

Can Large Language Models Serve as Evaluators for Code Summarization?

Yang Wu, Yao Wan*, Zhaoyang Chu, Wenting Zhao, Ye Liu, Hongyu Zhang, Xuanhua Shi,
Hai Jin *Fellow, IEEE*, Philip S. Yu *Fellow, IEEE, ACM*

Abstract—Code summarization facilitates program comprehension and software maintenance by converting code snippets into natural-language descriptions. Over the years, numerous methods have been developed for this task, but a key challenge remains: effectively evaluating the quality of generated summaries. While human evaluation is effective for assessing code summary quality, it is labor-intensive and difficult to scale. Commonly used automatic metrics, such as BLEU, ROUGE-L, METEOR, and BERTScore, often fail to align closely with human judgments. In this paper, we explore the potential of *Large Language Models (LLMs)* for evaluating code summarization. We propose CODERPE (Role-Player for Code Summarization Evaluation), a novel method that leverages role-player prompting to assess the quality of generated summaries. Specifically, we prompt LLM-based evaluators to take on diverse roles, such as code reviewer, code author, code editor, and system analyst. Each role evaluates the quality of code summaries across key dimensions, including coherence, consistency, fluency, and relevance. We further explore the robustness of LLMs as evaluators by employing various prompting strategies, including chain-of-thought reasoning, in-context learning, and tailored rating form designs. The results demonstrate that LLMs serve as effective evaluators for code summarization. Notably, our LLM-based evaluator, CODERPE, achieves an 80.18% Spearman correlation with human evaluations, outperforming the existing BERTScore metric by 10.39%.

Index Terms—Code Summarization, Large Language Models, Role Player, Model Evaluation

I. INTRODUCTION

Code summarization, which summarizes code snippets into natural-language descriptions, plays an important role in program comprehension and software maintenance [1], [2]. Current approaches to code summarization heavily leverage techniques from *Natural Language Processing (NLP)*, with the aim of translating code snippets from one linguistic representation to another. Recently, this trend has been further bolstered by the emergence of *Large Language Models (LLMs)* [3], e.g., GPT-4 [4], Gemini [5], and Llama [6], owing to their remarkable capabilities in NLP.

Yang Wu, Yao Wan, Zhaoyang Chu and Hai Jin are with the National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China, 430074. Email: {wuyang_emily, wanyao, chuzhaoyang, xhshi, hjin}@hust.edu.cn.

Wenting Zhao and Ye Liu are with Salesforce Research, Palo Alto, USA. Email: {wenting.zhao, yeliu}@salesforce.com.

Hongyu Zhang is with the School of Big Data and Software Engineering, Chongqing University, Chongqing, China. Email: hyzhang@cqu.edu.cn.

Philip S. Yu are with the Department of Computer Science, University of Illinois Chicago, Chicago, USA. Email: psyu@uic.edu.

*Yao Wan is the corresponding author.

Manuscript received November, 2024.

Despite significant advancements in generating code summaries, methods for evaluating their quality have not kept pace. Human evaluation remains the gold standard for assessing code summaries; however, it is labor-intensive and difficult to scale. Previously, in alignment with the NLP research trajectory, numerous automatic metrics, such as BLEU [7], ROUGE [8], METEOR [9], and BERTScore [10], have been widely adopted for evaluating code summarization models. These metrics assess model performance by automatically measuring the similarity between generated summaries and reference (or gold-standard) summaries, which are typically derived from comments provided by developers. However, studies have shown that these automated metrics exhibit a relatively low correlation with human evaluations [11], [12].

Inspired by the remarkable capabilities of LLMs in understanding and generating natural language, recent studies propose leveraging LLMs directly as *reference-free* evaluators [13]–[20]. This research rests on the premise that LLMs can assess candidate outputs based on their generation probabilities without requiring a reference summary, positing that LLMs can assign higher probabilities to outputs that are both fluent and of high quality. Building on these insights, this paper aims to investigate the following question: *Can LLMs effectively serve as evaluators for code summarization?*

A Motivating Example. Figure 1 illustrates an example aimed at providing motivation for our introduced LLM-based evaluator by comparing it with current automatic metrics. This example presents a code snippet accompanied by its corresponding comment, alongside the summary generated by ChatGPT [21]. Upon employing automatic metrics such as BLEU, ROUGE, METEOR, and BERTScore, it is evident that the scores are relatively low, suggesting subpar quality in the generated summary. Nevertheless, upon manual inspection, the quality of the summary generated by ChatGPT is actually high. Specifically, the phrase “a 10% discount for members” in the generated summary correlates directly to the variable “isMember” at line 2 and “0.9” at line 3 in the provided code, revealing the strong understanding and reasoning capabilities of LLMs over source code. We attribute the low scores from automatic metrics to the differences in textual structure and phrasing between the generated summary and the reference summary. Conversely, our LLM-based evaluator showcases its ability to effectively assess the quality of generated summarizations, demonstrating significant alignment with human evaluation metrics.

Our Work. In this paper, we present a pioneering empirical study aiming at investigating the capabilities of LLMs in assess-

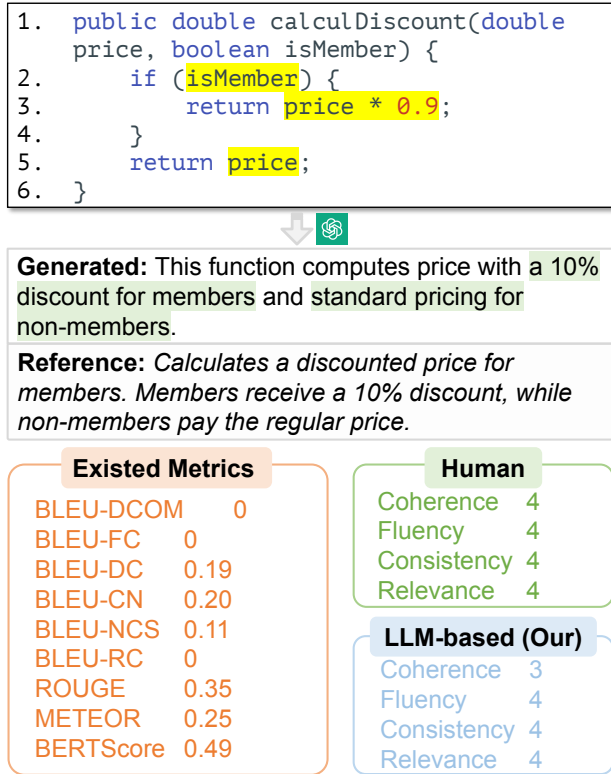


Figure 1: A code snippet with reference and generated summaries, along with scores from existing metrics, human evaluation, and LLMs.

ing code summarization models. In particular, we introduce an evaluation method, termed *CODERPE (Role-Player for Code Summarization Evaluation)*, designed to quantify the quality of generated code summaries. We prompt an LLM-based evaluator to perform various roles, including code reviewer, code author, code editor, and system analyst. Each role is tasked with evaluating the quality of generated code summaries along individual dimensions such as coherence, consistency, fluency, and relevance. By analyzing the outcomes, we determine the most proficient role for each dimension to ensure a more precise and specialized evaluation. In our experiments, we concentrate on three specific LLMs: `gpt-4o-mini` [22], `gpt-4o` [23], and `DeepSeek-V3` [24]. We assess the performance of eight code summarization models, namely `CodeNN` [1], `Deepcom` [25], `Astattgru` [2], `Rencos` [26], `NCS` [27], `Codex` [28], `CodeT5` [29] and `ChatGPT` [30]. We structure our empirical study around the following three *Research Questions (RQs)*.

RQ1: How does LLM-based evaluator CODERPE align with human evaluation compared to traditional metrics? We explore LLMs' capabilities in assessing code summarization, without reference summaries, by examining their alignment with human evaluation alongside conventional metrics such as BLEU, ROUGE, METEOR, and BERTScore.

RQ2: How does the LLM-based evaluator CODERPE perform under varying evaluation settings? We analyze different prompting strategies that may affect the performance of LLMs in evaluating code summarization, including the role-player design, rating form design, chain-of-thought prompting,

and in-context example selection. Our analysis provides clear guidelines for employing LLMs to automate the evaluation of code summarization.

RQ3: How do existing neural models for code summarization perform using our proposed CODERPE? We re-evaluate the effectiveness of six prominent neural models for code summarization tasks, namely `CodeNN`, `Deepcom`, `Astattgru`, `Rencos`, `NCS`, `Codex`, `CodeT5`, and `ChatGPT`, utilizing our LLM-based evaluator, `CODERPE`.

