# Large Language Models are Qualified Benchmark Builders: Rebuilding Pre-Training Datasets for Advancing Code Intelligence Tasks

Kang Yang*, Xinjun Mao*, Shangwen Wang*, Yanlin Wang†, Tanghaoran Zhang*,
Bo Lin*, Yihao Qin*, Zhang Zhang*, Yao Lu*, Kamal Al-Sabahi‡
* College of Computer, National University of Defense Technology, Changsha, China,
{yangkang, xjmao, wangshangwen13, zhangthr, linbo19, yihaoqin, zhangzhang14, luyao08 }@nudt.edu.cn
† Sun Yat-sen University, wangylin36@mail.sysu.edu.cn
‡ College of Banking and Financial Studies, Oman, kamal@cbfs.edu.om

*Abstract*—Pre-trained code models are essential for various code intelligence tasks. Yet, their effectiveness is heavily influenced by the quality of the pre-training dataset, particularly human-written reference comments, which usually serve as a bridge between the programming language and natural language. One significant challenge is that such comments could become inconsistent with the corresponding code as the software evolves, leading to suboptimal model performance. Large language models (LLMs) have demonstrated superior capabilities in generating high-quality code comments. This work investigates whether substituting original human-written comments with LLM-generated ones can improve pre-training datasets for more effective pre-trained code models. As existing reference-based metrics cannot evaluate the quality of human-written reference comments themselves, to enable direct comparison between LLM-generated and human reference comments, we introduce two auxiliary tasks as novel reference-free metrics, including code-comment inconsistency detection and semantic code search. Experimental results show that LLM-generated comments exhibit superior semantic consistency with the code compared to human-written reference comments. Our manual evaluation also corroborates this conclusion, which indicates the potential of utilizing LLMs to enhance the quality of the pre-training dataset. Based on this finding, we rebuilt the CodeSearchNet dataset with LLM-generated comments and re-pre-trained the CodeT5 model. Evaluations on multiple code intelligence tasks demonstrate that models pre-trained by LLM-enhanced data outperform their counterparts (pre-trained by original human reference comments data) on code summarization, code generation, and code translation tasks. This research validates the feasibility of rebuilding the pre-training dataset by LLMs to advance code intelligence tasks. It advocates rethinking the reliance on human reference comments for code-related tasks.

*Index Terms*—Code Summarization, Pre-Training, Large Language Models, Code Intelligence.

## I. INTRODUCTION

In the realm of code intelligence, pre-trained code models have significantly enhanced a spectrum of tasks such as code summarization [1], code generation [2]–[4], code search [5]–[7], clone detection [8] and automated bug fixing [9], [10].

Pre-trained source code models such as CodeBERT [11], GraphCodeBERT [12], CodeT5 [13] and UniXcoder [14] have achieved remarkable results on various software engineering tasks and even outperform large language models when fine-tuned with domain-specific data [15]. The comments of the corresponding code snippets serve as a crucial bridge between the programming language (PL) and natural language (NL), providing contextual understanding pivots that are vital for pre-training the above models. As such, the effectiveness of the pre-trained code models relies heavily on the quality of their pre-training datasets [6], which depend heavily on human reference comments.

However, this reliance on NL comments introduces a fundamental challenge: as software evolves, these comments often become mismatched with the code [16]–[18], leading to semantic inconsistencies between the code and comment. Previous research has highlighted such inconsistencies. Shi et al. [19] revealed that 41.9% of the code summarization dataset TLC [20] is noisy, with 22.8% of the comments being inconsistent with the corresponding code snippets. Consequently, inconsistent comments deteriorate the quality of the PL-NL dataset, which can degrade the training efficacy and performance of code models. Sun et al. [6] identified that more than one-third of the comments in CodeSearchNet (Java) [21] did not describe core functionalities. The model trained with noisy data faces severe performance degradation [6].

Recently, LLMs [22]–[24] have demonstrated superior generation capabilities in generating high-quality code comments [25]–[27]. In light of this, we are motivated to utilize LLMs to address the limitations faced by pre-trained code models. Specifically, we aim to answer the following question: *Can we rebuild the pre-training dataset by substituting the original human-written comments with LLM-generated ones for training more effective pre-trained code models?*

To that end, we first conduct a comprehensive evaluation to compare LLM-generated comments with human-written reference comments. To ensure the representativeness of the review, both open source (e.g., Code Llama [22] released by Meta, DeepSeek-Coder [28] developed by DeepSeek AI, and Star-

Coder2 [29] built by BigCode in collaboration with NVIDIA) and closed source LLMs (e.g., Text-davinci-003/GPT-3.5/GPT-4 released by OpenAI [23]) are considered in this work. A fundamental problem faced by our study is the lack of appropriate metrics. Specifically, traditional evaluation metrics for code summarization are reference-based, measuring the similarity between predicted and human-written reference comments. However, the underlying assumption of the reference-based evaluation is that reference comments are gold standard superior to other baselines [30]. A critical purpose of this work is to compare the quality of LLM-generated comments with human-written reference comments. Thus, we cannot directly apply off-the-shelf reference-based metrics to assess the quality of reference comments, which makes reference-free evaluation essential for our comparative comparison.

To address this challenge, we introduce two auxiliary tasks, code-comment inconsistency detection and semantic code search, to offer a more refined reference-free assessment of code comment quality. The inconsistency detection task aims to identify inconsistency between comment and code, and the semantic code search task assesses the ability to retrieve the correct code snippet using its comment as a query. Our intuition is that a higher-quality comment would exhibit better semantic consistency with the corresponding code so that (1) it is less likely to be detected as inconsistent by a well-trained classifier and (2) it is expected to facilitate accurate retrieval of the associated code from a database when used as a search query. The reference-free evaluation results show that ***comments generated by LLMs preserve better semantic consistency with the code than human reference comments***. In addition, we conduct a human evaluation to rate the LLMs-generated comments generated and human-written ones. The human evaluation results validate the effectiveness of two proposed reference-free metrics and confirm that comments generated by LLMs preserve better semantic consistency with code than the human-written reference comments. This insight forms the basis for reconstructing the training dataset.

Due to the data quality issues [6] in the widely used CodeSearchNet dataset, we rebuild it by substituting the human-written reference comments in CodeSearchNet [21] with LLM-generated ones. We further extend our research by re-pre-training the widely used model CodeT5 [13] with this rebuilt CodeSearchNet dataset and evaluating its performance across five downstream code intelligence tasks. The model trained with the LLM-rebuilt dataset exhibits superior performance in code intelligence tasks like code summarization, code generation, and code translation compared to the counterpart model trained with original datasets containing human reference comments. These findings affirm the potential of LLMs in improving the quality of training data for code intelligence tasks and underscore the need to reevaluate the longstanding reliance on human reference comments.

This research makes the following contributions: (1) Innovative Evaluation Metrics: To our knowledge, we are the first to leverage code comment inconsistency detection and semantic code search as reference-free evaluation metrics for assessing code comment quality. (2) Empirical Validation of LLMs: Empirical results demonstrate that LLM-generated comments preserve better semantic consistency with code than human-written references, validating the quality and reliability of LLM-generated comments in software engineering. (3) Exploration of LLMs in Dataset Construction: We demonstrate the efficacy of LLMs for creating high-quality code comment datasets, successfully generating over 2M comments using GPT-3.5-turbo to rebuild the CodeSearchNet dataset. This approach validates LLMs' potential for enhancing code intelligence training data.

## II. PRELIMINARIES

This section relates to the metrics measuring code comment quality (Section II-A), as assessing the quality of LLM-generated comments is essential in this work. Also, we explore the effectiveness of LLM as benchmark builders, so the NL-PL pre-trained models are discussed (Section II-B).

### A. Assessing the Quality of Code Comments

Quantitatively evaluating the quality of generated summaries is challenging. Typically, this involves comparing the model-generated summary to a reference summary. The similarity between the generated and reference summaries is then calculated as an indicator of quality. These metrics, known as reference-based, are classified into two main categories: n-gram overlap and semantic similarity [31], [32].

The dominant evaluation methods are traditional n-gram overlap-based metrics like BLEU [33], ROUGE [34] and METEOR [35], initially utilized in the machine translation community to measure the predicted translations' similarity to reference translations. They compute whether the same n-gram appear in the same order in both predictions and references.

