal computational resources per trial. The time and memory costs of R^{2}ComSync include retrieval, inference, and re-ranking phases. Since LLMs in R^{2}ComSync are mandated for direct inference, their training cost is 0. Besides, in our experiments, shot number (P) has been tested from 2 to 10 for R^{2}ComSync, we average the time and memory cost across diverse shot numbers, owing to their significant influence on cost. Apparently, since LLMs normally carry massive parameters, their memory and time costs are higher than existing approaches overall. However, their cost is still acceptable, especially for a single sample when making inferences. Except for Llama3-70B, R^{2}ComSync with most LLMs can complete a synchronization within 1 second, while still keeping substantial superiority against existing approaches overall, as shown in Sections 6.1 and 6.4. As for GPT-3.5-Turbo, since it was invoked from OpenAI's APIs rather than deployed locally, we cannot obtain its GPU memory consumption. However, we introduce another cost analysis perspective: token consumption, shown in Table 9, where both the input and output token consumption are recorded in detail for reference.

7.2 Case Study

Figures 8, 9, and 10 present three typical case studies that demonstrate the effectiveness of each proposed module. For clarification, we use red (green) lines in the code snippet to

Table 9 Total Token Consumption and Cost of GPT-3.5-Turbo on Each Dataset per Trial

Liu's Dataset (Input/Output/ Total Cost)	Panth's Dataset (Input/Output/ Total Cost)	Pai's Dataset (Input/Output/ Total Cost)
195158.2/12641.0/$0.052	360,315.4/29,172.6/$0.060	278,088.6/52,411.4/$0.060

Note: The cost of GPT-3.5-Turbo is calculated based on 0.75 per million output tokens (Islam et al. 2025)

denote deleted (newly added) statements, and we also highlight the removed (newly added) tokens in red (green) for old (new) comments.

In Case Study 1 (Fig. 8), we showcase an example from apache/geode. In the original code, the condition "if (this.recipients != null)" is used to ensure that "recipients" can only be set once, throwing an "IllegalStateException" if violated. In the updated new code, however, this check has been removed, allowing "recipients" to be set repeatedly. Additionally, the method for array conversion has been modified: the original code uses "recipients. toArray(new InternalDistributedMember[0])", whereas the new code performs an explicit cast with "(InternalDistributedMember[]) recipients.toArray(_RECIPIENTS_ARRAY)". These two changes significantly affect the semantics of the code. Given this CCS target, we

- A Code-Comment Synchronization (CCS) target -

Project: apache/geode
Commit ID: 0c1b33f586fff12ae8c3c9fef1635cac792eee09

```
1  -    public void setRecipients(Collection<? extends DistributedMember> recipients) {
2  +  public void setRecipients(Collection recipients) {
3  -    if (this.recipients != null) {
4  -      throw new IllegalStateException(\"Recipients can only be set once\");
5  -    }
6  -    this.recipients = recipients.toArray(new InternalDistributedMember[0]);
7  +     this.recipients=(InternalDistributedMember[]) recipients.toArray(EMPTY_RECIPIENTS_ARRAY);
8       }
```

Old Comment: If recipient set contains ~~{@link #ALL_RECIPIENTS}~~then the message will be sent to all distribution managers.

Reference Comment: If recipient set contains Message.ALL_RECIPIENTS then the message will be sent to all distribution managers.

- CodeBERT Retriever -

- A Retrieved Demonstration Example -

```
1  public static void sort(byte[] a) {
2  - if (a.length >COUNTING_SORT_THRESHOLD_FOR_BYTE){
3  -    countingSort(a, 0, a.length - 1);
4  - }
5  - else {
6  -    sort(a, 0, a.length - 1, true);
7  +    sort(a, 0, a.length - 1);
8  }
```

Old Comment: Sorts the specified array ~~into ascending numerical order~~.

Reference Comment: Sorts the specified array.

- Expert Retriever -

- A Retrieved Demonstration Example -

```
1  -  public boolean isVisible(int column) {
2  +  public boolean isEnabled(int column) {
3  -      return visibility[column];
4  +      return enabled[column];
5  }
```

Old Comment: Returns if the specified column is ~~visible~~.

Reference Comment: Returns if the specified column is enabled.

- Outputs -

R²ComSync (CodeBERT)	Set the recipients for this distributed operation. ✘
R²ComSync (Expert)	Set the recipients for the message to be sent to specific distributed members. ✘
R²ComSync (EHR)	If recipient set contains Message.ALL_RECIPIENTS then the message will be sent to all distribution managers. ✓

Fig. 8 Case Study 1, Illustrating the Strength of the EHR Method

use CodeBERT and expert features to retrieve the most similar example, respectively. As can be seen, CodeBERT-retrieved example clearly eliminates the threshold-checking logic "if (a.length> COUNTING_SORT_THRESHOLD_FOR_BYTE)" which is semantically the same as the cod changes in the CCS target. As for the expert-retrieved example, it carries similar code-comment change patterns in terms of the change complexity and involvement. However, neither of them can make LLMs generate *correct synchronization* solely. But when we gather these two examples together for ICL, we obtain the final correct result.