Takeaway Implications. In this paper, we outline several important implications to be noted: (1) Overall, our `CODERPE` demonstrates notable effectiveness in serving as evaluators for code summarization, exhibiting a correlation of 80.18% to human assessment, even in the absence of reference summaries. (2) The effectiveness of LLMs in assessing code summarization relies heavily on carefully crafted role-player engagement and prompting strategies. We recommend integrating a balanced selection of in-context learning examples and increasing evaluation iteration frequency. Additionally, enabling the model to analyze the scoring processes can significantly enhance relevance assessment. (3) With our LLM-based evaluator `CODERPE`, `ChatGPT` outperforms other models by producing summaries that maintain semantic accuracy while offering diverse phrasing, closely aligning with human evaluations.

Contributions. This paper makes the following contributions.

- We perform a pioneering investigation into the ability of LLMs to assess code summarization. Furthermore, we introduce a novel evaluation approach called `CODERPE` which leverages a roleplayer-based prompting strategy to evaluate the coherence and effectiveness of generated summaries, based on an understanding of the code itself, rather than relying on reference ground truth.
- We extensively conduct experiments to compare `CODERPE` with existing metrics in evaluating neural code summarization, employing various prompting strategies. Our experimental findings demonstrate that the LLM-based evaluator substantially enhances the correlation with human judgement across multiple criteria.
- We reassess the efficacy of eight prominent neural models (i.e., `CodeNN`, `Deepcom`, `Astattgru`, `Rencos`, `NCS`, `Codex`, `CodeT5`, and `ChatGPT`) for code summarization, leveraging our novel LLM-based evaluation framework, `CODERPE`.

II. BACKGROUND

A. Existing Code Summarization Evaluation Metrics

We classify prevailing evaluation metrics for code summarization into three categories: *n*-gram-based metrics, embedding-based metrics, and human evaluation metrics.

1) *N-gram-based Metrics*: The objective of *n*-gram-based metrics is to calculate the similarity between the generated summary and the reference summary through the counting of shared *n*-grams. Examples include BLEU [7], ROUGE-L [8], and METEOR [9].

BLEU [7]. *BLEU (Bilingual Evaluation Understudy)* is a traditional neural machine translation metric that calculates av-

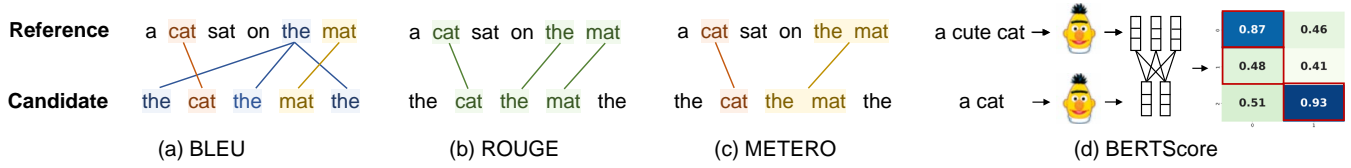


Figure 2: An illustration of existing evaluation metrics for code summarization.

erage n -gram precision with reference sentences and penalizes overly short translations.

$$p_n = \frac{\sum_{w_n \in c} \min \left(C_c(w_n), \max_{j=1, \dots, n} C_{r_j}(w_n) \right)}{\sum_{w_n \in c} C_c(w_n)}, \quad (1)$$

where c and r represent a candidate and its reference sentence, respectively. w_n is an n -gram, and $C_c(w_n)$ denotes its frequency in c . The BLEU score is then computed as:

$$\text{BLEU} = \text{BP} * \exp \left(\sum_{n=1}^N \alpha_n \log p_n \right), \quad (2)$$

where $N = 4$, p_n denotes the precision for n -grams up to N , and α_n represents the positive weighting assigned to each n -gram, and a brevity penalty BP penalizes the short sentences.

In practice, various implementations of the BLEU exist, each with specific modifications to handle n -gram precision differently. BLEU-CN is a sentence BLEU metric applying the Laplace-like smoothing [31] to the precision scores p_n for $n \geq 2$, by both the numerator and denominator by 1. BLEU-DM applies method 0 in NLTK toolkit¹, smoothing precision by assigning a small fixed value when zero-count n -grams occur. BLEU-DC adopts method 4, which adjusts precision based on n -gram frequency, offering a more adaptive solution in sparse data scenarios. Similar to BLEU-CN, BLEU-NCS [27] applies a Laplace-like smoothing method, incrementing the numerator and denominator of all precision values p_n by 1. BLEU-RC [26] is another unsmoothed sentence BLEU variant designed specifically to prevent division-by-zero errors. Instead of traditional smoothing, it adds between 10 and 15 to the numerator, and between 9 and 10 to the denominator of p_n . BLEU-FC is an unsmoothed corpus-level BLEU metric based on NLTK, which aggregates n -gram matches across all hypothesis-reference pairs [2], [32].

EXAMPLE 1. As illustrated in Figure 2(a), the candidate sentence yields unigram counts of "the", "cat", and "mat" as 3, 1, and 1, respectively, with a total count of 5. For reference, the relevant unigrams "the", "cat," and "mat" each have a count of 1. By taking the minimum counts between the candidate and reference, we sum to 3. Thus, with $n = 1$, we have $p_1 = 0.6$, yielding a BLEU score of 0.49.

ROUGE-L [8]. ROUGE-L [33]–[35] quantifies the similarity between a candidate summary and a reference summary by identifying their *Longest Common Subsequence (LCS)*. This metric emphasizes the importance of maintaining the sequential

order of information. The calculation of the ROUGE-L score and its components is defined as follows:

$$P = \frac{LCS(c, r)}{\text{len}(c)}, R = \frac{LCS(c, r)}{\text{len}(r)} \quad (3)$$

$$\text{ROUGE-L} = \frac{(1 + \beta^2) \cdot P \cdot R}{R + \beta^2 \cdot P}.$$

Here, c and r denote the generated candidate and reference summaries, respectively, and β is a parameter that adjusts the balance between precision and recall.

EXAMPLE 2. As the example shown in Figure 2 (b), with consideration of the longest common sequence of words is "cat the mat", we can get $LCS(c, r) = 3$, then $P = 0.6$, and $R = 0.5$. When $\beta = 1$, ROUGE-L equals to 0.54.

METEOR. METEOR, a recall-oriented metric, evaluates how well the model captures reference content by matching words between candidate and reference sentences and computing the harmonic mean of precision and recall, calculated as follows:

$$\text{METEOR} = \max_{j=1, \dots, n} \left(\frac{10PR}{R + 9P} \right) \left(1 - \frac{1}{2} \left(\frac{c}{u} \right)^3 \right), \quad (4)$$

where P and R denote unigram precision and recall, c is the count of matched chunks, and u represents matched unigrams.

EXAMPLE 3. As illustrated in Figure 2 (c), the alignment between the candidate and reference sentences produces two coherent chunks: "cat" and "the mat", resulting in a total of three matched unigrams. With a precision (P) of 0.6 and recall (R) of 0.5, the METEOR score is calculated as 0.5.

2) *Embedding-based Metrics:* While n -gram-based metrics assess semantic similarity through overlapping tokens, embedding-based metrics like BERTScore [10] compare the embeddings of generated and reference summaries.

BERTScore [10]. BERTScore measures sentence similarity using BERT embeddings [10], matching each token in the candidate sentence to tokens in the reference. It tokenizes sentences into words or subwords and represents tokens as embeddings using a pre-trained BERT model, e.g., RoBERTa [36]. Specifically, BERTScore computes a pairwise similarity matrix by taking the inner product of these embeddings, yielding a pre-normalized cosine similarity. Precision is defined as the average of the highest similarity scores for tokens in the candidate sentence, while recall is the average of the highest scores for tokens in the reference sentence. The final BERTScore is computed as the harmonic mean of precision and recall.

EXAMPLE 4. In Figure 2(d), using the RoBERTa model, we compute a cosine similarity matrix between the words of the candidate and reference sentences. Precision is computed by averaging the maximum similarity scores for each word in the

¹<https://pypi.org/project/nltk/3.2.4/>

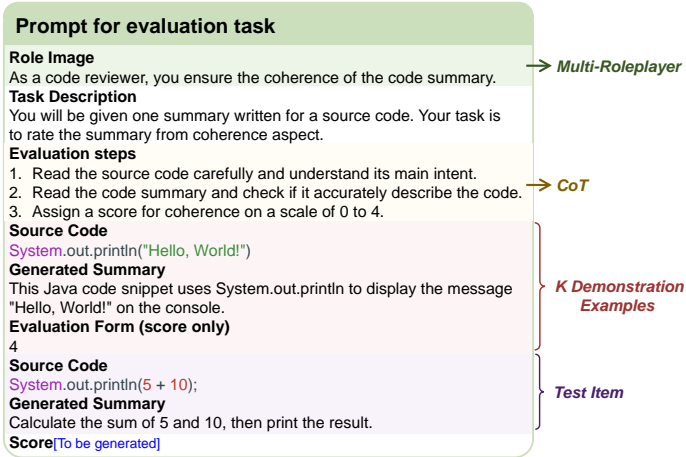


Figure 3: An example of the prompt design to elicit LLMs as a code evaluator taking a “code reviewer” role.

candidate, yielding a value of $(0.87+0.93)/2 = 0.90$. Recall is calculated by averaging the maximum similarity scores for each word in the reference, resulting in $(0.87 + 0.48 + 0.93)/3 = 0.76$. The BERTScore, computed as the harmonic mean of a precision of 0.9 and a recall of 0.76, yields a value of 0.82.