The evaluation methods based on semantic similarity assess similarity in embedding space, providing "partial credit" for word matches, as termed by Wieting et al. [36]. The application of semantic similarity to code summary assessment is supported by growing evidence of the insufficiency of mere word overlap [37], [38]. Mahmud et al. recommend BERTScore for its effectiveness in capturing semantic similarities [38], while Haque et al. find that cosine similarity using Sentence-BERT and Universal Sentence Encoder representations closely align with human judgments [39], [40].

However, reference-based metrics rely on human-written comments, often mined from software repositories, which may not always be high quality as software evolves [18], [41]. Additionally, with LLMs capable of generating high-quality comments, reference-based metrics may struggle to capture nuances, particularly when assessing if generated comments are equal to or better than human references.

### B. NL-PL Pre-trained Models

A standard NL-PL pre-trained model first involves training a large-scale model on extensive unlabeled datasets using self-supervised objectives, then fine-tuning it for specific downstream tasks like code understanding and generation using

task-specific loss functions. This paradigm, initially introduced in NLP communities [11]–[13], has been proposed, featuring variations in architecture and pre-training tasks.

CuBERT [42] and CodeBERT [11] were among the first NL-PL models, with CodeBERT being the first large pre-trained model for multiple programming languages. Both use a multi-layer bidirectional Transformer architecture [43], similar to BERT [44] and RoBERTa [45]. CodeBERT is pre-trained on the bimodal CodeSearchNet dataset [21] with two objectives: Masked Language Modeling (MLM) [46]. GraphCodeBERT [12], which shares CodeBERT's architecture, adds structural code features by incorporating data flow graphs (DFG), replacing RTD with DFG-specific tasks such as predicting data flow edges and aligning nodes, leading to improved performance in several downstream tasks.

Encoder-only models require an additional decoder for generation tasks, limiting their ability to leverage pre-training for such tasks. Conversely, GPT-C [47] and CodeGPT [48] employ unidirectional language modelling, which works well for code generation but is suboptimal for understanding tasks. Recent work has explored encoder-decoder models for both understanding and generation tasks. PLBART [49], based on the BART architecture [50], is pre-trained on both NL and PL with denoising objectives. CodeT5 [13] modifies the T5 model [51] to include token type information for identifiers, supporting multi-task learning in downstream applications. UniXcoder [14] further extends these ideas, using a decoder-only approach for auto-regressive tasks like code completion and incorporating multi-modal inputs such as code comments and abstract syntax trees (AST) to enhance representation.

Instead of optimizing model architectures or designing domain-specific pre-training tasks, our study emphasizes the impact of pre-training data quality. We investigate how improvements in pre-training dataset quality can enhance the performance of pre-trained code models on downstream tasks.

## III. EVALUATION OF LLM-GENERATED COMMENTS

In this section, we comprehensively compare the LLM-generated comments and human-written reference comments.

### A. Objective and Research Questions

Traditional code comment evaluation relies on reference-based metrics using human-written comments as the gold standard, limiting direct quality comparisons between generated and human reference comments [30]. To address this limitation, we propose a paradigm shift towards reference-free evaluation using extrinsic auxiliary tasks: code-comment inconsistency detection and semantic code search. These tasks assess the semantic alignment between code and comments without requiring human references. In this section, our study addresses two key research questions:

**RQ1**: **How effective are the proposed reference-free metrics in assessing the quality of code comments?** We explore the effectiveness of our newly proposed evaluation metrics. It examines the alignment of reference-free metrics with reference-based metrics and human judges, laying the foundation for a comparative evaluation of LLM-generated and human reference comments.

**RQ2: How does the quality of comments generated by LLMs compare to human-written reference comments?** We examine whether LLM-generated comments can match or exceed the quality of human-written references, which is central to determining the practicality of employing LLMs to enhance training datasets for code intelligence tasks.

### B. Reference-free Evaluation Metrics

*1) Code Comment Inconsistency Detection:* Code comment inconsistency detection (CCID) determines whether a comment is semantically misaligned with the corresponding code snippet [18]. Since CCID is of immense practical use to software developers who have a vested interest in keeping their code bases easily readable, navigable, and as bug-free as possible, prior works proposed kinds of approaches for detecting inconsistencies [52], [53].
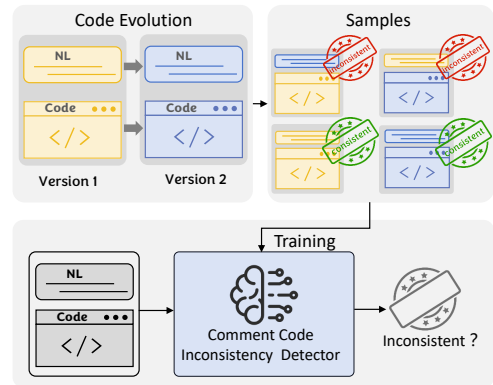


Fig. 1: Code comment inconsistency detection as a reference-free evaluation metric.

We propose the CCID task as a quantitative measure of comment quality. This approach stems from the principle that high-quality comments should accurately reflect their corresponding code. Inconsistent comments can cause confusion, errors, or misinterpretations, reducing effectiveness. The CCID task identifies discrepancies between code and comments, serving as an objective metric for comment quality and providing insights into the effectiveness of software documentation. We evaluate comment quality by measuring inconsistency rates in test datasets. The CCID task contains two distinct settings, post-hoc and just-in-time. In our work, we adopt the post-hoc setting because the comment/code pairs are available, and there are no code changes in the code summarization situation.

The data [54] we used to train the CCID classifier is curated by Panthaplackel et al. [52], which includes 40,688 samples of @return, @param, and summary Javadoc comments paired with their corresponding Java code methods. They consider comment-code pairs from each version of consecutive commits: $(c_1, nl_1), (c_2, nl_2)$. We collect examples that code changes do exist between two versions in which $c_1 \neq c_2$. As a result of code changes, the developer updated the comment

because it would have otherwise become inconsistent. Therefore, if $nl_1 \neq nl_2$, we take $nl_1$ comment to be inconsistent with $c_2$ code. As illustrated in Figure 1, $(c_1, nl_2)$ and $(c_2, nl_1)$ are consequently constitute the inconsistent examples. In contrast, if $nl_1 = nl_2$, the collected examples (i.e. $(c_1, nl_1)$ and $(c_2, nl_2)$) are labelled as consistent examples. The assumption is that the developer chose not to update the comment while modifying the code, as the comment was still consistent with the changes [52]. Figure 1 demonstrates the data constructing and training process of the CCID classifier.

Specifically, we utilize the state-of-the-art approach in the post-hoc setting proposed by Steiner et al. [53] as a classification model. The tokenized code $C$ and the $NL$ are concatenated into one sequence as input ([CLS] $C$ *tokens* [SEP] $NL$ *tokens* [SEP]), where [CLS] is the classification token and [SEP] is the separation token. The model outputs a binary label indicating whether the code-comment pair is inconsistent. Following the replication package [55], the model achieved 87.21% accuracy, 92.43% precision, 80.69% recall, and 86.16% F1 score on the test data curated by Panthaplackel et al. [52], and we utilize this model as a well-trained CCID classifier in this study.

Suppose $f$ denotes the well-trained CCID classifier, and the input of $f$ is a code snippet $c$ and its corresponding comment $nl$. The output $f(c, nl) = 1$ indicates they are semantically inconsistent; otherwise, $f(c, nl) = 0$. Assume $< C, NL >$ is the code snippets, its paired model-generated/human-written comments in test datasets, and the total examples in test datasets is $N$. We define the inconsistency rate (*IncRate*) by calculating the proportion of inconsistent examples.

$$IncRate = \frac{1}{N} \sum_{i=0}^{N} f(c_i, nl_i) \tag{1}$$

We propose to use *IncRate* as a reference-free metric to evaluate the quality of comments, and the lower *IncRate* indicates better semantic consistency between code and comments.

*2) Semantic Code Search:* The semantic code search task is to retrieve a code snippet that matches a given query by effectively capturing the semantic similarity between the query and code. Semantic code search is a vital software development assistant significantly improving development efficiency and quality [5].