In Case Study 2 (Fig. 9), the CCS target modifies the return type from "Hashtable" to the generic "Map<String, DatatypeValidator>" and accordingly makes some follow-up changes in code, such as optimizing collection operations and constituting a type-safety enhancement. The above changes are precisely reflected in comments, i.e., change "hashtable" to "map". CodeBERT retrieved an example featuring logical augmentation through adding conditional checks, encompassing array operations. Semantically, the CCS target is similar to this example, as "Array" intuitively contains a relatively similar representation to "Map" and "Hashtable", owing to their data structure usage. However, this example cannot instruct LLMs to their correct results in new comments (see the third-last row), because of the absolutely different code-comment change patterns. In contrast, the expert-retrieved example follows a similar change pattern to the CCS target: the modification is induced by the change of return type, i.e., from the "ConnectionFactory" to "HornetQConnectionFactory", which is also reflected in its old and new comments. As such, expert retrieval offers a more instructive demonstration and makes LLMs generate the correct results (see the second-last row).

Case Study 3 (Fig. 10) is a CCS sample in Liu's dataset. In this case, LLMs generate four candidate results for the given CCS target. Yet, the *correct synchronization* is ranked second without the MR strategy, as shown in the lower-left corner of Fig. 10. Subsequently, we conduct the MR strategy on the generated candidates and list the re-ranking results in the lower-right corner of Fig. 10. Specifically, Candidates 1[3] and 4[4] both violate Rule 3 because their edit distances relative to the old comment exceed our predefined threshold. Since it is a severe mistake, they are ranked third and fourth. Towards Candidate 3, the token "valid" is not changed by the token "active", which means it does not sense the code change occurring in the function name. That is to say, Candidate 3 violates Rule 1. As Rule 1 is a weak mistake, it consequently ranks second. To this end, Candidate 2 is finally prioritized among other candidates and ranks first, as it does not violate any rule. In this way, the MR strategy takes effect in the re-ranking phase of our proposal.

Owing to the advantages of our proposed modules in R^{2}ComSync as illustrated above, R^{2}ComSync makes LLMs becomes competitive and even substantially exeling existing approaches. Case 4 is a typical example, shown in Fig. 11. The code changes in this CCS target modifies the setting value from "tableGetResultsId" to "resource", which is also precisely reflected in the comment changes. As can be seen, none of the existing approach achieves the *correct synchronization*. For example, heuristic-based HEBCUP only handled the replacement between "table" and "resource". Deep learning-based CUP struggles in how to deal with the relationship between "resource" and "db table". R^{2}ComSync retrieves

[3] Rule 3 (ϵ) of Candidate 1: ED_{cm}/N_{cm_s}(3/7)=0.429> ϵ=0.25

[4] Rule 3 (ϵ) of Candidate 4: ED_{cm}/N_{cm_s}(5/7)=0.714> ϵ=0.25

- A Code-Comment Synchronization (CCS) target -

ID: JetBrains_jdk8u_jaxp-34-6323

```
- public Hashtable getBuiltInTypes() {
+ public Map<String, DatatypeValidator> getBuiltInTypes() {
-     Hashtable toReturn = (Hashtable) fBuiltInTypes.clone();
+     final HashMap<String, DatatypeValidator> toReturn = new HashMap<>(fBuiltInTypes);
-     Enumeration xml11Keys = fXml11BuiltInTypes.keys();
-     while (xml11Keys.hasMoreElements()) {
-         Object key = xml11Keys.nextElement();
-         toReturn.put(key, fXml11BuiltInTypes.get(key));
-     }
+     toReturn.putAll(xml11BuiltInTypes);
      return toReturn;
  }
```

Old Comment: a ~~hashtable~~ which contains all datatypes.

Reference Comment: a map which contains all datatypes.

- CodeBERT Retrieval -

- A Retrieved Demonstration Example -

```
    public byte[] getContentAsByteArray() {
+       if (!this.contentAccessed) {
+           return new byte[0];
+       }
        flushBuffer();
        return this.content.toByteArray();
    }
```

Old Comment: the content as a byte array.

Reference Comment: the content as a byte array or empty array if there is no content.

- Expert Retrieval -

- A Retrieved Demonstration Example -

```
- public static ConnectionFactory createConnectionFactory(
+ public static HornetQConnectionFactory
  createConnectionFactory(
      final ClientSessionFactory sessionFactory){
      return new HornetQConnectionFactory(sessionFactory);
  }
```

Old Comment: the ~~connection~~ factory.

Reference Comment: the hornet qconnection factory.

- Outputs -

R²ComSync (CodeBERT)	an array which contains all data types.	✘
R²ComSync (Expert)	a map which contains all datatypes.	✓
R²ComSync (EHR)	a map which contains all built-in data types.	✘

Fig. 9 Case Study 2, Illustrating the Limitation of the EHR Method

similar demonstrations from code-comment semantics and change patterns for ICL, leading to its successful synchronization for this case.

7.3 Where Does R²ComSync Fail

Although R²ComSync exhibits powerful CCS ability, it has limitations and failures in some cases. This section carefully inspects a series of failed examples that R²ComSync cannot generate *correct synchronization*s and summarizes three main situations:

(1) Misleading demonstrations Case Study 5 (Fig. 12) presents an example where the training-based CUP succeeds while our method fails. R²ComSync produces a semantically incorrect new comment, i.e., the term “item” does not match the intended “iterator”. However, the error does not stem from the key code changes involving precision optimization from “double” to “BigDecimal”, but rather from the substitution of “iterator” with “item”. However, based on the retrieved demonstrations from the CCS target and CodeBERT or expert aspects, “item” has never appeared in the input context of R²ComSync. Hence, we