3) *Human Evaluation Metrics*: In human evaluation, human assessors are typically presented with pairs of references and model-generated outputs. The assessors then evaluate the outputs based on predefined criteria such as fluency, coherence, and overall understanding. Often, a 5-point scale, ranging from 1 to 5 with options like “Strongly Disagree”, “Disagree”, “Neutral”, “Agree”, and “Strongly Agree” is employed, allowing assessors to express their judgments on a numerical scale. Aggregating these human judgments yields scores that reflect the perceived quality of the generated text. However, the expense associated with hiring professional evaluators makes human evaluation difficult to scale. Furthermore, human evaluation metrics introduce subjectivity, necessitating efforts to mitigate bias and ensure consistency in the evaluation process.

B. Large Language Models

Prompting. The advent of LLMs is shifting the learning paradigm from the traditional “pre-train and fine-tune” to a novel “pre-train, prompt, and predict” approach. In this framework, downstream tasks are reformulated using textual prompts to align with the original LLM training, rather than through full fine-tuning. In our specific scenario, involving a summary and a code snippet, we can construct a prompt as follows: “[SUMMARY], [SOURCE CODE], Please rate the coherence of the generated summary based on the source code”, where [SUMMARY] signifies the model-generated summary, and [SOURCE CODE] represents the code snippet.

In-Context Learning (ICL). ICL is a special form of prompt-based learning that leverages demonstration examples in prompts to promote the model’s performance. Specifically, given a test question x_t , ICL retrieves k examples related to x_t from the task dataset as demonstrations and uses the prompt function f to transform these examples, creating a set

of demonstration examples $D_k = \{f(x_i, y_i), \dots, f(x_k, y_k)\}$. Then, the LLMs predict \hat{y}_t based on the task description I and example set D_k .

Chain-of-Thought (CoT). CoT [37] enhances the reasoning capabilities of large language models by introducing explicit intermediate steps into the prompt. In our framework, CoT is instantiated as a set of structured evaluation instructions appended to the task description, which systematically guide the model from the input x through intermediate reasoning steps to produce the final output y .

Multi-Role Player. LLMs have demonstrated the capability to simulate human behavior through role-playing, adapting to diverse roles across contexts. Recent studies leverage this versatility, from simulating character personalities in gaming [38] to aiding consensus-building in robotics [39]. In this work, we engage LLMs in expert roles to elicit their ability in code summary evaluation.

EXAMPLE 5. Figure 3 showcases a prompt design that casts an LLM in the role of a “code reviewer”. The figure details five key components of LLMs: 1) a role, which assigns the expected identity and function of the LLM during the task, 2) a task description that clearly articulates the goal, 3) evaluation steps that segment the task into manageable parts, 4) k demonstration examples that provide a model for performing the task, and 5) a test example for the LLM to assess.

III. LLMs AS CODE SUMMARIZATION EVALUATORS

This section presents CODERPE, a novel evaluation method that employs a role-player prompting strategy to assess the quality of generated summaries, accommodating both reference-based and reference-free scenarios, as shown in Figure 4.

A. Multi-Role Player Design

The core idea of CODERPE is designing a multi-role player prompting strategy. Specifically, we prompt an LLM-based evaluator to play diverse roles such as code reviewer, code author, code editor, and system analyst. Each role is required to evaluate the quality of generated code summaries, focusing on one dimension at a time, such as coherence, consistency, fluency, and relevance. The schema of *role profile prompt* is presented as $\langle \text{role description}, \text{role dimension} \rangle$. We replace the *role description* slot with a variety of roles (e.g., code reviewer, original code author, code editor, and system analyst) and investigate their performance on the specific role dimension. The slot of *role dimension* could be filled with descriptions of various dimensions (e.g., coherence, consistency, fluency, and relevance) to guide the roles to behave. The specific descriptions are as follows.

Coherence Dimension: Guarantee that the summary presents the functionality and logic of the code in a clear, organized manner that is easy to follow.

Consistency Dimension: Check whether all information in the summary is factually grounded in the source code, without hallucinated or unsupported content.

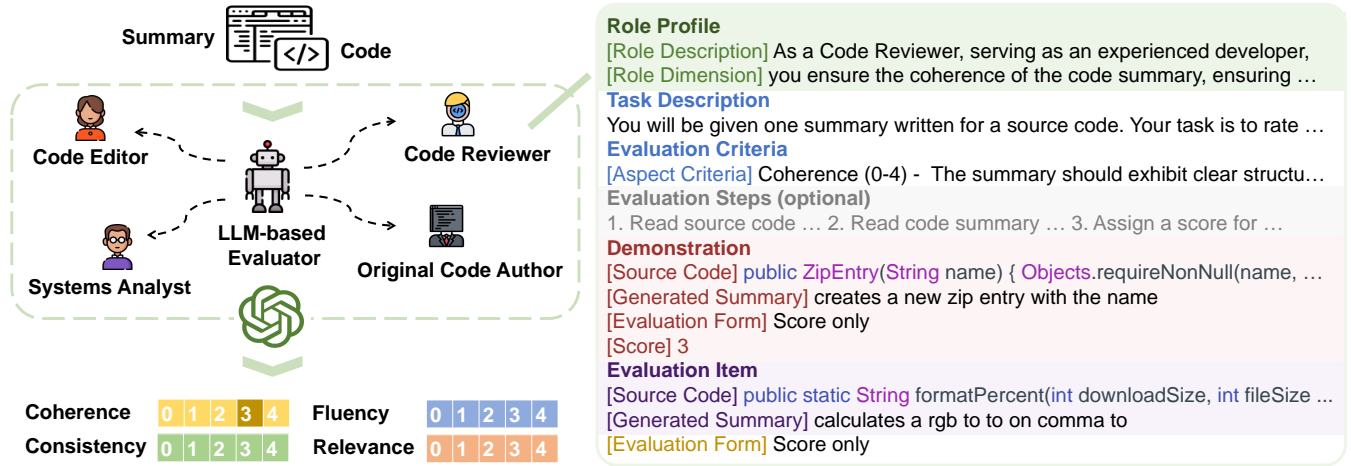


Figure 4: An overview of our proposed framework for prompting LLMs as diverse evaluators for code summarization.

Fluency Dimension: Focus on ensuring that the summary is written smoothly, with clear sentences and appropriate wording.

Relevance Dimension: Determine whether the summary captures the most important aspects of the code, avoiding trivial or marginal details.

B. Prompting Strategies

To clarify the evaluation task for LLMs, we provide a *task description* stating, “you will be given one summary written for a source code and your task is to rate the summary on one metric”. Subsequently, we investigate various prompt strategies to enhance the performance of LLMs.

1) *CoT*: Considering the evaluation task, detailed evaluation instructions can guide the annotator in inferring the rating score. We follow the CoT strategies [14] to prompt LLM to generate detailed evaluation steps on its own. Specifically, we incorporate the *evaluation step* illustrated in Figure 4, detailing the reasoning process to derive the final score. The specific stages of the evaluation process are outlined in the boxed evaluation guidelines below.

1. Read the source code carefully and understand its main functionality and key operations.
2. Read the code comments and compare them to the source code. Check if the comments accurately describe the main functionality and key operations of the code, and present them in a clear and logical order.
3. Assign a score for coherence on a scale of 0 to 4, based on the Evaluation Criteria.

2) *ICL*: To enhance LLMs’ performance in the evaluation task, we provide them with selected annotated examples. As illustrated in Figure 4, we employ a *demonstration prompt*, which can be extended by adjusting the number of examples, denoted as K . Each example is meticulously linked to a specific evaluation dimension and rated on a scale of 0 to 4. These examples comprise a code snippet, an optional reference summary, a generated summary, a specific rating form, and an associated human-assigned score.

3) *Rating Forms*: Lastly, we feed LLMs the generated summary for evaluation, along with its source code and an optional reference summary, ending with a *evaluation form* slot in the *evaluation item prompt*, as illustrated in Figure 4. We explore diverse scoring formats to guide LLMs to output ratings. In addition to the traditional *score-only form*, where LLMs output only a numerical score, we introduce two more nuanced approaches based on the evaluation guidelines detailed in [16]: the *analyze-rate form* and the *rate-explain form*. The *analyze-rate form* requires LLMs to process the reasoning behind their assessment before giving a score, whereas the *rate-explain form* prompts LLMs to score first and then justify their evaluation. Specifically, we replace the *evaluation form* slot with diverse form descriptions as follows.

Score-only Form: “Score only”.