Our motivation is rooted in the premise that the code search task could indicate the degree to which comments are aligned with their corresponding code. We assume that high-quality comments should not only elucidate code functionality but also enhance the discoverability of code snippets through semantic code search. This assumption aligns with the practical use case of developers who regularly rely on comments to navigate and understand large code bases. Additionally, code comments are an alternative to the practical queries in research communities [6], [21]. Consequently, the effectiveness of comments in facilitating accurate and efficient code search results serves as a proxy for their quality, providing a concrete, measurable dimension to an otherwise subjective attribute of software

documentation. Therefore, we argue comments that are more helpful for code search tasks are likely to be higher quality. Our study employs comments as queries within a code search task to assess the comment quality.
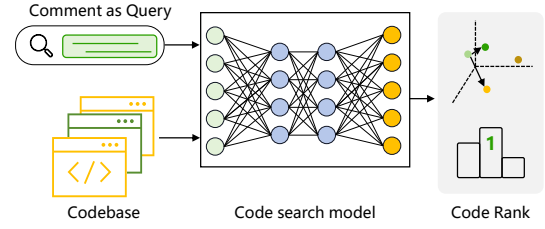


Fig. 2: Code search as a reference-free evaluation metric.

To evaluate the performance of code search, we use the Mean of Reciprocal Rank (MRR) [5], [56], [57], which has been widely adopted in the evaluation of semantic code search. The MRR score quantifies the ranking of the target code snippet to the given comment query, and it only cares about where the most relevant result is ranked. We use CodeBERT [11] for code search task and follow the configuration reported in their artifacts [58] to construct data and fine-tune the model. When computing MRR scores in the testing set, for each query-code pair, $|Q| - 1$ code snippets from other pairs in the same batch play the role of distractor codes, where $|Q|$ is the batch size, and we set as 1000 in this work. For a single batch, MRR is calculated as follows:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{Rank(\tilde{c}_i, nl_i)} \tag{2}$$

where $\tilde{c}_i$ is the ground-truth code snippet for its paired comment query $nl_i$, and $Rank(\tilde{c}_i, nl_i)$ is its corresponding rank in the retrieved results. MRR gives a score of the predicted result based on its rank. The average value of all batches is the final MRR score.

We propose to use MRR as a reference-free metric to evaluate the quality of comments. A higher MRR indicates better semantic consistency between the code and the comments.

*C. Experiment Design*

*1) Dataset:* We conduct experiments on a widely used Java benchmark dataset TLC [20], which has 66k code-comment pairs collected from more than 9K open-source Java projects created from 2015 to 2016 with at least 20 stars. They first extracted Java methods and Javadocs and treated the first sentence of the Javadoc as the ground-truth comment of the corresponding code. We use the TLC dataset open-sourced by the previous work [19]. The training/validation/test set contains 53,528/7,555/4,985 samples, respectively.

*2) Baselines:* Our research aims to investigate the effectiveness of two proposed reference-free metrics, comprehensively analyze comment quality, and compare LLMs' performance with human-written comments. Three deep learning (DL) based code summarization models are introduced as baselines to provide a meaningful context for evaluation. Six LLMs,

encompassing both open-source and closed-source models, are examined in this work to enhance the representativeness of the LLMs. This comparison will facilitate a more nuanced understanding of the performance differences in DL model-generated, LLMs-generated, and human-written comments.

**NCS** [59] is a transformer-based model that utilizes relative distances in self-attention and includes a copy mechanism to handle rare tokens from source code. **SIT** [60] integrates AST structure features into self-attention by combining the AST tree, data-flow graph, and control-flow graph into a multi-view graph matrix to refine token connections. **DOME** [61] leverages intent-guided selective attention to extract intent-relevant details, generating comments tailored to various intents. We use consistent data splits and default training parameters as in the original studies.

In this work, three types of models published by OpenAI are considered. **Text-davinci-003**. The text-davinci-003 [62] is a model trained using reinforcement learning with rewards from human comparisons. It performs well in consistent instruction following and longer output when completing any language task. **GPT-3.5 Turbo**. The capable and most cost-effective model (costs 1/10th of text-davinci-003) in the GPT-3.5 family can understand and generate natural language and code. The snapshot of GPT-3.5-turbo from Jan 25th, 2024, gpt-3.5-turbo-0125 [63] is used in this work. **GPT-4 Turbo** can solve complex problems with greater accuracy than any previous models of OpenAI, thanks to its broader general knowledge and advanced reasoning capabilities. The snapshot gpt-4-0125-preview [64] is utilized in this work.

The development process of LLMs varies in openness. Proprietary models, like OpenAI's GPT-4, are accessible via paid APIs, while their development specifics remain undisclosed. In contrast, open-source LLMs release model weights, allowing the community to run, inspect, and fine-tune these models. We consider three open-source LLMs: **Code Llama** by Meta AI, based on Llama2 [65], provides state-of-the-art performance, extensive input context handling, and zero-shot instruction-following for programming tasks. This study uses the 70B-parameter Code-Llama model [66]. **DeepSeek Coder** [67] is a range of open-source code models trained on 2 trillion tokens from 87 programming languages, with a 16K context length for complex tasks. We employ the largest model, DeepSeekCoder-33B. **StarCoder2** [68] by BigCode includes models with 3B, 7B, and 15B parameters trained on The Stack v2 dataset. All models and training resources are fully open-source.

To obtain summaries written by LLMs, we design a simple prompt with the format:

```
You are an expert [PL] programmer.
For the given [PL] method, please write a
one-sentence description as comment:
[Code Snippet Content]
```

We collected summaries from LLMs using this prompt for 4,985 Java methods in the test set. As LLM tends to give much longer comments than human-written references, we set

TABLE I: Statistics of unique words occurred in baseline-generated comments on TLC test set.

| Methods | Avg.words | Unique words |
|---|---|---|
| NCS | 8.72 | 3,485 |
| DOME | 8.91 | 3,924 |
| SIT | 8.68 | 4,034 |
| Human References | 11.76 | 5,854 |
| StarCoder2-15B | 13.80 | 5,420 |
| Text-davinci-003 | 13.49 | 6,392 |
| CodeLlama-70B | 14.18 | 5,931 |
| DeepSeekCoder-33B | 15.56 | 6,742 |
| GPT-3.5-Turbo | 15.22 | 6,721 |
| GPT-4-Turbo | 16.44 | 7,719 |

the parameter *max_tokens=30* to meet the same length level as human references for a fair comparison. Table I displays the word length statistics and unique words for LLMs-generated and reference comments. As for the sampling temperature parameters, we set the $top\_p = 1$ and the $tempreture = 1$.

*3) Evaluation Metrics:* We must also report commonly used traditional metrics in research communities for comparison as we propose new evaluation metrics. We categorize the evaluation metrics into two main groups based on whether references are needed as a standard: (1) reference-free metrics (IncRate and MRR) and (2) reference-based metrics (BLEU, ROUGE, METEOR and USE).

**IncRate** is a reference-free metric for code summarization. As introduced in the former subsection III-B1, IncRate evaluates the quality by measuring the percentage of inconsistent comment-code pair examples in the test set.

**MRR** is used initially to evaluate information retrieval. As detailed in former subsection III-B2, we adopt it to measure the semantic relationship between the comment query and its paired code snippet. Higher MRR means a better quality of comment in semantic coherence.

**BLEU** [33] is a textual similarity metric that calculates the precision of n-grams in a translated sentence compared to a reference sentence, with a weighted brevity penalty to punish short translations. We use the standard BLEU score, which provides a cumulative score of uni-, bi-, tri-, and quat-grams.

**ROUGE** [34] is a popular automatic evaluation metric that is recall-oriented. It computes the count of several overlapping units such as n-grams, word pairs, and sequences. ROUGE has several different variants, which we consider the ROUGE-L.

**METEOR** [35] is a metric based on the general concept of unigram matching, and it combines precision, recall, and a custom score determining the degree to which words are ordered correctly in the translation.

**USE** [32] is a metric that encodes the reference and the predicted summary to a fixed-length vector using a universal encoder and computes the similarity scores between two summaries. In this work, our experiments use the *pre-trained universal-sentence-encoder-large* model [69].