- A Code-Comment Synchronization (CCS) target -

Project: real-logic/aeron
Commit ID: d25dc2a91464a95397a0aa9c3fdd29d79e43bcfe

```
1  - public static boolean isValid(final CountersReader countersReader,
2  + public static boolean isActive(final CountersReader countersReader,
3  final int counterId, final long registrationId)
4      {
5          final DirectBuffer buffer = countersReader.metaDataBuffer();
6  +       final int recordOffset = CountersReader.metaDataOffset(counterId);
7  -       if (countersReader.getCounterState(counterId) == RECORD_ALLOCATED)
8  -       {
9  -           final int recordOffset = CountersReader.metaDataOffset(counterId);
10 -           return buffer.getLong(recordOffset + KEY_OFFSET + REGISTRATION_ID_OFFSET)
11 - == registrationId;
12 +           return buffer.getLong(recordOffset + KEY_OFFSET + REGISTRATION_ID_OFFSET)
13 == registrationId &&countersReader.getCounterState(counterId) == RECORD_ALLOCATED;
14 -       }
15 -       return false;
16     }
```

Old Comment: Is the counter ~~valid~~ for usage?

Reference Comment: Is the counter active for usage?

- Without MR Strategy -	**- With MR Strategy -**
- Outputs -	**- Outputs -**
Candidate 1: Is the counter active? ✘	**Candidate 2:** Is the counter active for usage? ✓
Candidate 2: Is the counter active for usage? ✓	**Candidate 3:** Is the counter valid for usage? ✘
Candidate 3: Is the counter valid for usage? ✘	**Candidate 1:** Is the counter active? ✘
Candidate 4: Is the counter active and may be used? ✘	**Candidate 4:** Is the counter active and may be used? ✘

Fig. 10 Case Study 3, Illustrating the Strength of the MR Strategy

speculate it is a hallucination based on LLMs' own internal distribution. The reason for this hallucination is potentially due to misleading demonstrations. As can be seen, although CodeBERT retrieval obtains a semantically similar example, its code-comment change patterns differ from those of the CCS target. As for the expert retrieval, it also does not offer proper examples for ICL. In other words, our prepared demonstration pools have no instructive examples from the two perspectives mentioned above. As such, R^{2}ComSync fails on this case.

In contrast, CUP, being directly fine-tuned on in-domain data, exhibits stronger conservatism toward domain-specific terminology. Although it is less flexible, its internal distribution aligns more closely with the target domain, preventing arbitrary hallucinations. It is conceivable that incorporating CUP-generated examples into the retrieval corpus could help LLMs focus more consistently on the same domain and thereby reduce hallucinations. While this would increase computational cost, it represents a promising direction for future research.

Besides, developers' intentions when updating comments sometimes are elusive. Especially, when a great chunk of code is changed, various updated information has the probability of being reflected in the new comments, making it hard to precisely capture the

- A Code-Comment Synchronization (CCS) target -

Project: Azure/azure-sdk-for-java
Commit ID: e3f61b302c5aae1d9aa594a12ba928b6645284c8

```
1 - public TableGetResultsInner withTableGetResultsId(String tableGetResultsId) {
2 + public TableGetResultsInner withResource(TableGetPropertiesResource resource) {
3 -     this.tableGetResultsId = tableGetResultsId;
4 +     this.resource = resource;
5       return this;
6   }
```

Old Comment: Set ~~name of the Cosmos DB table~~.

Reference Comment: Set the resource value.

- CodeBERT Retrieval -	**- Expert Retrieval -**
- A Retrieved Demonstration Example -	**- A Retrieved Demonstration Example -**

1 - public SiteConfigInner withAppSettings(List<NameValuePair> appSettings){
2 + public SiteConfigInner withMachineKey(SiteMachineKey machineKey) {
3 - this.appSettings = appSettings;
4 + this.machineKey = machineKey;
5 return this;
6 }
	1 - public Builder isCall(boolean isCall) {
	2 + public Builder iborFuture(ExpandedIborFuture iborFuture) {
	3 - this.isCall = isCall;
	4 + JodaBeanUtils.notNull(iborFuture, \"iborFuture\");
	5 + this.iborFuture = iborFuture;
	6 return this;
	7 }

Old Comment: Set the ~~appSettings~~ value.	**Old Comment:** Sets the {@code ~~isCall~~} property in the builder.
Reference Comment: Set the machineKey value.	**Reference Comment:** Sets the {@code iborFuture} property in the builder.

- Outputs -

CUP	set the resource db table.	✘
HEBCUP	set name of the cosmos db resource.	✘
HatCUP	set name of the db table.	✘
CBS	set the resource db table.	✘
Toper	set name of the cosmos db table .	✘
R^{2}ComSync	Set the resource value.	✓

Fig. 11 Case Study 4: Comparison between R^{2}ComSync and Baseline

developers' updating intention by representations of code changes and old comments. To this end, the CCS demonstrations retrieved by the above representations may mislead the targets, such as case (CCS 2023a).

(2) Specious candidates A specious candidate does not violate any rule we predefined, but is not identical to the reference comment. Typically, these candidates normally correctly update partial comments or make relatively tiny mistakes when synchronization. In this case, once its initial ranking is high, the heuristic-based MR strategy cannot detect it and re-rank its order. For example, when handling cases (CCS 2023b, c), R^{2}ComSync misses a Javadoc markup "@code" but correctly updates all other contents, leaving specious candidates.