Analyze-rate Form: “Answer by starting with ‘Analysis’ to analyze the given example regarding the evaluation criteria as concisely as possible, and then give the numeric rating on the next line by ‘Rating’”.

Rate-explain Form: “Answer by starting with ‘Rating’ and then give the explanation of the rating on the next line by ‘Rationale’”.

IV. EXPERIMENTS

We assess the performance of LLMs as evaluators for the code summarization task by addressing the following *Research Questions (RQs)*:

- **RQ1: Performance of CODERPE Evaluator.** To what extent does the LLM-based evaluator CODERPE align with human evaluation compared to traditional metrics?
- **RQ2: Influence of Evaluator Settings.** How does the LLM-based evaluator CODERPE perform across different settings, such as evaluator types, number of demonstration examples, turns, and prompt strategies?
- **RQ3: Re-Evaluation of Current Models.** How do existing neural models for code summarization perform when evaluated using our proposed LLM-based CODERPE?

A. Datasets and Models to Re-Evaluate

1) *Datasets*: We conduct experiments using three widely used datasets for code summarization, following the methodology of prior work [11]. The TL-CodeSum dataset contains 87,136 Java code-summary pairs from 9,732 GitHub projects (2015–2016), each with at least 20 stars, and is split into training, validation, and test sets in an 8:1:1 ratio. CodeSearchNet consists of 496,688 Java methods with corresponding summaries, covering a wide range of programming tasks. Funcom includes 2.1 million Java method-summary pairs from 28,945 GitHub projects. For our evaluation, we adopt the 300 summaries randomly sampled from these three datasets (100 per dataset) as provided by prior work [11].

2) *Models to Re-Evaluate*: In our experiments, we re-evaluate the following eight code summarization models: **CodeNN** [1] is an early neural model for code summarization, which encodes source code into context vectors and then generates summaries using an attention mechanism. **Deepcom** [25] linearizes source code by traversing its abstract syntax tree and employs an LSTM-based model to translate the traversal into a summary. **Astattgru** [2] employs two recurrent neural networks to encode both the lexical and syntactic information of source code. **NCS** [27] utilizes a Transformer-based model to generate summaries for code. **Rencos** [26] incorporates similar code snippets retrieved from the training dataset to enhance the code summarization model. **ChatGPT** [30] denotes the ChatGPT-based model for code summarization via prompting. Following their proposed setup, we use the optimal prompt design with gpt-4o model to generate summaries. **CodeT5** [29], a pre-trained encoder-decoder Transformer specifically fine-tuned for code-related tasks with identifier-aware representation learning. **CodeX** [28], OpenAI's advanced code generation model that was trained on an extensive corpus of public source code and demonstrates remarkable few-shot learning capabilities for various programming tasks.

3) *LLMs as Evaluators*: To ensure a rigorous and unbiased evaluation of summary quality, we employ a diverse set of LLMs as evaluators, to mitigate potential biases that may arise from model-series preferences or architectural homogeneity. Our evaluation framework incorporates multiple state-of-the-art models from both close-source and open-source domains. The close-source models include OpenAI's GPT-3.5 series (gpt-3.5-turbo [40]), which demonstrates strong performance in code-related tasks, the compact yet efficient gpt-4o-mini [22] optimized for rapid inference, and the advanced gpt-4o [23] representing the cutting-edge of OpenAI's model capabilities. To provide balance and ensure generalizability, we incorporate two prominent open-source alternatives: DeepSeek-V3 [24], a powerful multimodal model with strong code understanding capabilities, and Qwen3-32B [41], Alibaba's large-scale monolingual model that has shown competitive performance on various NLP benchmarks.

B. Evaluation Criteria

1) *Dimensions to Assess a Code Summary*: We evaluate generated code summaries across four key dimensions [42]:

- **Coherence (0-4)**. The summary should be logically organized, with a clear flow of ideas from sentence to sentence, forming a coherent description of the source code.
- **Consistency (0-4)**. The summary must align with the *facts* within the source code, e.g., specific statements, avoiding unsupported or hallucinated content.
- **Fluency (0-4)**. The summary should be grammatically correct, well-structured, and free from repetition, formatting issues, and capitalization errors that impede readability.
- **Relevance (0-4)**. The summary should capture the essential information from the source code, with penalties for redundancies and excessive details.

Following [12], we engage 15 annotators—9 senior undergraduates and 6 graduate students, all with advanced English proficiency and 5+ years of software development experience—to ensure stable and reproducible evaluation scores. Each annotator rates 300 samples on a 0-4 scale for coherence, consistency, fluency, and relevance of generated summaries based on the corresponding code snippets. We calculate Kendall's Tau to verify agreement among the 15 annotators. Then, we average their scores for each generated summary. To enhance the robustness of the evaluation, a post-annotation discussion was conducted to review and reconcile samples with substantial score discrepancies. During this process, each annotator explained their scoring rationale, and a lead annotator reassigned a final reconciled score based on the discussion. Final scores are updated accordingly.

2) *Metrics to Assess the Alignment with Human*: We calculate the correlation between automatic evaluation metrics with human scores employing Kendall-Tau correlation coefficient [43] and Spearman correlation coefficient [44].

Kendall-Tau Correlation Coefficient (τ). This metric evaluates the ordinal association between the datasets. It provides a robust measure of the ordinal association between two measured quantities. For two datasets X and Y each with n data points, the Kendall-Tau correlation is defined as:

$$\tau = \frac{2}{n(n-1)} \sum_{i < j} \text{sgn}(x_i - x_j) \cdot \text{sgn}(y_i - y_j), \quad (5)$$

where n is the number of pairs, and x_i, x_j, y_i, y_j are the ranks of the data points in the two datasets, X and Y, respectively.

Spearman Correlation Coefficient (ρ). It is particularly useful when the data does not conform to a normal distribution or when the relationship between variables is non-linear but monotonic. Spearman correlation ρ is given by:

$$\rho = 1 - \frac{6 \sum_{i=1}^n d_i^2}{n(n^2 - 1)}, \quad (6)$$

where d_i is the difference between the ranks of corresponding variables, and n is the total number of observations.

C. RQ1: Performance of CODERPE Evaluator

We investigate LLM-based evaluators by assigning diverse role-players, such as code reviewers, original code authors, code editors, and systems analysts, to assess code summarization across four dimensions: coherence, consistency, fluency, and relevance. We evaluate the performance of CODERPE

Table I: The overall performance of code summarization by employing CODERPE backend by LLMs across various role players, compared with conventional metrics.

Metric	Coherence		Consistency		Fluency		Relevance		Average	
	τ	ρ	τ	ρ	τ	ρ	τ	ρ	τ	ρ
Existing Metrics										
BLEU-DM	23.98%	50.53%	26.92%	55.28%	23.60%	49.44%	28.44%	58.31%	25.74%	53.39%
BLEU-FC	23.97%	50.53%	26.92%	55.28%	23.60%	49.44%	28.44%	58.31%	25.73%	53.39%
BLEU-DC	39.05%	55.86%	43.66%	60.41%	36.83%	52.39%	46.81%	63.86%	41.59%	58.13%
BLEU-CN	37.00%	53.37%	39.80%	55.14%	35.44%	50.41%	42.44%	58.50%	38.74%	54.36%
BLEU-NCS	35.80%	51.30%	31.71%	44.76%	33.95%	48.27%	34.27%	47.83%	33.93%	48.04%
BLEU-RC	23.98%	50.53%	26.92%	55.28%	23.60%	49.44%	28.44%	58.31%	25.74%	53.39%
ROUGE-L	37.17%	53.16%	49.29%	66.98%	37.43%	53.09%	51.98%	69.75%	43.97%	60.75%
METEOR	40.78%	57.72%	51.03%	68.68%	38.79%	54.56%	54.58%	72.11%	46.30%	63.27%
BERTScore	46.41%	64.50%	54.10%	71.42%	47.72%	65.64%	59.39%	77.59%	51.91%	69.79%
LLM-based Evaluators										
gpt-4o-mini backend										
Baseline (No Role)	48.34%	71.35%	54.89%	79.15%	51.11%	74.30%	56.39%	79.62%	52.68%	76.11%
Code Reviewer	50.83%	73.88%	56.04%	79.93%	50.79%	73.40%	58.12%	80.82%	53.95%	77.01%
Original Code Author	57.25%	80.74%	58.14%	82.22%	57.26%	80.74%	53.34%	77.02%	56.50%	80.18%
Code Editor	51.31%	74.72%	54.71%	78.70%	55.93%	78.75%	56.09%	79.53%	54.51%	77.93%
Systems Analyst	51.59%	75.08%	57.03%	81.24%	51.37%	73.95%	54.62%	77.34%	53.65%	76.90%
gpt-4o backend										
Code Reviewer	44.46%	66.88%	50.67%	74.05%	46.29%	73.48%	53.72%	77.56%	48.79%	72.99%
Original Code Author	48.24%	71.90%	49.88%	71.53%	43.60%	72.63%	55.91%	80.37%	49.41%	74.11%
Code Editor	49.19%	73.36%	53.35%	76.86%	46.43%	73.84%	54.32%	78.58%	50.82%	75.66%
Systems Analyst	49.52%	72.96%	53.40%	76.70%	45.53%	73.80%	53.26%	76.27%	50.43%	74.93%
DeepSeek-V3 backend										
Code Reviewer	49.31%	72.21%	53.10%	77.83%	49.86%	77.30%	57.62%	82.04%	52.47%	77.35%
Original Code Author	50.57%	73.31%	51.68%	76.58%	51.72%	77.65%	58.34%	83.23%	53.08%	77.69%
Code Editor	48.88%	71.45%	54.37%	79.11%	52.65%	80.12%	57.48%	81.52%	53.35%	78.05%
Systems Analyst	51.67%	75.70%	50.79%	76.63%	51.32%	80.55%	56.38%	81.11%	52.54%	78.50%
Qwen3-32B backend										
Code Reviewer	48.76%	72.73%	49.46%	73.45%	41.15%	70.02%	50.11%	75.73%	47.37%	72.98%
Original Code Author	48.23%	72.89%	47.89%	70.47%	36.68%	63.62%	53.69%	80.28%	46.62%	71.82%
Code Editor	48.23%	71.68%	52.02%	76.73%	34.50%	62.65%	50.51%	77.63%	46.32%	72.17%
Systems Analyst	49.23%	72.54%	52.75%	77.04%	36.19%	64.77%	49.75%	75.15%	46.98%	72.38%