*D. Automatic Evaluation Results Analysis*

*1) reference-based metrics:* From the first three rows of Table II, the performance of NCS, DOME, and SIT models

TABLE II: Evaluation of DL-based methods generated, LLMs-generated comments and human references comments.

| Methods | reference-based metrics | | | | reference-free metrics | | human-evaluation Avg. (Std.) | | | |
| | BLEU | ROUGE | METEOR | USE | InRate ↓ | MRR | Naturalness | Consistency | Usefulness | Average |
|---|---|---|---|---|---|---|---|---|---|---|
| NCS | 21.55 | 35.98 | 15.08 | 0.5198 | 31.84% | 0.5614 | 3.31 (0.66) | 3.19 (0.52) | 3.36 (0.71) | 3.29 |
| DOME | 22.20 | 36.67 | **16.47** | 0.5460 | 24.99% | 0.6296 | 3.27 (0.51) | 3.23 (0.74) | 3.47 (0.65) | 3.32 |
| SIT | **22.54** | **37.87** | 16.11 | 0.5634 | 23.65% | 0.6404 | 3.35 (0.73) | 3.28 (0.61) | 3.44 (0.69) | 3.36 |
| Human References | - | - | - | - | 15.05% | 0.8165 | 3.41 (0.82) | 3.35 (0.73) | 3.67 (0.91) | 3.48 |
| StarCoder2-15B | 14.59 | 32.29 | 15.68 | 0.6297 | 3.95% | 0.8750 | 3.59 (0.78) | 3.44 (0.80) | 3.70 (0.86) | 3.58 |
| Text-Davinci-003 | 17.49 | 34.69 | 15.38 | 0.6234 | 3.65% | 0.8826 | 3.73 (0.85) | 3.57 (0.91) | 3.77 (0.81) | 3.69 |
| CodeLlama-70B | 14.73 | 32.51 | 16.19 | 0.6353 | 2.91% | 0.9007 | 3.66 (0.83) | 3.70 (0.68) | 3.79 (0.92) | 3.72 |
| DeepSeekCoder-33B | 12.39 | 29.71 | 16.06 | 0.6242 | 2.31% | 0.9214 | 3.95 (0.75) | 3.79 (0.75) | 3.80 (0.83) | 3.85 |
| GPT-3.5-turbo | 12.91 | 30.56 | 16.45 | **0.6505** | 1.30% | 0.9562 | 3.98 (0.71) | 3.96 (0.88) | 3.95 (0.81) | 3.96 |
| GPT-4.0-turbo | 11.24 | 28.61 | 16.63 | 0.6434 | **0.52%** | **0.9679** | **4.10** (0.67) | **4.02** (0.68) | **3.96** (0.95) | **4.03** |

appears closely matched, with no more than 2% difference in BLEU, ROUGE, and METEOR scores. Such marginal disparities challenge the intuitive differentiation of model efficacy, and less than 2 points improvements of overlap-based metrics do not guarantee systematic improvements in summarization quality [70]. When measured using reference-based evaluation metrics, the USE reveals a more significant discrepancy among the three DL-based models than BLEU, ROUGE, and METEOR. This phenomenon highlights the limitations of traditional n-gram overlap-based metrics in capturing subtle semantic differences.

The traditional n-gram overlap-based metrics merely measure the literal proximity of predicted comments to reference comments. Yet, many words have close synonyms and certain words within a sentence carry more weight than others [32]. While comments generated by three DL-based models may seem similar on a superficial literal level, the USE metric, which assesses semantic similarity through embedding vectors, provides a nuanced ability to distinguish the quality differences between these DL models' generated comments.

*Finding 1:* Compared to n-gram overlap metrics, the USE metric demonstrates superior nuanced ability in capturing quality differences of DL baselines generated comments.

2) *reference-free metrics:* Among three DL-based baselines, the SIT-generated comments exhibit the lowest IncRate, 23.65%, DOME performs the second, 24.99%, and NCS-generated comments get the highest inconsistency rate, 31.84%, with their corresponding code snippets. Simultaneously, SIT-generated comments achieve the highest MRR scores, 0.6404, while DOME ranks second, 0.6296, and CSN performs the lowest MRR score, 0.5614. Our proposed reference-free metrics, IncRate and MRR scores, align well with the four traditional reference-based metrics when evaluating three DL-based models.

Furthermore, our proposed two reference-free metrics can reveal more substantial performance differences among NCS, DOME, and SIT than reference-based metrics. While n-gram-based metrics show less than a 2% difference and USE metrics less than 5%, the IncRate and MRR metrics demonstrate a wider gap of 8%. It highlights the effectiveness of IncRate and MRR in detecting subtle but crucial semantic differences in comment quality across the DL-based models.

*Finding 2:* The reference-free metrics, IncRate and MRR, align well with reference-based metrics and demonstrate greater sensitivity in detecting performance differences in measuring DL baselines.

3) *LLMs models vs DL models:* In comparing LLMs with three DL-based baseline models, all six LLMs fall behind in BLEU, ROUGE, and METEOR scores but surpass all three DL baselines in the USE metric. This discrepancy indicates that relying solely on reference-based metrics could yield unreliable or contradictory assessments. Notably, as shown in Table I, most LLMs (5/6) exhibit a richer vocabulary in their comments than human written reference comments. The DL-based models share the same vocabulary list with human references. It suggests that LLMs-generated comments preserve more semantic diversity in word/token selection, as they are usually pre-trained in a massive corpus. Therefore, the large vocabulary underscores the importance of semantic measurement over literal overlap for a more comprehensive evaluation of LLM's capabilities.

Based on the USE metric, all six LLM-generated comments show better likeness with human references than three DL-based baselines. According to our reference-free metrics, six LLMs achieve lower IncRate and higher MMR scores than all three DL-based baselines, which aligns well with the conclusion obtained by the USE metric. These results validate the effectiveness of IncRate and MRR in measuring the semantic likeness between code snippets and comments.

*Finding 3:* The comments generated by LLMs show better semantic likeness with human references than DL baselines.

4) *LLMs vs human references:* Reference-based evaluation metrics cannot evaluate the quality of the human reference comments themselves. Thus, they can not be used to compare the quality of comments on LLMs and human references. In contrast, our proposed metrics, IncRate and MRR, evaluate comment quality independently of references, which provide an objective basis for assessing comment quality across code summarization models, LLMs, and human developers.

As shown in Table II, 15.05% of human reference comments are detected as semantically inconsistent by the inconsistency detection classifier, suggesting that non-standard practices of co-evolving and maintaining comments with code often exist

in software development and evolution. In comparison, the IncRate for six LLM-generated comments is markedly lower, no more than 4%. GPT-4-Turbo achieves the lowest 0.52% inconsistency rate. Additionally, in the code search task where comments are served as queries, all six LLM-generated comments surpass human references. GPT-3.5-Turbo and GPT-4-Turbo achieve MRR scores of 0.9562 and 0.9679, outperforming the human reference MRR score of 0.8165.

> *Finding 4:* LLM-generated comments preserve better semantic consistency than human reference comments.

### E. Human Evaluation

The proposed two reference-free metrics can measure the semantic consistency and semantic similarity between code snippets and comments without relying on human-written references, but it's not clear how they align with human judgment. To further validate the effectiveness of IncRate and MRR metrics, we perform a human evaluation to assess the quality of human reference and baseline-generated comments.

To evaluate the large number of instances across six LLMs and human references, we adopt a sampling method [71] to ensure a reliable confidence interval. The minimum number of cases required is calculated using $MIN = n_0/(1 + (n_0 - 1)/N)$, where $N$ is the total number of test cases, and $n_0 = (Z^2 \times 0.5)/e^2$, with $Z$ as the z-score for the desired confidence level and $e = 0.05$ at a 95% confidence level. Using this method, we selected 357 samples from a total of 4985 cases. For each code snippet, its ten corresponding comments (three DL-model-generated, one human reference and six LLM-generated comments) are evaluated. We recruited 36 participants (24 Undergraduate, 8 Master's, and 6 PhD students) with over three years of programming experience in Software Engineering or Computer Science to manually assess the quality of these comments.

Following the previous study [27], [72], [73], participants rated each comment based on three aspects: (1) **Naturalness** reflects the fluency of comments from the grammar perspective. (2) **Consistency** reflects the degree to which the semantics of comments match the code. (3) **Usefulness** reflects the practical usage value for developers and how comments can help them. Scores range from 1 to 5 (1 for poor, 2 for marginal, 3 for acceptable, 4 for good, and 5 for excellent). For each code snippet, participants evaluated a reference comment, three comments generated by DL-based approaches, and six by LLMs. To ensure fairness, all ten comments were anonymized, and each participant completed the questionnaire independently. To avoid subjective bias, each comment was evaluated by two participants, and their average score was used. If their scores differed by two or more points, a third participant's evaluation was introduced to solve the conflict and determine the final score, as we consider a difference of more than 2 points to be an enormous disagreement.

