Exploring Large Language Models for Code Explanation

Paheli Bhattacharya[1,*,†], Manojit Chakraborty[1,†], Kartheek N S N Palepu[1], Vikas Pandey[1], Ishan Dindorkar[2], Rakesh Rajpurohit[2] and Rishabh Gupta[1]

[1]*Bosch Research and Technology Centre, Bangalore, India*
[2]*Bosch Global Software Technologies, Bangalore, India*

Abstract
Automating code documentation through explanatory text can prove highly beneficial in code understanding. Large Language Models (LLMs) have made remarkable strides in Natural Language Processing, especially within software engineering tasks such as code generation and code summarization. This study specifically delves into the task of generating natural-language summaries for code snippets, using various LLMs. The findings indicate that Code LLMs outperform their generic counterparts, and zero-shot methods yield superior results when dealing with datasets with dissimilar distributions between training and testing sets.

Keywords
Code Comment Generation, Code Summarization, Large Language Models, AI for Software Engineering

1. Introduction

Understanding legacy codes in large code repositories is a big challenge in the domain of software engineering. Liang et.al. [1] showed that only 15.4% of Java GitHub codes are documented. This makes it difficult and time-consuming for developers to comprehend the underlying functionality [2, 3]. Automating the task of code documentation through explanations can therefore prove beneficial.

Large Language Models (LLMs) have brought in a progressive breakthrough in Natural Language Processing, especially in the field of Generative AI. LLMs have been applied in many software engineering tasks [4], popularly in code generation [5], code summarization [6] and unit test case generation [7].

In this paper we focus on the task of code explanation – generating the intent or summary in natural-language for a given code snippet. We benchmark a suite of LLMs – both generic LLMs [8] and Code LLMs [9] using zero-shot, few-shot and instruction fine-tuning approaches. Extensive experiments on the IRSE dataset [10] leads to the following insights: (i) Code LLMs perform better than generic LLMs for the task. (ii) Zero-shot approaches achieve better results than few-shot and fine-tuning, where the train and test sets follow dissimilar distribution.

Forum for Information Retrieval Evaluation, December 15-18, 2023, India
[*]Corresponding author.
[†]Equal Contribution

paheli.bhattacharya@bosch.com (P. Bhattacharya)

Table 1
Example data points from IRSE and conala-train datasets along with their average lengths (#words).

Dataset	Size	Example Code Snippet	Example Code Explanation	Avg. length	
				Code	Comment
IRSE	100	pattern = re.compile('\\s+') sentence = re.sub(pattern, "", sentence)	This code snippet uses the re (regular expression) module in Python to define a pattern that matches one or more whitespace characters. It then uses the re.sub() function to remove any occurrences of the pattern from the string variable 'sentence'. The result is a modified version of 'sentence' with all whitespace characters removed.	21.18	84.28
conala-train	1666	re.sub('[^A-Z]', ", s)	remove uppercased characters in string 's'	13.92	14.68

2. Related Work

Code explanation [11], also termed as code summarization [3, 12] and comment generation [13, 2], is an important problem in the field of software engineering. Traditional approaches [14, 15, 16] as well as deep learning methods [13, 2] have been attempted for this task.

Large Language Models have been successfully employed in a wide variety of natural language generation tasks [17]. The zero shot and few shot capabilities of these systems make them highly adaptable to any NLP task. There are several general domain, open source LLMs like LLama-2 [8], Alpaca [18] and Falcon [19]. There are also Code LLMs which have been trained or finetuned on code-specific data (usually source code files, covering 80+ programming languages). The most popular LLMs for code are OpenAI CodeX and Co-pilot. Among the open source models, we have StarCoder [9], CodeUp [5], CodeLlama [20] and Llama-2-Coder [21].

Large Language Models have been used for **Code explanation** in a few shot setting [3, 22]. Ahmed et.al. [3] found that giving few shot examples from the same project gives better results than from a different project. Geng et.al. [22] show that selecting relevant examples in a few shot setting is an important design criteria.

3. Dataset

In this work, we consider a dataset of 100 samples released at the Information Retrieval in Software Engineering (IRSE) track at Forum for Information Retrieval Evaluation (FIRE) 2023 [10]. Each sample in the dataset is a $(code\ snippet, code\ explanation)$ pair. The explanation is a natural language description that denotes what task the code snippet is performing. We refer to this dataset as "IRSE" in the rest of the paper. Additionally, we use a publicly available conala-train [23] dataset as a secondary data source for few-shot and instruction finetuning. This dataset consists of 1666 unique samples of $(code\ snippet, code\ explanation)$ pairs.