in its basic mode. This mode includes role profiles, task descriptions, evaluation criteria, demonstrations, and evaluation items. The default setting uses four demonstration examples. It then proceeds with a single rating round in a score-only format. We consider 4 variants of LLMs, i.e., GPT-4 series (e.g., gpt-4o-mini and gpt-4o) and open-source LLMs (e.g., DeepSeek-V3 and Qwen3-32B) as the backbone models and conduct experiments using the same dataset and summarization models—CodeNN, Deepcom, Astatgru, NCS, and Rencos—utilized in previous work [11]. To ensure a fair comparison, we adhere to the methodology of prior work [11] and evaluate 300 randomly sampled summaries, with 100 summaries per dataset, generated by these models on the TL-CodeSum, CodeSearchNet, and Funcom datasets with approximately 20 summaries per model per dataset.

Table I presents the results of experiments employing LLM-based evaluators across various role players, compared with conventional metrics. Importantly, while these conventional metrics rely on lexical overlap with reference summaries and are limited in their ability to assess nuanced qualities like coherence or fluency. BERTScore achieves the highest average Spearman correlation (69.79%) among them.

Remarkably, LLM-based evaluators under the CODERPE framework consistently outperform these baselines. Notably, models prompted with specific roles, such as Original Code Author, Systems Analyst, and Code Editor, achieve stronger alignment with human judgments. CODERPE allows flexible role selection tailored to each backbone model. For example,

using the gpt-4o-mini backend, the Original Code Author role reaches an average Spearman correlation of 80.18%, which surpasses BERTScore by over 10 percentage points. Similar performance is observed for this role under the gpt-4o (74.11%) and DeepSeek-V3 (77.69%) backends. Code Editor role yields the highest correlation under the DeepSeek-V3 (78.05%) and gpt-4o (75.66%). gpt-4o-mini backbone not only achieves the best overall correlation but also offers a favorable trade-off between performance, latency, and cost, making it a practical choice for scalable evaluation scenarios. These results further validate the robustness and flexibility of the CODERPE framework across different model families.

Significantly, it is crucial to observe that these exceptional performances are achieved without the necessity of reference summaries, a requirement intrinsic to conventional metrics. This indicates that reference-free assessments are effective, as the scores are not strongly dependent on the reference, showing that LLMs can objectively evaluate code summaries under our framework.

Answer to RQ1. Our proposed CODERPE demonstrates a superior correlation with human scores across various dimensions, including coherence, consistency, fluency, and relevance, surpassing existing metrics.

D. RQ2: Influence of Evaluator Settings

1) *Influence of Number of Demonstration Examples:* To analyze the impact of demonstration examples on our proposed

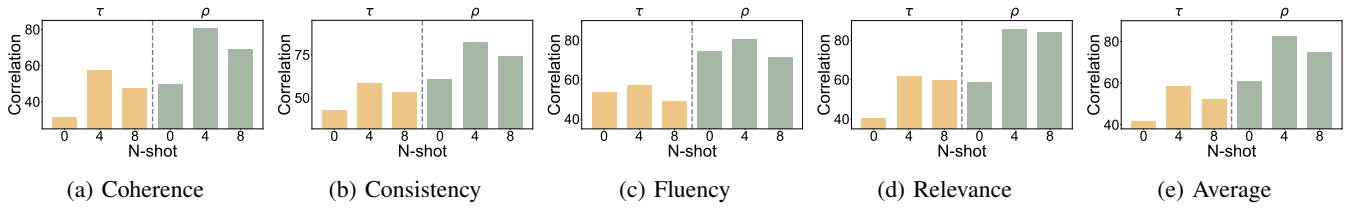


Figure 5: Impact of demonstrations in LLM-based evaluator by gpt-4o-mini in the reference-free scenario.

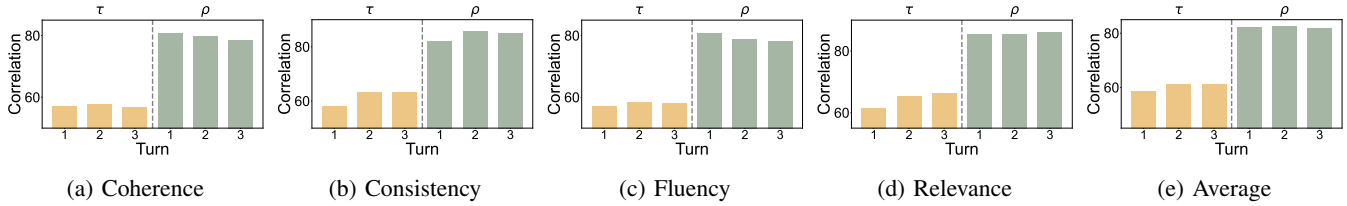


Figure 6: Impact of rounds in LLM-based evaluator by gpt-4o-mini in the reference-free scenario.

Table II: The correlation of various prompting strategies on the LLM-based evaluator by gpt-4o-mini with human evaluation.

Ablations		Coherence		Consistency		Fluency		Relevance		Average	
CoT	Forms	τ	ρ	τ	ρ	τ	ρ	τ	ρ	τ	ρ
✓	Score only	40.06%	63.27%	44.04%	63.04%	44.04%	69.75%	45.51%	67.70%	43.41%	65.94%
×	Score only	57.25%	80.74%	58.14%	82.22%	57.26%	80.74%	58.12%	80.82%	57.69%	81.13%
×	Rate-explain	45.15%	66.59%	51.67%	72.60%	49.83%	73.72%	52.53%	73.89%	49.80%	71.70%
×	Analyze-rate	45.89%	67.88%	57.94%	80.88%	55.46%	80.92%	61.56%	85.50%	55.21%	78.80%

LLM-based evaluator for code summarization, we target the gpt-4o-mini model and vary the number of demonstration examples used: 0, 4, and 8. Figure 5 presents the results for varying numbers of demonstration examples. Notably, compared to the 0-shot configuration, the 4-shot setup improves the average Kendall’s Tau correlation from 41.79% to 58.55%, and Spearman correlation from 60.94% to 82.30%, marking absolute gains of 16.76% and 21.36%, respectively. Thus, we recommend this configuration for enhanced performance.

2) *Influence of Rounds*: We examine rating generalizability and robustness by varying the number of evaluation rounds (i.e., the number of times each summary is independently assessed by the model) and averaging the resulting scores. Specifically, we explore the impact of rounds on gpt-4o-mini evaluator when assessing code summarization by varying the number of rounds from 1 to 3. From Figure 6, we observe that increasing the number of evaluation rounds leads to more stable performance across metrics, rather than a consistent improvement. While individual scores fluctuate slightly with each additional round, the overall averages remain within a narrow range. This indicates that using multiple rounds enhances the robustness and reliability of the evaluator. Notably, the average results from three rounds closely approximate human judgment, achieving a Kendall-Tau correlation of 61.20% and a Spearman correlation of 82.38%.