Human evaluation results are presented in the third column of Table II. The manual rating scores of human reference comments in Naturalness, Consistency and Usefulness are 3.41, 3.35 and 3.67, respectively, surpassing the three DL baseline models but scoring lower than all six LLM-generated comments in three aspects. The GPT-4-Turbo achieves the highest scores from participants, 4.10, 4.02 and 3.96 across these three aspects. In addition, the average score of humans rated in naturalness, consistency and usefulness is shown in the last sub-column. Evaluation results on two reference-free metrics aligned well with the overall quality level of manual evaluation. We thus answer the first two RQs (**RQ1&RQ2**).

> **Ans. to RQ1: effectiveness of reference-free metrics**
> Our proposed reference-free metrics, IncRate and MRR, demonstrate a strong correlation with both four traditional reference-based metrics and human evaluations, indicating their effectiveness in assessing code comment quality.

> **Ans. to RQ2: LLMs-generated vs. reference comments**
> The comments generated by LLMs exhibit higher quality than human reference comments, which preserve lower inconsistency and higher semantic relevance to the corresponding code. Human evaluations confirm that LLM-generated comments demonstrate superior naturalness, consistency and usefulness.

## IV. DISTILLING LLM FOR CODE INTELLIGENCES

In pre-trained source code models, the quality of training data is crucial for model performance. PL-NL paired data helps bridge the semantic gap between programming and natural language, providing context and descriptive insight that facilitate code-intelligence tasks. Section III reveals that LLM-generated comments exhibit higher semantic relevance and lower inconsistency with the corresponding code snippets than human reference comments. These findings suggest that incorporating LLM-generated comments could enhance training data quality, motivating the reconstruction of pre-training datasets for code intelligence. Thus, we ask the following research question: **RQ3: How does the pre-training data rebuilt by LLM impact the performance of the downstream code intelligence tasks?**

### A. Experiment Design

To assess the impact of LLM-rebuilt data on code intelligence tasks, we conduct the following experiments: First, we reconstruct a popular dataset by replacing human-written comments with LLM-generated ones to improve semantic consistency between NL comments and PL code. We then use this rebuilt dataset to pre-train a widely used source code model, fine-tune it on downstream tasks, and evaluate its performance. Finally, we compare the results of models trained on the rebuilt versus the original dataset, quantitatively and qualitatively.

*1) Dataset and Experimental Settings:* As the empirical results in Section III, the comments generated by GPT-4-turbo exhibit the best quality among six LLMs. The performance of GPT-4-turbo and GPT-3.5-turbo shows a minor difference in metrics, while the expenses of GPT-4-turbo are ten times

TABLE III: Statistics of cgpt-CSN dataset.

| PLs | W/ ChatGPT-NL | W/o NL |
|---|---|---|
| Ruby | 53,269 | 110,551 |
| JavaScript | 138,577 | 1,717,933 |
| Go | 346,333 | 379,103 |
| Python | 457,429 | 657,030 |
| Java | 496,651 | 1,070,271 |
| PHP | 578,072 | 398,058 |
| Total | 2,070,331 | 4,332,946 |

more expensive than GPT-3.5-turbo. Due to consideration of OpenAI API costs, we rebuild the CodeSearchNet [21] by replacing the human reference comments with GPT-3.5-turbo generated ones, noted as cgpt-CSN. The prompt we used is shown in Section III-C2. We did not incorporate the C/CSharp dataset added in the original CodeT5 [13] for the sake of training computation costs. We utilized approximately 2.07 million paired PL-NL instances, encompassing six PLs: Java, Python, PHP, Javascript, Go, and Ruby. The total OpenAI API costs approximately $2,000, and the rebuilt CodeSearchNet cgpt-CSN statistics are displayed in Table III.

In this study, we choose CodeT5 [13], a widely-used pre-trained model for source code based on an encoder-decoder framework similar to T5 [51], as our pre-training model for downstream code intelligence tasks. We follow Feng et al. [11] to employ cgpt-CSN to pre-train CodeT5, consisting of six PLs with unimodal and bimodal data. It is trained with four pre-training tasks, including a Masked Span Prediction (MSP) task, Identifier Tagging (IT), Masked Identifier Prediction (MIP), and Bimodal Dual Generation (BDG).

As the pre-training implementation is not available, we re-implement the CodeT5 pre-training process based on Huggingface's T5 [51] PyTorch implementation, and the size of the model is 220M, same as CodeT5-base. We set the maximum source and target sequence lengths to be 512. We use the mixed precision of FP16 to accelerate the pre-training. We set the batch size to 48 and employed the peak learning rate 2e-5 with linear decay. Following the settings in CodeT5 [13], we pre-train the model with the denoising objective for 100 epochs and bimodal dual training for a further 50 epochs. In the fine-tuning phase, we follow their default settings for the hyperparameters in the CodeXGLUE [48] tasks, such as learning rate, training steps, and batch size.

*2) Code Intelligence Tasks and Metrics:* For the code intelligence tasks, we cover four generation and one understanding tasks in the CodeXGLUE benchmark [48] and employ the provided public datasets and the same data splits following it for all these tasks. We first consider two cross-modal, text-to-code and code-to-text generation tasks, two code-to-code generation tasks, and one code understanding task.

**Code summarization** aims to summarize a function-level code snippet into natural language descriptions. The dataset comprises six PLs, including Ruby, JavaScript, Go, Python, Java, and PHP from CodeSearchNet [21]. Empirical findings in Section III show that the USE metric aligns well with human evaluation. To provide a rich perspective view, we employ one reference-based metric USE [32] and one of our newly proposed reference-free metrics, MRR, to evaluate code summarization. Note that the reference-based metric USE refers to the comment generated by GPT-3.5-Turbo, as it provides better consistency with code.

**Code generation** is an NL-PL task that generates a code snippet based on NL descriptions. We employ the Concode dataset [74] in Java, where the input contains both NL texts and class contexts, and the output is a Java function. We evaluate it with CodeBLEU [75], BLEU-4, and exact match (EM) accuracy that considers syntactic and semantic matches based on the code structure in addition to the n-gram match.
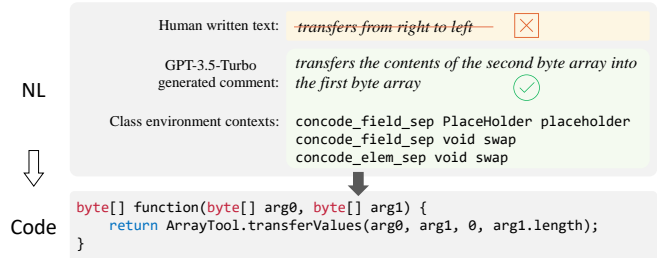


Fig. 3: A Concode data sample rebuilt by GPT-3.5-Turbo.

**Code translation** aims to migrate legacy software from one PL to another, where CodeXGLUE focuses on translating functions from Java to CSharp and vice versa. Because we omit C/CSharp in our updated version of the pre-training dataset, cgpt-CSN, we use the AVATAR [76], a parallel corpus for Java-Python program translation, for code translation tasks.

**Code refinement** is to detect which parts of code are buggy and fix them by generating a correct code sequence. We employ two Java datasets provided by Tufano et al. [77] with various function lengths: small (fewer than 50 tokens) and medium (50-100 tokens). Due to the limited edit refining, there is a large token overlap between the source and target code. The EM measurement could better reflect the correctness of the refinement generation. We report EM and BLEU-4.

**Clone detection** aims to measure the similarity between two code snippets and predict whether they have the same functionality. We experiment with the Java data provided by Wang et al. [78]. We employ Precision, Recall, and F1 scores for evaluating code clone detection.

*B. Results and Analysis*

In this section, we evaluate the re-pre-trained CodeT5 on downstream tasks. For better illustration, cgpt-CSN denotes the model was pre-trained with an updated version of Code-SearchNet that we rebuilt using GPT-3.5-Turbo.