(3) Misjudgments of automatic evaluation Automatic evaluation cannot identify synonyms or paraphrases when assessing the qualities of generated new comments. Thus, some cases are considered correct via human evaluation but are deemed flawed in automatic evaluation, such as case (CCS 2023d), where the generated new comment uses "is not" instead of "no" in its reference, causing the misjudgment.

- A Code-Comment Synchronization (CCS) target -

Project: BaseXdb/basex

Commit ID: 68b20408b9c627cb7608725ac09b514d8206f950

```
1 - protected static Item decimal(final double d) {
2 + protected static Item decimal(final int integer) {
3 -     return Dec.get(d);
4 +     return Dec.get(new BigDecimal(integer));
5     }
```

Old Comment: Creates an iterator for the specified ~~decimal~~.

Reference Comment: Creates an iterator for the specified integer.

- CodeBERT Retrieval -

- A Retrieved Demonstration Example -

```
1 - public static Iter iter(final double d) {
2 + public static Dec get(final double d) {
3 -   return iter(BigDecimal.valueOf(d));
4 +   return get(BigDecimal.valueOf(d));
5 }
```

Old Comment: ~~Returns an iterator~~.

Reference Comment: Constructor.

- Expert Retrieval -

- A Retrieved Demonstration Example -

```
1 - public final boolean isFree() {
2 + public boolean isFreed() {
3 -   return isFreed;
4 +   return this.memory == null;
5 }
```

Old Comment: If the argument is, ~~it is checked if any~~ variable ~~is used~~.

Reference Comment: If the argument is, all variables are counted.

- Outputs -

CUP	Creates an iterator for the specified integer.	✓
R²ComSync	Creates an item for the specified integer.	✘

Fig. 12 Case Study 5, A Sample which R^{2}ComSync Fail

7.4 Implications

This section introduces the implications of our work from the perspectives of researchers and practitioners, respectively.

Implications for researchers This work, for the first time, revealed the weakness of LLMs in handling the CCS task against existing SOTA approaches. Compared with recent work on the CCS task, such as Fan et al. (2024), our work introduces the motivation of further enhancing LLMs instead of only empirically discussing their capabilities. Besides, as the first attempt to unleash LLMs' ability in CCS, we design an EHR method to retrieve effective ICL demonstrations and propose novel re-ranking rules to prioritize correct-prone candidates. Although ICL and re-ranking are common methods in LLM-augmented approaches, we provide novel thoughts for the above methods in the domain of CCS research. For example, we combine code-comment semantics and change patterns for relevant demonstration retrieval, where the former can provide synchronization directions in a macro aspect, while the latter can offer detailed synchronization guidance on specific patterns (See Section 6.2 for details), making our retrieval method different from prior retrieval-augmented LLM approaches. On the other hand, we also summarize a series of re-ranking rules via large-scale analysis on CCS samples, further improving the CCS performance of R^{2}ComSync as shown in the Section 6.3. As such, the academic novelty of R^{2}ComSync lies more in its

specific and targeted design for the CCS task, rather than in applying other methods directly to this area. In addition, we also discuss a series of typical failures that R^{2}ComSync exhibited in our experiments (see Section 7.3 for details), thereby providing insights for future improvement and research directions.

Implications for practitioners This paper proposed R^{2}ComSync, an effective LLM-augmented tool to synchronize comments to their corresponding code snippets. To help practitioners understand its use in daily development and maintenance, we thoroughly investigated R^{2}ComSync on both real-world Java and Python projects to assess its effectiveness in Section 6.1. Besides, we also provided a series of actionable insights on the configuration of the EHR method and the MR strategy of R^{2}ComSync (See Sections 6.2 and 6.3). For example, we discussed the weight allocation of CodeBERT retrieval and expert retrieval in the EHR method for different data samples, empirically highlighting the superiority of change-pattern-aligned demonstrations in short CCS samples and their low dependency on data distribution consistency. Besides, we also examined appropriate settings for shot numbers when retrieving ICL demonstrations. In addition, the utility of each re-ranking rule and its hyper-parameter settings was also studied to offer practical suggestions. Moreover, in Section 7, we further explored the cost of R^{2}ComSync in practical usage and showcased both its successful and failed cases, intuitively illustrating its advantages and precautions for practitioners.

7.5 Threats to Validity

There are mainly four threats to the validity of this work.

Potential data leakage Data leakage occurs when testing samples are visible during model training, leading to an overestimation of model performance in experiments, which is also a type of data snooping problem. Our proposal is experimented on Llama3, GPT3.5, and Qwen series, which are LLMs trained on open-source code repositories that are not publicly disclosed. As such, there is a potential threat that they may have seen both before- and after-change code in our testing set during their pre-training, causing their memorization in certain CCS samples. Nonetheless, all studied LLMs were pre-trained on GitHub code files using Causal Language Modeling (CLM), which is entirely different from the inference pattern of the CCS task. This task involves using old and new code, along with old comments, as inputs, while designating new comments as outputs. This discrepancy can be categorized as an unpaired contamination, which has limited effects on the follow-up inference on the CCS task as revealed in Yang et al. (2025). Besides, No LLM performs well in the zero-shot setting compared to SOTA baselines (refer to Table 1), indicating that these LLMs have not memorized the testing samples. Instead, our proposed R^{2}ComSync significantly improves the performance of LLMs, demonstrating the effectiveness of our proposal. To this end, we believe this threat is minimal.