3) *Influence of Prompt Settings*: We conduct an ablation study to investigate the effect of different prompting strategies on evaluation quality, focusing on two aspects, as introduced in Sec. ??: the inclusion of CoT reasoning in the evaluation steps, and the design of rating forms (*score-only*, *rate-explain*, and *analyze-rate*). Table II shows the correlation between

different prompting strategies with human evaluation results. Interestingly, the use of CoT prompts does not yield performance gains. In fact, when CoT is enabled, the average correlation drops notably across all dimensions. This suggests that generic step-by-step reasoning does not enhance alignment with human evaluators and may introduce unnecessary verbosity or distraction, especially when role-specific prompts are already guiding the evaluation behavior. Among the rating forms, the *score-only* approach without CoT achieves consistently strong performance and serves as a competitive baseline. The *rate-explain* and *analyze-rate* forms offer limited improvement in most dimensions. However, a notable exception is the *Relevance* dimension: the *analyze-rate* form substantially outperforms other variants, achieving the highest correlation with human scores ($\tau=61.56\%$, $\rho=85.50\%$). Interestingly, we find the *analyze-rate* form consistently outperforms the *rate-explain* form across all aspects. It is important to highlight that employing these form-based strategies does not necessarily result in improved performance when compared to a simplistic *score-only* approach, where LLMs are simply tasked with providing a numerical score. Based on this observation, we adopt a hybrid configuration: the *analyze-rate* form is used exclusively for the relevance dimension, while the *score-only* form is retained for coherence, consistency, and fluency. This selective strategy balances simplicity with targeted enhancement, leveraging structured reasoning only where it offers clear benefit.

Answer to RQ2. While CODERPE surpasses traditional metrics in achieving best performance for code summarization evaluation, it requires careful prompt design and the selection of an appropriate base LLM.

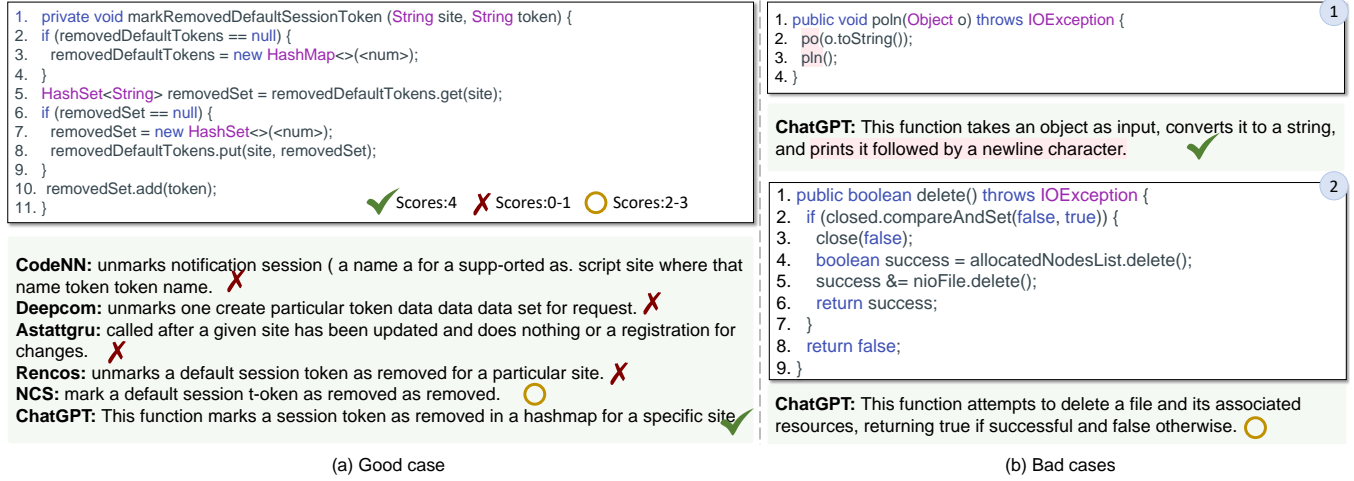


Figure 7: Case studies to investigate when CODERPE works and fails.

Table III: Reevaluation of code summarization models with CODERPE (with gpt-4o-mini), normalizing the average scores to a range of 0% to 100%.

Model	Coherence	Consistency	Fluency	Relevance
CodeNN	27.00%	29.00%	30.75%	32.50%
Deepcom	20.50%	19.25%	22.50%	24.50%
Astattgru	33.75%	31.25%	39.75%	39.25%
Rencos	39.25%	48.00%	53.75%	51.75%
NCS	43.00%	57.50%	57.00%	55.75%
CodeT5	45.75%	51.50%	60.00%	62.00%
Codex	83.50%	74.75%	87.25%	80.25%
ChatGPT	97.00%	88.75%	97.75%	89.25%

E. RQ3: Re-Evaluation of Current Models

We systematically re-evaluate established code summarization models, e.g., CodeNN, Deepcom, Astattgru, Rencos, NCS, Codex, CodeT5, and ChatGPT, with the LLM-based evaluator CODERPE equipped by gpt-4o-mini. Specifically, we randomly sample 100 code snippets from the TL-CodeSum dataset and generate summaries for them using each model.

Table III shows the evaluation results. Interestingly, our findings contrast sharply with prior studies using traditional automatic metrics. For example, Sun et al. [30] report that ChatGPT has 10.28 BLEU and 20.81 ROUGE-L, lower than NCS’s 15.8 BLEU and 31.3 ROUGE-L. However, our LLM-based evaluation reveals a starkly different ranking, where ChatGPT achieves the highest performance across all four human-centric quality dimensions, with 97.00% in coherence, 88.75% in consistency, 97.75% in fluency, and 89.25% in relevance. In comparison, NCS model reaches only 43.00%, 57.50%, 57.00%, and 55.75% on the same dimensions, respectively. Codex and CodeT5 also lag behind, with Codex achieving 83.50% coherence and CodeT5 only 45.75%. Our research demonstrates ChatGPT’s effectiveness through LLM evaluators, emphasizing the importance of developing robust evaluation methods for code summarization performance assessment.

Answer to RQ3. Our proposed CODERPE shows that ChatGPT outperforms other neural models in code summarization, often underestimated by traditional automatic metrics.

V. DISCUSSION

A. When do LLM-based Evaluators Work and Fail?

To enhance our comprehension of the conditions under which LLM-based evaluators succeed or encounter challenges, we conduct an exhaustive case study, encompassing both a good case and a case characterized by errors. In Figure 7 (a), a good case is presented, wherein both a code snippet and its corresponding summaries generated by various neural models are provided. In analyzing this instance, it becomes apparent that the code summary produced by ChatGPT exhibits notable differences compared to other summaries. However, employing our proposed LLM-based evaluator reveals that the code summary generated by ChatGPT attains the highest score of 4, signifying superior quality. Upon manual inspection of the quality of generated summaries, we acknowledge that the summaries produced by ChatGPT exhibit superior quality compared to those generated by other models, showcasing a remarkable alignment with human judgments. This superiority can be attributed to the impressive summarization capabilities inherent in LLMs.

Moreover, we present two scenarios to illustrate instances where LLM-based evaluators may fail to assess code summarization models, as depicted in Figure 7 (b). In the first error case, the LLM evaluator gives the consistency of the summary a score of 4. However, from our manual inspection, we can see a hallucination content “*prints it followed by a newline character*”, which is extended from the words “po” and “pln”. The LLM-based evaluator is unaware of the fact that “po” and “pln” functions are defined outside the scope of the provided code. This oversight might originate from pre-training biases, where LLMs prematurely learned to associate these abbreviations with common methods. In the second error case, we can see that the code summary generated by ChatGPT receives a score of 2, despite being deemed concise and effective by human annotators. This could be due to the model’s subjective interpretation of an ideal summary, leading it to perceive the provided summary as lacking in detail.

These failure cases reveal two key challenges: (1) LLM evaluators sometimes exhibit hallucination tendencies, where their

judgments rely on prior associations rather than context-aware understanding of the code; and (2) subjective factors—such as preferences for summary length or style—can influence the evaluation outcomes, leading to inconsistencies in scoring. To address these limitations, one possible direction for future work might be to explore multi-agent evaluation setups, where multiple LLMs assess and critique each other's judgments to reduce individual bias and improve robustness.

B. Implications

In this study, we have obtained several significant implications that offer valuable insights for further study.

Implication 1. Our comprehensive investigations demonstrate that LLMs can indeed serve as effective evaluators for code summarization, surpassing established evaluation metrics (i.e., BLEU-R, ROUGE-L, METEOR, and BERTScore) in their alignment with human judgments. This can inspire our research community to develop enhanced LLMs for assessing the efficacy of code intelligence tasks, encompassing code generation, vulnerability detection, and many others.

Implication 2. Our comprehensive ablation studies underscore the importance of meticulous attention to designing effective prompting strategies, careful selection of the LLM as the backbone, and thoughtfully setting the roles for each LLM-based evaluator. This provides valuable insights for the research community to optimize the use of LLMs, enhancing their effectiveness in evaluating code intelligence tasks.

Implication 3. Our LLM-based evaluators reveal that ChatGPT outperforms other neural models in code summarization, providing strong evidence for the efficacy of LLMs in understanding code and generating precise summaries.