**Code Summarization** The initial two rows in Table IV reveal that the model pre-trained with cgpt-CSN outperforms the one trained with vanilla CSN on code summarization task, as indicated by higher USE and MRR scores across all six PLs. We further explore the effect of GPT-3.5-Turbo-generated comments data in the fine-tuning phase; we rebuild

TABLE IV: Evaluation results of Code Summarization.

| pre-train | fine-tune | Javascript | | PHP | | Ruby | | Java | | Python | | Go | | **Average** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | USE | MRR | USE | MRR | USE | MRR | USE | MRR | USE | MRR | USE | MRR | USE | MRR |
| CSN | CSN | 0.5404 | 0.8102 | 0.6381 | 0.8102 | 0.5262 | 0.8189 | 0.6031 | 0.8083 | 0.5875 | 0.8294 | 0.6692 | 0.8580 | 0.5941 | 0.8225 |
| cgpt-CSN | CSN | 0.5420 | 0.8889 | 0.6429 | 0.8992 | 0.5375 | 0.8563 | 0.6111 | 0.9135 | 0.5911 | 0.9074 | 0.6763 | 0.9621 | 0.6002 | 0.9046 |
| CSN | cgpt-CSN-Sum | 0.7698 | 0.9623 | 0.8142 | 0.9202 | 0.7599 | 0.9477 | 0.8127 | 0.9342 | 0.6953 | 0.9529 | 0.8142 | 0.9120 | 0.7777 | 0.9382 |
| cgpt-CSN | cgpt-CSN-Sum | 0.7731 | 0.9665 | 0.8162 | 0.9617 | 0.7738 | 0.9617 | 0.8154 | 0.9553 | 0.6983 | 0.9552 | 0.8162 | 0.9244 | 0.7822 | 0.9541 |
| Human References | | - | 0.7710 | - | 0.8094 | - | 0.7988 | - | 0.8091 | - | 0.8153 | - | 0.8481 | - | 0.8086 |
| GPT-3.5-Turbo | | - | 0.9580 | - | 0.9559 | - | 0.9694 | - | 0.9624 | - | 0.9089 | - | 0.9351 | - | 0.9483 |

TABLE V: Evaluation results of NL-Code generation.

| pre-train | fine-tune | CodeBLEU | BLEU | EM |
|---|---|---|---|---|
| CSN | Concode | 39.45 | 38.52 | 22.00 |
| CSN | cgpt-Concode | 49.25 | 48.31 | 29.60 |
| cgpt-CSN | Concode | 40.56 | 40.91 | 22.10 |
| cgpt-CSN | cgpt-Concode | 50.49 | 50.20 | 30.00 |

TABLE VI: Evaluation results of Code Translation.

| pre-train | fine-tune | CodeBLEU | BLEU | EM |
|---|---|---|---|---|
| CSN | Java2Python | 48.28 | 51.29 | 2.57 |
| cgpt-CSN | | 52.38 | 56.72 | 2.60 |
| CSN | Python2Java | 55.52 | 56.82 | 1.21 |
| cgpt-CSN | | 57.19 | 59.92 | 1.84 |

TABLE VII: Evaluation results of Code Refinement.

| pre-train | fine-tune | EM | BLEU |
|---|---|---|---|
| CSN | Refine-small | 21.41 | 77.41 |
| cgpt-CSN | | 21.24 | 77.36 |
| CSN | Refine-medium | 13.90 | 89.39 |
| cgpt-CSN | | 13.74 | 89.51 |

the CSN code summarization datasets, denoted as cgpt-CSN-Sum. Table IV shows the results in the middle two rows. It should be noted that the ground truth for reference-based metrics USE on the rebuilt CSN summarization test set is GPT-3.5-Turbo-generated comments. We observe that the model fine-tuned with the rebuilt summarization data outperforms the model fine-tuned with human-referenced data. These findings indicate that incorporating GPT-3.5-Turbo-generated data, both in the pre-training and fine-tuning phases, advances code summarization performance.

The final two rows in Table IV compare MRR scores between reference comments and GPT-3.5-Turbo-generated comments on the CSN summarization test set. GPT-3.5-Turbo-generated comments achieve markedly higher MRR scores across all six PLs, aligning with findings from Section III. This consistency underscores the generalization capabilities of MRR metric across multiple programming languages.

**Code Generation** The vanilla input in the Concode dataset contains both NL text and class environment context. To investigate the impact of GPT-3.5-Turbo-generated comments in the fine-tuning stage, we also rebuild the Concode dataset by replacing the NL texts with GPT-3.5-Turbo-generated comments to form the updated inputs, as shown in Figure 3 and the updated version noted as cgpt-Concode. Table V shows that the model pre-trained with cgpt-CSN and fine-tuned with cgpt-Concode outperforms other training-tuning settings in three metrics. Compared to the model pre-trained on CSN and fine-tuned on Concode, GPT-3.5-Turbo-generated comments data in both pre-training and fine-tuning phases contributes to 11.04%, 11.68% and 8.00% points improvement on CodeBLEU, BLEU, and EM respectively. Notably, in the fine-tuning phase, the ChatGPT enhanced cgpt-Concode contributes more significant performance gains than pre-training.

We attribute this phenomenon to the characteristics of the NL-Code generation task itself. High-quality NL can better align the semantics with Code, and the model learns better representations in the embedding space during training. In the NL-Code generation task, the quality of NL directly determines the semantic correlation between the input and the final target code. Therefore, in the fine-tuning stage, improving the NL quality of the fine-tuning data of the NL-Code generation task can significantly improve the quality of the generated code. These results demonstrate that high-quality comment data generated from GPT-3.5-Turbo could significantly advance the performance of the NL-code generation task.

**Code Translation** The experiment results of the code translation task in the AVATAR test set are displayed in Table VI. The model pre-trained with cgpt-CSN data performs better than its counterpart pre-trained with human-commented CSN. Compared to the original CSN, GPT-3.5-Turbo-generated comments in the pre-training dataset contribute to an improvement of 4.10 CodeBLEU, 5.43 BLEU and 0.03 EM scores in Java-to-Python, 1.67 CodeBLEU, 3.10 BLEU and 0.63 EM scores in Python-to-Java translation, respectively. The code translation is an NL-unrelated task, and GPT-3.5-Turbo-generated NL in bimodal data can still improve performance. We attribute the improvement to the better alignment between PL and NL. The comments generated by GPT-3.5-Turbo preserve better semantic consistency with code, which could better representation for generation.

**Code Refinement** Experiment results of EM and BLEU scores for code refinement tasks are shown in Table VII. The results indicate no significant performance differences between models trained with CSN and cgpt-CSN on both small and medium code refinement test sets. These results imply that a deep comprehension of programming logic and syntax is essential for recognizing and correcting incorrect code patterns. The enhancement of NL comments in the pre-training stage is less directly applicable to identifying and correcting code errors.

**Clone Detection** Table VIII presents F1 scores, Precision,

TABLE VIII: Evaluation results of Clone Detection.

| pre-train | F1 | P | R |
|---|---|---|---|
| CSN | 0.9464 | 0.9455 | 0.9473 |
| cgpt-CSN | 0.9432 | 0.9358 | 0.9508 |

and Recall for the clone detection task. Models pre-trained on CSN and cgpt-CSN show comparable performance, with F1-score differences of less than 0.5 percentage points. This none non-significant difference can be attributed to the fundamental nature of clone detection: the task primarily relies on analyzing syntactic and semantic similarities between code content pairs rather than their natural language descriptions. Consequently, while GPT-3.5-Turbo generates higher-quality comments, this enhancement does not transfer to improved clone detection performance, as the task is inherently more dependent on code-specific features than natural language documentation.

According to the experimental results on the five code intelligence tasks, we thus conclude that LLMs are qualified benchmark builders and answer the third research question(**RQ3**).

> **Ans. to RQ3: impact of rebuilt data for code tasks**
> Training data rebuilt using GPT-3.5-Turbo's high-quality comments significantly improves the performance of code intelligence tasks that rely on NL, like code summarization and NL-code generation, and it also enhances code translation capabilities. However, tasks primarily focused on code structure and semantics (code refinement and clone detection) show no meaningful improvement from enhanced comment quality.

## V. THREATS TO VALIDITY

**Internal validity**. The scope of our study's conclusions is constrained by limitations in computing resources and the expenses related to using the OpenAI API. Consequently, our research focused solely on a specific pre-trained code model, CodeT5, and we did not extend our analysis to include open-source Code LLMs such as Code Llama [22].