The generalizability of our experimental results In this work, we conducted experiments on three real-world CCS datasets across two programming languages, including the statically-typed Java and dynamically-typed Python, explicitly demonstrating the generality of R^{2}ComSync. Although our studied datasets are mainly on the function level, it is the most prevalent comment updating granularity in practical development (Liu et al. 2020; Lin et al.

2021; Zhu et al. 2022; Yang et al. 2023; Lin et al. 2023; Liu et al. 2023). Besides, we exhaustively select all state-of-the-art approaches in this field as our baselines for comparison and conduct both automatic and human evaluations for comprehensive assessments. Therefore, the threat of generalizability is limited.

CCS Trials on other LLMs In this paper, we conduct CCS experiments with Llama3-8/70B, GPT3.5, Qwen2.5, and Qwen2.5-Coder, yet there are other alternatives we have not experimented on, such as CodeGen (Nijkamp et al. 2022), InCoder (Fried et al. 2026), and DeepSeek series (Liu et al. 2024; Guo et al. 2024). Nevertheless, our selected LLMs contain both closed-source (e.g., GPT-3.5-Turbo) and open-source LLMs (e.g., Llama3); code (e.g., Qwen2.5-Coder-1.5B) and general (e.g., Qwen2.5-1.5B) LLMs; as well as large (e.g., Llama3-70B) and small LLMs (e.g., Qwen2.5-1.5B) of different families. Therefore, we believe the choice of studied LLMs is adequate and comprehensive to a great extent.

Limited experiments on hyper-parameter settings The hyper-parameters used in RComSync involve the shot number () and two thresholds in Rule 2 () and Rule 3 (). These hyper-parameters were selected empirically based on our usage experience with RComSync and validated within effective ranges, under which we achieved SOTA performance compared with existing baselines. However, such an empirical selection process may introduce potential risks, including mild overfitting to the specific datasets studied or sensitivity to dataset characteristics (e.g., programming language or code-change patterns). Although our experiments across multiple datasets and languages suggest that RComSync is relatively robust to these settings, other hyper-parameter combinations might yield better or more stable performance. Due to computational resource constraints, we leave a more systematic and broader exploration of hyper-parameters for future work.

8 Conclusion

This paper introduces the R^{2}ComSync approach, aiming to automate the synchronization of code and comments to reduce the workload of developers during software maintenance and evolution. Specifically, we propose a hybrid ensemble retrieval method to collect similar CCS examples, focusing on both the semantics of code and comments as well as the patterns of changes. This method provides precise guidance for the generation objectives of LLMs. To enhance the effectiveness of our approach, we also propose a multi-turn re-ranking strategy to prioritize correct-prone candidates generated by LLMs. Extensive experiments have demonstrated that R^{2}ComSync significantly lifts LLMs' capability in the CCS task and achieves the SOTA performance among diverse existing approaches. In the future, we will further explore more effective methodologies to advance in this area. For example, designating LLMs to summarize code changes as intermediate guidance for synchronizing old comments to their new versions.

Author Contributions Zhen Yang: Conceptualization, Methodology, Software, Formal analysis, Validation, Writing. Hongyi Lin: Software, Formal analysis, Investigation, Data curation, Writing. Xiao Yu: Conceptualization, Validation, Writing. Jacky Wai Keung: Supervision, Conceptualization, Validation. Shuo Liu: Software, Formal analysis, Investigation, Data curation, Writing. Pak Yuen Patrick Chan: Validation, Writing. Yicheng Sun: Validation, Writing. Fengji Zhang: Validation, Writing.

Funding This work was partially supported by the National Natural Science Foundation of China (Grant Nos. U24B20149, and 62502283), the Natural Science Foundation of Shandong Province (Grant No. ZR2024QF093), and the Young Talent of Lifting Engineering for Science and Technology in Shandong, China (Grant No. SDAST2025QTB031).

Data Availability We open-source the replication package of R^2ComSync for facilitating its application and future research, which is available at YDS0823/ComSync (2025).

Declarations

Conflict of Interests The authors declared that they have no conflict of interest.

Ethical Approval Not applicable.

Informed Consent All participants who joined the human evaluation were provided with an informed consent form outlining the study's objectives, procedures, potential risks, and benefits. Participation was voluntary, and respondents could withdraw at any time without consequences.

Clinical Trial Number in the manuscript Not applicable.

References

Aarts S, Van Den Akker M, Winkens B (2014) The importance of effect sizes. Eur J Gen Pract 20(1):61–64

CCS for function 'create' (2023a) https://github.com/mongodb/mongo-java-driver/commit/b7dfa1d397ce3cc0baad10e4c8da15a8a0f38c1e. (Accessed on 04/28/2023)

CCS for function 'ensureNoSplitBrain' (2023d) https://github.com/gridgentoo/hazelcast/commit/a759e820aa7af5e2a6dba1f1502bbcac75e1ff7f. (Accessed on 04/29/2023)

CCS for function 'getKeyFilter' (2023b) https://github.com/Azure/azure-sdk-for-java/commit/9f9a4550225e95d7957c5dc0e723e18fc720ce3e. (Accessed on 04/29/2023)

CCS for function 'previousTokenOnChannel' (2023c) https://github.com/ramjyroo/antlr4-jyroo/commit/e311ec63b44694f30923858cf77dbde7f327b89e. (Accessed on 04/29/2023)

Chen D, Qian K, Yu Z (2023) Stabilized in-context learning with pre-trained language models for few shot dialogue state tracking. In: Findings of the Association for Computational Linguistics: EACL 2023, pp 1551–1564