C. Threats to Validity

Construct Validity. Prompt design plays a central role in steering the behavior of LLM-based evaluators. To ensure evaluations are both consistent and fair, we focus on two key challenges that can affect reliability—prompting variability and role ambiguity. Prompting variability arises when subtle differences in wording, structure, or examples unintentionally alter model behavior. To reduce such variability, we adopt a standardized prompt template (as described in Section III-B) with fixed phrasing and format. Prompt content is constrained to predefined role-specific fields, and a curated set of examples ensures consistent n-shot demonstrations across all conditions. Role ambiguity refers to the model's potential failure to internalize the intended responsibilities when prompted to act as a specific role. This can result in inconsistent or biased evaluations, especially when the model lacks domain-specific knowledge (e.g., in technical roles such as system analyst). Future work could investigate instruction-tuning to better ground each role's role-specific decision-making.

Internal Validity. A key concern is carry-over and order effects, where the same model evaluates multiple summaries in sequence and its scoring of later summaries may be biased by earlier ones. To mitigate this, we treat each evaluation—defined by a specific summary, dimension, and role—as an independent

API call, ensuring that each assessment is conducted in isolation, free from influence by prior evaluations. To validate the model-based evaluation, human annotation serves as the gold standard, with 15 annotators scoring the summaries, and reconciliation applied to cases with significant disagreement. This multi-rater design and post-annotation resolution process improve reliability and reduce the risk of bias, thus supporting the internal validity of our findings.

External Validity. In this study, we adopt prior work [11] amassing a dataset comprising 300 code summaries. These summaries are used to assess both proprietary evaluators (e.g., gpt-3.5-turbo, gpt-4o-mini, and gpt-4o) and open-source models (e.g., DeepSeek-V3 and Qwen3-32B). We do not consider potential prior exposure to this dataset as a major threat, as the task focuses on evaluating rather than generating summaries. To reduce bias toward any specific model family, we include summaries generated by a diverse set of models, including traditional methods (e.g., CodeNN, DeepCom, Astatgru, Rencos and NCS) and recent LLM-based approaches (e.g., Codex, CodeT5, and ChatGPT). In addition, we acknowledge that some recent summarization models (e.g., EditSum, SG-Trans) and other structure-aware or task-specific approaches may also be relevant, and we intend to explore such specialized methods in future work.

VI. RELATED WORK

A. Code Summarization

Source code summarization plays a critical role in improving program comprehension and maintenance. Recently, deep learning-based techniques have driven the development of automated approaches to code summarization [45]–[48], such as DeepCom [25], NCS [27], and SIT [49], utilizing large-scale code-summaries corpora for training generative models to translate code into natural language summaries [1], [2], [11], [50], [51]. More recently, the emergence of LLMs has garnered significant interest, prompting numerous studies to investigate its potential for code summarization [30], [52]–[55]. For example, Su et al. [55] investigated diverse prompting techniques, including zero-shot, few-shot, chain-of-thought, critique, and expert methods, to adapt LLMs for code summarization. Complementary to these studies, this paper focuses on evaluating the quality of generated code summaries.

B. Code Summarization Evaluation

Evaluating code summarization is challenging due to the inherently open-ended nature of the task. Existing evaluation approaches can be generally divided into two categories: automatic and human evaluations. Automatic methods typically rely on metrics such as BLEU, METEOR, and ROUGE-L to compare generated summaries against reference summaries. When reference summaries are unavailable, human-generated summaries are sometimes used as benchmarks. An in-depth analysis of these metrics, particularly BLEU, investigated their correlation with human perception [11]. However, as Stapleton et al. [56] argued, these metrics often focused more on syntactic rather than semantic aspects, and did not fully capture

the impact of machine-generated code summaries on human comprehension or productivity. Moreover, this gap is evident in the work that highlights the limitations of these metrics in evaluating the creative and diverse outputs from models like ChatGPT [52]. SIDE [57] proposed a contrastive-learning-based metric that aligns code snippets with summaries, avoiding the drawbacks of reference-based evaluation. By learning from positive and negative pairs, it better correlates with human judgments, especially in semantic adequacy. However, it does not assess specific quality aspects such as coherence, fluency, or relevance. While human evaluations can address this gap, they are labor-intensive and rarely performed in practice.

C. Natural Language Generation Evaluators

Recent studies have explored using LLMs to evaluate generative tasks [13], [15], [58], [59]. DRPE [60] introduced roleplayer-based prompting for human-like reference-based evaluation, while AutoCalibrate [61] refined scoring via self-generated examples and feedback. Several works advocate reference-free evaluation: GEMBA [18] demonstrated GPT's strength in translation assessment, and G-EVAL [14] used CoT prompting for improved alignment with human judgment. ChatGPT's meta-evaluation [13] and guidelines for automatic use [15], [16] further support its evaluator role. Chan et al. [17] simulated annotator consensus with multiple LLM agents. In code tasks, ICE-Score [20] provided reference-free evaluation of code generation. To assess evaluator quality, Zeng et al. [19] built a robust benchmark, and Doostmohammadi et al. [62] examined evaluation reliability. Finally, Shankar et al. [63] proposed a mixed-initiative method to align LLM-based evaluations with human preferences.

Building on prior work, we advance the field in two key ways. First, unlike existing studies that focus on natural language summarization, we investigate LLMs as evaluators for code summarization—a domain that poses unique challenges such as understanding semantics, identifier roles, and structural intent. Second, we propose domain-specific, multi-role prompting strategies inspired by real-world software engineering roles (e.g., code reviewer, system analyst), and empirically examine their impact on evaluation reliability and alignment with human judgment. To our knowledge, this is the first systematic study of multi-role prompting for code summarization evaluation.

VII. CONCLUSION

In this paper, we have explored the capability of LLMs to evaluate code summarization models. We propose CODERPE, a novel LLM-based metric for assessing the quality of code summaries in four dimensions: coherence, consistency, fluency, and relevance. Particularly, we focus on the role-playing ability of LLMs, simulating various personas such as code reviewers, original code authors, code editors, and systems analysts. Moreover, we investigate the effects of prompt strategies and assess the robustness of the metrics. Experimental results reveal that our approach aligns more closely with human evaluations, presenting a promising alternative to traditional metrics.

Data Availability. All the source code and experimental data referenced in this paper can be accessed

at: <https://github.com/CGCL-codes/naturalcc/tree/main/examples/codesum-eval> [64].

ACKNOWLEDGEMENTS

This work is partially supported by the Major Program (JD) of Hubei Province (Grant No. 2023BAA024).

REFERENCES

- [1] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, "Summarizing source code using a neural attention model," in *54th Annual Meeting of the Association for Computational Linguistics 2016*. Association for Computational Linguistics, 2016, pp. 2073–2083.
- [2] A. LeClair, S. Jiang, and C. McMillan, "A neural model for generating natural language summaries of program subroutines," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 795–806.
- [3] 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 International Conference on Software Engineering*, 2024, pp. 1–13.
- [4] "Gpt-4 technical report," 2023.
- [5] G. Team, R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth et al., "Gemini: a family of highly capable multimodal models," *arXiv preprint arXiv:2312.11805*, 2023.
- [6] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.
- [7] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.
- [8] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.
- [9] 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*, J. Goldstein, A. Lavie, C.-Y. Lin, and C. Voss, Eds. Ann Arbor, Michigan: Association for Computational Linguistics, Jun. 2005, pp. 65–72.
- [10] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "Bertscore: Evaluating text generation with bert," *arXiv preprint arXiv:1904.09675*, 2019.
- [11] E. Shi, Y. Wang, L. Du, J. Chen, S. Han, H. Zhang, D. Zhang, and H. Sun, "On the evaluation of neural code summarization," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 1597–1608.
- [12] 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*, 2021, pp. 1105–1116.
- [13] J. Wang, Y. Liang, F. Meng, H. Shi, Z. Li, J. Xu, J. Qu, and J. Zhou, "Is chatgpt a good nlg evaluator? a preliminary study," *arXiv preprint arXiv:2303.04048*, 2023.
- [14] Y. Liu, D. Iter, Y. Xu, S. Wang, R. Xu, and C. Zhu, "G-eval: Nlg evaluation using gpt-4 with better human alignment, may 2023," *arXiv preprint arXiv:2303.16634*.
- [15] C.-H. Chiang and H.-y. Lee, "Can large language models be an alternative to human evaluations?" *arXiv preprint arXiv:2305.01937*, 2023.
- [16] C. Chiang and H. Lee, "A closer look into automatic evaluation using large language models," *arXiv preprint arXiv:2310.05657*, 2023.
- [17] C.-M. Chan, W. Chen, Y. Su, J. Yu, W. Xue, S. Zhang, J. Fu, and Z. Liu, "Chateval: Towards better llm-based evaluators through multi-agent debate," *arXiv preprint arXiv:2308.07201*, 2023.
- [18] T. Kocmi and C. Federmann, "Large language models are state-of-the-art evaluators of translation quality," *arXiv preprint arXiv:2302.14520*, 2023.
- [19] Z. Zeng, J. Yu, T. Gao, Y. Meng, T. Goyal, and D. Chen, "Evaluating large language models at evaluating instruction following," *arXiv preprint arXiv:2310.07641*, 2023.
- [20] T. Y. Zhuo, "Ice-score: Instructing large language models to evaluate code," in *Findings of the Association for Computational Linguistics: EACL 2024*, 2024, pp. 2232–2242.