Our study focuses on demonstrating the superior quality of LLM-generated comments over human references for improving training datasets, rather than achieving SOTA performance in code intelligence tasks. While current LLMs might outperform traditional pre-trained models on these tasks, our work's primary contribution lies in dataset quality enhancement rather than model performance comparison. We acknowledge that a direct performance comparison between our rebuilt pre-trained models and current LLMs could provide additional insights, but this falls outside our core research objective of improving training data quality for future model development.

**External validity**. LLMs' outputs can vary significantly with slight changes in the prompt structure, wording, or context. Our study employs a simple, consistent prompt across programming languages to ensure reproducibility. However, this approach may not capture the full potential of LLMs, which could be achieved through more sophisticated prompt engineering. Another threat to the validity stems from the

versions of ChatGPT used in our research, specifically Text-davinci-003, GPT-3.5-turbo, and GPT-4-Turbo, which represent ChatGPT's capabilities at a certain point in time. As commercial LLMs undergo continuous updates, the performance characteristics and findings reported in this study may evolve with newer versions.

## VI. RELATED WORKS

Recent advancements in code intelligence tasks are heavily dependent on the quality of code-comment paired data. Previous studies have identified significant issues in code documentation [18], [41], [79], [80]. Fluri et al. found that 3-10% of code comment changes lagged behind the corresponding code changes in seven Java open-source projects [81]. Such obsolete comments may provide misleading information to developers, leading them to write vulnerable code [79] and thus degrading the quality of the software. The noisy code comments data could degrade the performance of data-driven-based learning models for code intelligence tasks [6].

The main focus of the research community is on developing customized models that can unleash the value of the available data in specific tasks. As mentioned in [82], [83], improving the quality of the training data is still a research opportunity for machine learning, including DL-based source code models. Sun et al. [6] proposed the first framework to improve the dataset quality for code search datasets. Their data cleaning framework, which consists of two subsequent filters, a rule-based syntactic filter and a model-based semantic filter, is considered to filter the noisy data. Xu et al. [84] investigated the data quality issue in the obsolete comment detection problem by proposing data cleaning and adversarial learning techniques. They found that the performance of DL models does improve with the cleaned training data. Unlike the above studies, our work tackles the data quality issue by rebuilding the dataset via LLMs, thereby replacing the noisy data with high-quality ones.

## VII. CONCLUSION

We present a comprehensive evaluation of LLM-generated code comments against human-written references using novel reference-free metrics based on inconsistency detection and code search tasks. Our empirical study found that LLM-generated comments are superior to comments written by humans in the original repositories, challenging the conventional use of human references as the gold standard for code summarization. This finding led us to reconstruct the CodeSearchNet dataset using LLM-generated comments and subsequently retrain the CodeT5 model. The rebuilt model showed significant improvements across multiple code intelligence tasks, particularly in natural language-related tasks such as code summarization and NL to Code generation. These results validate both the superior quality of LLM-generated comments and their practical value in enhancing training datasets for code intelligence models.

We release all data and code of our work online at: https://anonymous.4open.science/r/LLM4DataConstruction.

REFERENCES

[1] X. Song, H. Sun, X. Wang, and J. Yan, "A survey of automatic generation of source code comments: Algorithms and techniques," *IEEE Access*, vol. 7, pp. 111411–111428, 2019.

[2] S. Wang, M. Geng, B. Lin, Z. Sun, M. Wen, Y. Liu, L. Li, T. F. Bissyandé, and X. Mao, "Natural language to code: How far are we?," in *Proceedings of the 31st ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, ACM, 2023.

[3] M. Singh, J. Cambronero, S. Gulwani, V. Le, C. Negreanu, and G. Verbruggen, "Codefusion: A pre-trained diffusion model for code generation," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 11697–11708, 2023.

[4] S. Wang, B. Lin, Z. Sun, M. Wen, Y. Liu, Y. Lei, and X. Mao, "Two birds with one stone: Boosting code generation and code search via a generative adversarial network," *Proceedings of the ACM on Programming Languages*, no. OOPSLA2, 2023.

[5] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *Proceedings of the 40th ICSE*, pp. 933–944, 2018.

[6] Z. Sun, L. Li, Y. Liu, X. Du, and L. Li, "On the importance of building high-quality training datasets for neural code search," in *Proceedings of the 44th ICSE*, pp. 1609–1620, 2022.

[7] S. Wang, M. Geng, B. Lin, Z. Sun, M. Wen, Y. Liu, L. Li, T. F. Bissyandé, and X. Mao, "Fusing code searchers," *IEEE Transactions on Software Engineering*, vol. 50, no. 7, pp. 1852–1866, 2024.

[8] M. Lei, H. Li, J. Li, N. Aundhkar, and D.-K. Kim, "Deep learning application on code clone detection: A review of current knowledge," *Journal of Systems and Software*, vol. 184, p. 111141, 2022.

[9] B. Lin, S. Wang, M. Wen, L. Chen, and X. Mao, "One size does not fit all: Multi-granularity patch generation for better automated program repair," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 1554–1566, 2024.

[10] B. Lin, S. Wang, M. Wen, and X. Mao, "Context-aware code change embedding for better patch correctness assessment," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, may 2022.

[11] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, *et al.*, "Codebert: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1536–1547, 2020.

[12] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, *et al.*, "Graphcodebert: Pre-training code representations with data flow," *arXiv preprint arXiv:2009.08366*, 2020.

[13] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 8696–8708, 2021.

[14] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "Unixcoder: Unified cross-modal pre-training for code representation," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7212–7225, 2022.

[15] C. Li, Z. Xu, P. Di, D. Wang, Z. Li, and Q. Zheng, "Understanding code changes practically with small-scale language models," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pp. 216–228, 2024.

[16] D. Steidl, B. Hummel, and E. Juergens, "Quality analysis of source code comments," in *2013 21st International Conference on Program Comprehension (ICPC)*, pp. 83–92, Ieee, 2013.

[17] A. T. Ying, J. L. Wright, and S. Abrams, "Source code that talks: an exploration of eclipse task comments and their implication to repository mining," *ACM SIGSOFT software engineering notes*, vol. 30, no. 4, pp. 1–5, 2005.

[18] F. Wen, C. Nagy, G. Bavota, and M. Lanza, "A large-scale empirical study on code-comment inconsistencies," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, pp. 53–64, IEEE, 2019.

[19] L. Shi, F. Mu, X. Chen, S. Wang, J. Wang, Y. Yang, G. Li, X. Xia, and Q. Wang, "Are we building on the rock? on the importance of data preprocessing for code summarization," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 107–119, 2022.

[20] X. Hu, G. Li, X. Xia, D. Lo, S. Lu, and Z. Jin, "Summarizing source code with transferred api knowledge," 27th International Joint Conference on Artificial Intelligence, IJCAI 2018, 27.

[21] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "Codesearchnet challenge: Evaluating the state of semantic code search," *arXiv preprint arXiv:1909.09436*, 2019.

[22] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[23] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[24] A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. Ostrow, A. Welihinda, A. Hayes, A. Radford, *et al.*, "Gpt-4o system card," *arXiv preprint arXiv:2410.21276*, 2024.

[25] Q. Guo, J. Cao, X. Xie, S. Liu, X. Li, B. Chen, and X. Peng, "Exploring the potential of chatgpt in automated code refinement: An empirical study," *arXiv preprint arXiv:2309.08221*, 2023.

[26] J. T. Liang, C. Yang, and B. A. Myers, "A large-scale survey on the usability of ai programming assistants: Successes and challenges," in *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*, pp. 605–617, IEEE Computer Society, 2024.

[27] M. Geng, S. Wang, D. Dong, H. Wang, G. Li, Z. Jin, X. Mao, and X. Liao, "Large language models are few-shot summarizers: Multi-intent comment generation via in-context learning," in *Proceedings of the 46th IEEE/ACM ICSE*, pp. 1–13, 2024.

[28] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. Li, *et al.*, "Deepseek-coder: When the large language model meets programming–the rise of code intelligence," *arXiv preprint arXiv:2401.14196*, 2024.