Chen B, Zhang F, Nguyen A, Zan D, Lin Z, Lou J-G, Chen W (2022) Codet: Code generation with generated tests. In: The Eleventh International Conference on Learning Representations

Chmura Kraemer H, Periyakoil VS, Noda A (2002) Kappa coefficients in medical research. Stat Med 21(14):2109–2129

Dong Q, Li L, Dai D, Zheng C, Ma J, Li R, Xia H, Xu J, Wu Z, Chang B, et al (2024) A survey on in-context learning. In: Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pp 1107–1128

Dubey A, Jauhri A, Pandey A, Kadian A, Al-Dahle A, Letman A, Mathur A, Schelten A, Yang A, Fan A, et al (2024) The llama 3 herd of models. arXiv:2407.21783

Fan L, Liu J, Liu Z, Lo D, Xia X, Li S (2024) Exploring the capabilities of llms for code change related tasks. ACM Trans Software Eng Methodol

Feng Z, Guo D, Tang D, Duan N, Feng X, Gong M, Shou L, Qin B, Liu T, Jiang D, Zhou M (2020) CodeBERT: A pre-trained model for programming and natural languages. In: Cohn T, He Y, Liu Y (eds) Findings of the Association for Computational Linguistics: EMNLP 2020, pp 1536–1547. Association for Computational Linguistics, Online. https://doi.org/10.18653/v1/2020.findings-emnlp.139 . https://aclanthology.org/2020.findings-emnlp.139/

Fried D, Aghajanyan A, Lin J, Wang S, Wallace E, Shi F, Zhong R, Yih S, Zettlemoyer L, Lewis M (2026) Incoder: A generative model for code infilling and synthesis. In: The Eleventh International Conference on Learning Representations

Gao Z, Xia X, Grundy J, Lo D, Li Y-F (2020) Generating question titles for stack overflow from mined code snippets. ACM Trans Softw Eng Methodol (TOSEM) 29(4):1–37

Gao S, Chen C, Xing Z, Ma Y, Song W, Lin S-W (2019) A neural model for method name generation from functional description. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 414–421. IEEE
GitHub (2023) https://github.com/. (Accessed on 04/14/2023)
Guo D, Zhu Q, Yang D, Xie Z, Dong K, Zhang W, Chen G, Bi X, Wu Y, Li Y, et al (2024) Deepseek-coder: When the large language model meets programming–the rise of code intelligence. arXiv:2401.14196
Holtzman A, Buys J, Du L, Forbes M, Choi Y (2026) The curious case of neural text degeneration. In: International Conference on Learning Representations
Hu X, Li G, Xia X, Lo D, Jin Z (2020) Deep code comment generation with hybrid lexical and syntactical information. Empir Softw Eng 25:2179–2217
Hugging Face (2025) – The AI community building the future. [Online; accessed 2025-10-14] . https://huggingface.co/
Hui B, Yang J, Cui Z, Yang J, Liu D, Zhang L, Liu T, Zhang J, Yu B, Lu K, et al (2024) Qwen2. 5-coder technical report. arXiv:2409.12186
Inala JP, Wang C, Yang M, Codas A, Encarnación M, Lahiri S, Musuvathi M, Gao J (2022) Fault-aware neural code rankers. Adv Neural Inf Process Syst 35:13419–13432
Islam MA, Jonnala DV, Rekhi R, Pokharel P, Cilamkoti S, Imran A, Kosar T, Turkkan B (2025) Evaluating the energy-efficiency of the code generated by llms. arXiv:2505.20324
Jain N, Vaidyanath S, Iyer A, Natarajan N, Parthasarathy S, Rajamani S, Sharma R (2022) Jigsaw: Large language models meet program synthesis. In: Proceedings of the 44th International Conference on Software Engineering, pp 1219–1231
Jiang X, Dong Y, Wang L, Fang Z, Shang Q, Li G, Jin Z, Jiao W (2024) Self-planning code generation with large language models. ACM Trans Software Eng Methodol 33(7):1–30
Kaplan J, McCandlish S, Henighan T, Brown TB, Chess B, Child R, Gray S, Radford A, Wu J, Amodei D (2020) Scaling laws for neural language models. arXiv:2001.08361
Kenton JDM-WC, Toutanova LK (2019) Bert: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of NAACL-HLT, pp 4171–4186
Keyes J (2002) Software Engineering Handbook. CRC Press, Boca Raton, FL
Li J, Zhao Y, Li Y, Li G, Jin Z (2024) Acecoder: An effective prompting technique specialized in code generation. ACM Trans Softw Eng Methodol 33(8):1–26
Lin B, Wang S, Liu Z, Xia X, Mao X (2023) Predictive comment updating with heuristics and ast-path-based neural learning: A two-phase approach. IEEE Trans Software Eng 49(4):1640–1660. https://doi.org/10.1109/TSE.2022.3185458
Lin B, Wang S, Liu K, Mao X, Bissyandé TF (2021) Automated comment update: How far are we? In: 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC), pp 36–46. IEEE
Liu P, Yuan W, Fu J, Jiang Z, Hayashi H, Neubig G (2023) Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. ACM Comput Surv 55(9):1–35
Liu Z, Xia X, Lo D, Yan M, Li S (2023) Just-in-time obsolete comment detection and update. IEEE Trans Software Eng 49(1):1–23. https://doi.org/10.1109/TSE.2021.3138909
Liu A, Feng B, Xue B, Wang B, Wu B, Lu C, Zhao C, Deng C, Zhang C, Ruan C, et al (2024) Deepseek-v3 technical report. arXiv:2412.19437
Liu Z, Xia X, Yan M, Li S (2020) Automating just-in-time comment updating. In: Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering, pp 585–597
Lu S, Duan N, Han H, Guo D, Hwang S-w, Svyatkovskiy A (2022) Reacc: A retrieval-augmented code completion framework. In: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp 6227–6240
Lu S, Guo D, Ren S, Huang J, Svyatkovskiy A, Blanco A, Clement C, Drain D, Jiang D, Tang D, et al (2026) Codexglue: A machine learning benchmark dataset for code understanding and generation. In: Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)
Min S, Lyu X, Holtzman A, Artetxe M, Lewis M, Hajishirzi H, Zettlemoyer L (2022) Rethinking the role of demonstrations: What makes in-context learning work? In: Conference on Empirical Methods in Natural Language Processing
Nashid N, Sintaha M, Mesbah A (2023) Retrieval-based prompt selection for code-related few-shot learning. In: Proceedings of the 45th International Conference on Software Engineering (ICSE'23)
Navarro G (2001) A guided tour to approximate string matching. ACM Comput Surv (CSUR) 33(1):31–88
Nijkamp E, Pang B, Hayashi H, Tu L, Wang H, Zhou Y, Savarese S, Xiong C (2022) Codegen: An open large language model for code with multi-turn program synthesis. arXiv:2203.13474
Ni C, Wang W, Yang K, Xia X, Liu K, Lo D (2022) The best of both worlds: integrating semantic features with expert features for defect prediction and localization. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp 672–683