- [21] “Chatgpt,” <https://openai.com/blog/chatgpt>, 2022.
- [22] “Gpt-4o-mini model documentation,” 2024, accessed: 2025-05-05. [Online]. Available: <https://platform.openai.com/docs/models/gpt-4o-mini>
- [23] “Gpt-4o model documentation,” 2024, accessed: 2025-05-05. [Online]. Available: <https://platform.openai.com/docs/models/gpt-4o>
- [24] DeepSeek-AI, “Deepseek-v3 technical report,” 2024. [Online]. Available: <https://arxiv.org/abs/2412.19437>
- [25] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, “Deep code comment generation,” in *Proceedings of the 26th Conference on Program Comprehension*, ser. ICPC ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 200–210.
- [26] J. Zhang, X. Wang, H. Zhang, H. Sun, and X. Liu, “Retrieval-based neural source code summarization,” in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 1385–1397.
- [27] W. U. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, “A transformer-based approach for source code summarization,” *arXiv preprint arXiv:2005.00653*, 2020.
- [28] M. Chen, J. Twarek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto *et al.*, “Evaluating large language models trained on code,” *CoRR*, vol. abs/2107.03374, 2021.
- [29] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, “Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation,” *arXiv preprint arXiv:2109.00859*, 2021.
- [30] Y. Wu, Y. Li, and S. Yu, “Commit message generation via chatgpt: How far are we?” in *Proceedings of the 2024 IEEE/ACM First International Conference on AI Foundation Models and Software Engineering*, ser. FORGE ’24. New York, NY, USA: Association for Computing Machinery, 2024, p. 124–129.
- [31] S. F. Chen and J. Goodman, “An empirical study of smoothing techniques for language modeling,” *Computer Speech & Language*, vol. 13, no. 4, pp. 359–394, 1999.
- [32] B. Wei, Y. Li, G. Li, X. Xia, and Z. Jin, “Retrieve and refine: exemplar-based neural comment generation,” in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020, pp. 349–360.
- [33] A. Bansal, S. Haque, and C. McMillan, “Project-level encoding for neural source code summarization of subroutines,” in *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. IEEE, 2021, pp. 253–264.
- [34] C. Lin, Z. Ouyang, J. Zhuang, J. Chen, H. Li, and R. Wu, “Improving code summarization with block-wise abstract syntax tree splitting,” in *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. IEEE, 2021, pp. 184–195.
- [35] R. Shahbazi, R. Sharma, and F. H. Fard, “Api2com: On the improvement of automatically generated code comments using api documentations,” in *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. IEEE, 2021, pp. 411–421.
- [36] 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.
- [37] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, “Chain-of-thought prompting elicits reasoning in large language models,” in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS ’22. Red Hook, NY, USA: Curran Associates Inc., 2022.
- [38] J. S. Park, J. O’Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, “Generative agents: Interactive simulacra of human behavior,” in *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023, pp. 1–22.
- [39] H. Chen, W. Ji, L. Xu, and S. Zhao, “Multi-agent consensus seeking via large language models,” *arXiv preprint arXiv:2310.20151*, 2023.
- [40] “Gpt-3.5 turbo model documentation,” 2022, accessed: 2025-05-05. [Online]. Available: <https://platform.openai.com/docs/models/gpt-3.5-turbo>
- [41] “Qwen3,” April 2025. [Online]. Available: <https://qwenlm.github.io/blog/qwen3/>
- [42] A. R. Fabbri, W. Kryściński, B. McCann, C. Xiong, R. Socher, and D. Radev, “Summeval: Re-evaluating summarization evaluation,” *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 391–409, 2021.
- [43] M. G. Kendall, “The treatment of ties in ranking problems,” *Biometrika*, vol. 33, no. 3, pp. 239–251, 1945.
- [44] E. R. Ziegel, “Standard probability and statistics tables and formulae,” *Technometrics*, vol. 43, no. 2, p. 249, 2001.
- [45] W. Wang, Y. Zhang, Y. Sui, Y. Wan, Z. Zhao, J. Wu, S. Y. Philip, and G. Xu, “Reinforcement-learning-guided source code summarization using hierarchical attention,” *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 102–119, 2020.
- [46] J. Guo, J. Liu, Y. Wan, L. Li, and P. Zhou, “Modeling hierarchical syntax structure with triplet position for source code summarization,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2022, pp. 486–500.
- [47] Y. Wan, Z. Zhao, M. Yang, G. Xu, H. Ying, J. Wu, and P. S. Yu, “Improving automatic source code summarization via deep reinforcement learning,” in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 397–407.
- [48] Y. Wan, Z. Bi, Y. He, J. Zhang, H. Zhang, Y. Sui, G. Xu, H. Jin, and P. Yu, “Deep learning for code intelligence: Survey, benchmark and toolkit,” *ACM Comput. Surv.*, vol. 56, no. 12, Oct. 2024.
- [49] H. Wu, H. Zhao, and M. Zhang, “Code summarization with structure-induced transformer,” *arXiv preprint arXiv:2012.14710*, 2020.
- [50] U. Alon, S. Brody, O. Levy, and E. Yahav, “code2seq: Generating sequences from structured representations of code,” *arXiv preprint arXiv:1808.01400*, 2018.
- [51] D. Gros, H. Sezhiyan, P. Devanbu, and Z. Yu, “Code to comment” translation” data, metrics, baselining & evaluation,” in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020, pp. 746–757.
- [52] J. Y. Khan and G. Uddin, “Automatic code documentation generation using gpt-3,” in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–6.
- [53] T. Ahmed, K. S. Pai, P. Devanbu, and E. Barr, “Automatic semantic augmentation of language model prompts (for code summarization),” in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.
- [54] C.-Y. Su and C. McMillan, “Distilled gpt for source code summarization,” *Automated Software Engineering*, vol. 31, no. 1, p. 22, 2024.
- [55] W. Sun, Y. Miao, Y. Li, H. Zhang, C. Fang, Y. Liu, G. Deng, Y. Liu, and Z. Chen, “Source code summarization in the era of large language models,” in *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 2024, pp. 419–431.
- [56] 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*, 2020, pp. 2–13.
- [57] A. Mastropaolo, M. Ciniselli, M. Di Penta, and G. Bavota, “Evaluating code summarization techniques: A new metric and an empirical characterization,” in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE ’24. New York, NY, USA: Association for Computing Machinery, 2024.
- [58] J. Fu, S.-K. Ng, Z. Jiang, and P. Liu, “Gptscore: Evaluate as you desire,” *arXiv preprint arXiv:2302.04166*, 2023.
- [59] D. Chen, R. Chen, S. Zhang, Y. Wang, Y. Liu, H. Zhou, Q. Zhang, Y. Wan, P. Zhou, and L. Sun, “Mllm-as-a-judge: Assessing multimodal llm-as-a-judge with vision-language benchmark,” in *Forty-first International Conference on Machine Learning*.
- [60] N. Wu, M. Gong, L. Shou, S. Liang, and D. Jiang, “Large language models are diverse role-players for summarization evaluation,” *arXiv preprint arXiv:2303.15078*, 2023.
- [61] Y. Liu, T. Yang, S. Huang, Z. Zhang, H. Huang, F. Wei, W. Deng, F. Sun, and Q. Zhang, “Calibrating LLM-based evaluator,” in *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, N. Calzolari, M.-Y. Kan, V. Hoste, A. Lenci, S. Sakti, and N. Xue, Eds. Torino, Italia: ELRA and ICCL, May 2024, pp. 2638–2656.
- [62] E. Doostmohammadi, O. Holmström, and M. Kuhlmann, “How reliable are automatic evaluation methods for instruction-tuned llms?” *arXiv preprint arXiv:2402.10770*, 2024.
- [63] S. Shankar, J. Zamfirescu-Pereira, B. Hartmann, A. Parameswaran, and I. Arawjo, “Who validates the validators? aligning llm-assisted evaluation of llm outputs with human preferences,” in *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST ’24. New York, NY, USA: Association for Computing Machinery, 2024.
- [64] Y. Wan, Y. He, Z. Bi, J. Zhang, Y. Sui, H. Zhang, K. Hashimoto, H. Jin, G. Xu, C. Xiong, and P. S. Yu, “Naturalcc: an open-source toolkit for code intelligence,” in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, ser. ICSE ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 149–153.