[29] A. Lozhkov, R. Li, L. B. Allal, F. Cassano, J. Lamy-Poirier, N. Tazi, A. Tang, D. Pykhtar, J. Liu, Y. Wei, *et al.*, "Starcoder 2 and the stack v2: The next generation," *arXiv preprint arXiv:2402.19173*, 2024.

[30] S. Gao, Z. Yao, C. Tao, X. Chen, P. Ren, Z. Ren, and Z. Chen, "Umse: Unified multi-scenario summarization evaluation," *arXiv preprint arXiv:2305.16895*, 2023.

[31] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "Bertscore: Evaluating text generation with bert," *arXiv preprint arXiv:1904.09675*, 2019.

[32] S. Haque, Z. Eberhart, A. Bansal, and C. McMillan, "Semantic similarity metrics for evaluating source code summarization," in *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, pp. 36–47, 2022.

[33] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pp. 311–318, 2002.

[34] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, pp. 74–81, 2004.

[35] S. Banerjee and A. Lavie, "Meteor: An automatic metric for mt evaluation with improved correlation with human judgments," in *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pp. 65–72, 2005.

[36] J. Wieting, T. Berg-Kirkpatrick, K. Gimpel, and G. Neubig, "Beyond bleu: Training neural machine translation with semantic similarity," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4344–4355, 2019.

[37] S. Stapleton, Y. Gambhir, A. LeClair, Z. Eberhart, W. Weimer, K. Leach, and Y. Huang, "A human study of comprehension and code summarization," in *Proceedings of the 28th International Conference on Program Comprehension*, pp. 2–13, 2020.

[38] J. Mahmud, F. Faisal, R. I. Arnob, A. Anastasopoulos, and K. Moran, "Code to comment translation: A comparative study on model effectiveness & errors," *arXiv preprint arXiv:2106.08415*, 2021.

[39] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019.

[40] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, *et al.*, "Universal sentence encoder," *arXiv preprint arXiv:1803.11175*, 2018.

[41] M. Linares-Vásquez, B. Li, C. Vendome, and D. Poshyvanyk, "How do developers document database usages in source code?(n)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 36–41, IEEE, 2015.

[42] A. Kanade, P. Maniatis, G. Balakrishnan, and K. Shi, "Learning and evaluating contextual embedding of source code," in *International conference on machine learning*, pp. 5110–5121, PMLR, 2020.

[43] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[44] J. D. M.-W. C. Kenton and L. K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL-HLT*, pp. 4171–4186, 2019.

[45] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[46] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," *arXiv preprint arXiv:2003.10555*, 2020.

[47] A. Svyatkovskiy, S. K. Deng, S. Fu, and N. Sundaresan, "Intellicode compose: Code generation using transformer," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1433–1443, 2020.

[48] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. Clement, D. Drain, D. Jiang, D. Tang, *et al.*, "Codexglue: A machine learning benchmark dataset for code understanding and generation," *arXiv preprint arXiv:2102.04664*, 2021.

[49] W. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "Unified pre-training for program understanding and generation," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2655–2668, 2021.

[50] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 7871–7880, 2020.

[51] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551, 2020.

[52] S. Panthaplackel, J. J. Li, M. Gligoric, and R. J. Mooney, "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, 2021.

[53] T. Steiner and R. Zhang, "Code comment inconsistency detection with bert and longformer," *arXiv preprint arXiv:2207.14444*, 2022.

[54] "CCID Dadaset GoogleDrive." https://drive.google.com/drive/folders/1heqEQGZHgO6gZzCjuQD1EYertN4SAYZ.

[55] "Project Longformer4CCID." https://github.com/theo2023/coco-bert-longformer, 2023.

[56] F. Lv, H. Zhang, J.-g. Lou, S. Wang, D. Zhang, and J. Zhao, "Codehow: Effective code search based on api understanding and extended boolean model (e)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 260–270, IEEE, 2015.

[57] X. Ye, R. Bunescu, and C. Liu, "Learning to rank relevant files for bug reports using domain knowledge," in *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*, pp. 689–699, 2014.

[58] "CodeBERT-CodeSearch." https://github.com/microsoft/CodeBERT/tree/master/CodeBERT/codesearch, 2021.

[59] W. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "A transformer-based approach for source code summarization," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4998–5007, 2020.

[60] H. Wu, H. Zhao, and M. Zhang, "Code summarization with structure-induced transformer," in *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 1078–1090, 2021.

[61] F. Mu, X. Chen, L. Shi, S. Wang, and Q. Wang, "Developer-intent driven code comment generation," *arXiv preprint arXiv:2302.07055*, 2023.

[62] "Text-davinci-003." https://platform.openai.com/docs/models/text-davinci-003, 2023.

[63] "GPT-3.5-Turbo." https://platform.openai.com/docs/models/gpt-3-5, 2024.

[64] "GPT-4-Turbo." https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo, 2024.

[65] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[66] "Code-Llama." https://llama.meta.com/code-llama, 2024.

[67] "CodeBERT-CodeSearch." https://deepseekcoder.github.io, 2024.

[68] "Starcoder2." https://github.com/bigcode-project/starcoder2, 2024.

[69] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. St. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, B. Strope, and R. Kurzweil, "Universal sentence encoder for English," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (E. Blanco and W. Lu, eds.), (Brussels, Belgium), pp. 169–174, ACL, Nov. 2018.

[70] D. Roy, S. Fakhoury, and V. Arnaoudova, "Reassessing automatic evaluation metrics for code summarization tasks," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1105–1116, 2021.

[71] R. Singh and N. S. Mangat, *Elements of survey sampling*, vol. 15. Springer Science & Business Media, 2013.

[72] C.-Y. Su and C. McMillan, "Semantic similarity loss for neural source code summarization," *arXiv preprint arXiv:2308.07429*, 2023.

[73] B. Lin, S. Wang, Z. Liu, Y. Liu, X. Xia, and X. Mao, "Cct5: A code-change-oriented pre-trained model," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1509–1521, 2023.

[74] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, "Mapping language to code in programmatic context," in *Proceedings of the 2018 Conference on EMNLP*, pp. 1643–1652, 2018.

[75] S. Ren, D. Guo, S. Lu, L. Zhou, S. Liu, D. Tang, N. Sundaresan, M. Zhou, A. Blanco, and S. Ma, "Codebleu: a method for automatic evaluation of code synthesis," *arXiv preprint arXiv:2009.10297*, 2020.

[76] W. U. Ahmad, M. G. R. Tushar, S. Chakraborty, and K.-W. Chang, "Avatar: A parallel corpus for java-python program translation," *arXiv preprint arXiv:2108.11590*, 2021.

[77] M. Tufano, C. Watson, G. Bavota, M. D. Penta, M. White, and D. Poshyvanyk, "An empirical study on learning bug-fixing patches in the wild via neural machine translation," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 28, no. 4, pp. 1–29, 2019.

[78] W. Wang, G. Li, B. Ma, X. Xia, and Z. Jin, "Detecting code clones with graph neural network and flow-augmented abstract syntax tree," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 261–271, IEEE, 2020.

[79] W. M. Ibrahim, N. Bettenburg, B. Adams, and A. E. Hassan, "On the relationship between comment update practices and software bugs," *Journal of Systems and Software*, vol. 85, no. 10, pp. 2293–2304, 2012.

[80] H. Malik, I. Chowdhury, H.-M. Tsou, Z. M. Jiang, and A. E. Hassan, "Understanding the rationale for updating a function's comment," in *2008 IEEE International Conference on Software Maintenance*, pp. 167–176, IEEE, 2008.

[81] B. Fluri, M. Wursch, and H. C. Gall, "Do code and comments co-evolve? on the relation between source code and comment changes," in *14th Working Conference on Reverse Engineering (WCRE 2007)*, pp. 70–79, IEEE, 2007.

[82] Y. Zhao, L. Li, H. Wang, H. Cai, T. F. Bissyandé, J. Klein, and J. Grundy, "On the impact of sample duplication in machine-learning-based android malware detection," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 3, pp. 1–38, 2021.

[83] C. Liu, X. Xia, D. Lo, Z. Liu, A. E. Hassan, and S. Li, "Simplifying deep-learning-based model for code search," *arXiv preprint arXiv:2005.14373*, 2020.

[84] S. Xu, Y. Yao, F. Xu, T. Gu, J. Xu, and X. Ma, "Data quality matters: A case study of obsolete comment detection," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 781–793, IEEE, 2023.