OpenAI-CodeX (2023) https://platform.openai.com/docs/guides/code. (Accessed on 04/03/2023)
Pai K, Devanbu P, Ahmed T (2025) Codocbench: A dataset for code-documentation alignment in software maintenance. In: 2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR), pp 451–455. https://doi.org/10.1109/MSR66628.2025.00077
Panthaplackel S, Li JJ, Gligoric M, Mooney RJ (2021) Deep just-in-time inconsistency detection between comments and source code. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol 35, pp 427–435
Panthaplackel S, Nie P, Gligoric M, Li JJ, Mooney R (2020) Learning to update natural language comments based on code changes. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp 1853–1868
Pascarella L, Bruntink M, Bacchelli A (2019) Classifying code comments in java software systems. Empir Softw Eng 24(3):1499–1537
Qwen, Yang A, Yang B, Zhang B, Hui B, Zheng B, Yu B, Li C, Liu D, Huang F, Wei H, Lin H, Yang J, Tu J, Zhang J, Yang J, Yang J, Zhou J, Lin J, Dang K, Lu K, Bao K, Yang K, Yu L, Li M, Xue M, Zhang P, Zhu Q, Men R, Lin R, Li T, Tang T, Xia T, Ren X, Ren X, Fan Y, Su Y, Zhang Y, Wan Y, Liu Y, Cui Z, Zhang Z, Qiu Z (2025) Qwen2.5 Technical Report. 2412.15115
Rabbi F, Siddik MS (2020) Detecting code comment inconsistency using siamese recurrent network. In: Proceedings of the 28th International Conference on Program Comprehension, pp 371–375
Ram O, Levine Y, Dalmedigos I, Muhlgay D, Shashua A, Leyton-Brown K, Shoham Y (2023) In-context retrieval-augmented language models. Trans Ass Comput Linguist 11:1316–1331
Ratol IK, Robillard MP (2017) Detecting fragile comments. In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), pp 112–122. IEEE
See A, Liu PJ, Manning CD (2017) Get to the point: Summarization with pointer-generator networks. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp 1073–1083
Sridhara G, Hill E, Muppaneni D, Pollock L, Vijay-Shanker K (2010) Towards automatically generating summary comments for java methods. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, pp 43–52
Sun W, Miao Y, Li Y, Zhang H, Fang C, Liu Y, Deng G, Liu Y, Chen Z (2025) Source code summarization in the era of large language models. In: 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE), pp 1882–1894. IEEE
Tan L, Yuan D, Krishna G, Zhou Y (2007) /* icomment: Bugs or bad comments?*/. In: Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles, pp 145–158
Tan L, Yuan D, Zhou Y (2007) Hotcomments: how to make program comments more useful? In: HotOS, vol 7, pp 49–54
vLLM (2025) [Online; accessed 2025-10-18]. https://docs.vllm.ai/en/latest/
Wei J, Wang X, Schuurmans D, Bosma M, Xia F, Chi E, Le QV, Zhou D et al (2022) Chain-of-thought prompting elicits reasoning in large language models. Adv Neural Inf Process Syst 35:24824–24837
Wen F, Nagy C, Bavota G, Lanza M (2019) A large-scale empirical study on code-comment inconsistencies. In: 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), pp 53–64. IEEE
Wilcoxon F (1945) Individual comparisons by ranking methods. Biomet Bulletin 1(6):80–83
Xia CS, Wei Y, Zhang L (2023) Automated program repair in the era of large pre-trained language models. In: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), pp 1482–1494. IEEE
Xu FF, Alon U, Neubig G, Hellendoorn VJ (2022) A systematic evaluation of large language models of code. In: Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming, pp 1–10
Xu Z, Guo S, Wang Y, Chen R, Li H, Li X, Jiang H (2024) Code comment inconsistency detection based on confidence learning. IEEE Trans Software Eng 50(3):598–617
Xue P, Wu L, Yu Z, Jin Z, Yang Z, Li X, Yang Z, Tan Y (2024) Automated commit message generation with large language models: An empirical study and beyond. IEEE Trans Software Eng 50(12):3208–3224. https://doi.org/10.1109/TSE.2024.3478317
Xue P, Wu L, Yang Z, Li X, Yu Z, Jin Z, Li G, Xiao Y, Wu J (2024) Exploring and lifting the robustness of llm-powered automated program repair with metamorphic testing. arXiv:2410.07516
Xue P, Wu L, Yang Z, Wang C, Li X, Zhang Y, Li J, Jin R, Pei Y, Shen Z, et al (2025) Classeval-t: Evaluating large language models in class-level code translation. Proceed ACM Software Eng 2(ISSTA):1421–1444
Xue P, Zheng K, Yang Z, Pei Y, Wu L, Dong J, Luo X, Xiao Y, Liu F, Zhang Y, et al (2025) A new benchmark for evaluating code translation with third-party libraries. arXiv:2509.12087
Yang Z, Keung JW, Yu X, Xiao Y, Jin Z, Zhang J (2023) On the significance of category prediction for code-comment synchronization. ACM Trans Softw Eng Methodol 32(2) https://doi.org/10.1145/3534117
Yang Z, Lin H, He Y, Xu J, Sun Z, Liu S, Wang P, Yu Z, Liang Q (2025) Rethinking the effects of data contamination in code intelligence. arXiv:2506.02791

Yang Z, Liu F, Yu Z, Keung JW, Li J, Liu S, Hong Y, Ma X, Jin Z, Li G (2024) Exploring and unleashing the power of large language models in automated code translation. Proceed ACM Software Eng 1(FSE):1585–1608

Yang G, Zhou Y, Chen X, Zhang X, Han T, Chen T (2023) Exploitgen: Template-augmented exploit code generation based on codebert. J Syst Softw 197:111577

Yan B, Li K, Xu M, Dong Y, Zhang Y, Ren Z, Cheng X (2025) On protecting the data privacy of large language models (llms) and llm agents: A literature review. High-Confidence Comput, 100300

Yan L, Qin Z, Zhuang H, Jagerman R, Wang X, Bendersky M, Oosterhuis H (2024) Consolidating ranking and relevance predictions of large language models through post-processing. In: Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pp 410–423

Yao Y, Duan J, Xu K, Cai Y, Sun Z, Zhang Y (2024) A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. High-Confidence Comput 4(2):100211

YDS0823/ComSync: code comment synchronization. [Online; accessed 2025-03-24]. https://github.com/YDS0823/ComSync

Yu X, Liu L, Hu X, Keung JW, Liu J, Xia X (2024) Fight fire with fire: How much can we trust chatgpt on source code-related tasks? IEEE Trans Software Eng

Zhang Y (2024) Detecting code comment inconsistencies using llm and program analysis. In: Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering. FSE 2024, pp 683–685. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3663529.3664458

Zhang F, Chen B, Zhang Y, Keung J, Liu J, Zan D, Mao Y, Lou J-G, Chen W (2023) Repocoder: Repository-level code completion through iterative retrieval and generation. In: The 2023 Conference on Empirical Methods in Natural Language Processing

Zhang Y, Liu Z, Feng Y, Xu B (2024) Leveraging large language model to assist detecting rust code comment inconsistency. In: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering. ASE '24, pp 356–366. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3691620.3695010

Zhang T, Yu T, Hashimoto T, Lewis M, Yih W-t, Fried D, Wang S (223) Coder reviewer reranking for code generation. In: International Conference on Machine Learning, pp 41832–41846. PMLR

Zhang F, Yu X, Keung J, Li F, Xie Z, Yang Z, Ma C, Zhang Z (2022) Improving stack overflow question title generation with copying enhanced codebert model and bi-modal information. Inf Softw Technol 148:106922

Zhou X, Han D, Lo D (2021) Assessing generalizability of codebert. In: 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp 425–436. https://doi.org/10.1109/ICSME52107.2021.00044

Zhu H, He X, Xu L (2022) Hatcup: hybrid analysis and attention based just-in-time comment updating. In: Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension, pp 619–630

Authors and Affiliations

Zhen Yang[1] · Hongyi Lin[1] · Xiao Yu[2] · Jacky Wai Keung[3] · Shuo Liu[3] · Pak Yuen Patrick Chan[3] · Yicheng Sun[3] · Fengji Zhang[3]

✉ Shuo Liu
sliu273-c@my.cityu.edu.hk

Zhen Yang
zhenyang@sdu.edu.cn

Hongyi Lin
Hongyi.Lin@mail.sdu.edu.cn

Xiao Yu
xiao.yu@zju.edu.cn

Jacky Wai Keung
jacky.keung@cityu.edu.hk

Pak Yuen Patrick Chan
ppychan2-c@my.cityu.edu.hk

Yicheng Sun
yicsun2-c@my.cityu.edu.hk

Fengji Zhang
fengji.zhang@my.cityu.edu.hk

[1] School of Computer Science and Technology, Shandong University, 72 Binhai Rd, 266237 Qingdao, Shandong, China

[2] The State Key Laboratory of Blockchain and Data Security, Zhejiang University, 38 Zheda Rd, 310058 Hangzhou, Zhejiang, China

[3] Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, 999077 Hong Kong, China