Table 1 shows a few examples from both the datasets. It can be observed that while the code snippets in are comparable in length (21 and 14 tokens respectively), the code explanations in the IRSE dataset are lengthier (mean length = 84 words) than the ones in the conala-train set (mean length = 15 words).

4. Evaluation

The model generated textual descriptions are evaluated with respect to the ground truth explanations using the following measures:
(i) **Token-based**: BLEU [24] score combines precision scores of n-grams (typically up to 4-grams) using weighted geometric mean, with higher weight given to shorter n-grams. BLEU-1,

Table 2
Zero-shot prompt templates used for Code Explanation. $\{code\}$ denotes the query code snippet whose explanation needs to be generated.

#	Prompt	Models
P1	[INST] <> You are an expert in Programming. Below is a line of python code that describes a task. Return only one line of summary that appropriately describes the task that the code is performing. You must write only summary without any prefix or suffix explanations. Note: The summary should have minimum 1 words and can have on an average 10 words. <> {*code*} [/INST]	Llama-2-70B-Chat CodeLlama-13B-Instruct CodeUp-13B-Chat
P2	#Human: You are a helpful code summarizer. Please describe in simple english the purpose of the following Python code snippet: {*code*} #Assistant:	StarCoder (15.5B) Llama-2-Coder-7B

BLEU-2, and BLEU-N (for any integer N) extend the evaluation to unigrams, bigrams, and n-grams of varying lengths, respectively.
(ii) **Semantics-based**: We use this measure to assess the semantic similarity between the model generated explanation (m) and the ground truth explanation (g). We project both m and g in a continuous embedding space, $\overrightarrow{e_m}$ and $\overrightarrow{e_g}$ respectively using the pretrained CodeBERT [25] model. We then take a cosine similarity between the embeddings $cosine(\overrightarrow{e_m}, \overrightarrow{e_g})$ to get the score.

5. Methodology

We experiment with 5 LLMs (i) Generic LLM: Llama-2-70B-Chat model [8], which is the largest, open source model available. (ii) Code LLM – Llama-2-Coder-7B [21], CodeLlama-13B-Instruct [20], CodeUp-13B-Chat [5] and StarCoder [9] (15.5B) models, using the zero-shot, few-shot and instruction fine-tuning strategies, described below:
(i) Zero-shot: In this setting, we directly prompt the LLM to generate output for a particular input code snippet. We experiment with several prompts, some of which are listed in Table 2 as prompts P1 and P2. Based on the model cards, we provide the prompt template P1 to the Llama-2-70B Chat, CodeLlama-13B-Instruct and CodeUp-13B-Chat models. The template P2 is provided to StarCoder and Llama-2-Coder-7B models.
(ii) Few-shot: In few shot prompting, we provide a few examples that demonstrate the nature of the task. For the task of code explanation [3] suggest using 10 examples in a few-shot setup. Therefore, we provide 10 randomly selected $(code\ snippet, natural\ language\ description)$ pairs selected from the conala-train set (ref. Section 3).
(iii) Instruction Finetuning: For instruction finetuning of LLMs, we take CodeUp-13B-Chat model [5]. We take each sample from conala-train dataset and generate instruction based training instances using the following format:

Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.
### Instruction : Below is a line of python code that describes a task. Write one line of summary that appropriately describes the task that the code is performing.
### Input : $sorted(l, key = lambdax : (-int(x[1]), x[0]))$
### Output : Sort a nested list by two elements

Table 3
Performance evaluation of the different LLMs and three approaches for the code explanation task on the IRSE dataset. We report the BLEU and CodeBERT based metrics.

Approach	**LLM**	**Token-based**			**Semantics-based**
		BLEU1	BLEU2	BLEUN	CodeBERT
Zero Shot	Llama2-70B-Chat	0.019	0.008	0.004	0.338
	CodeLlama-13B-Instruct	**0.189**	0.073	**0.036**	**0.498**
	CodeUp-13B	0.010	0.003	0.001	0.310
	StarCoder-15.5B	0.069	0.024	0.005	0.336
	Llama-2-Coder-7B	**0.189**	**0.075**	0.023	0.475
Few Shot	Llama2-70B-Chat	0.064	0.024	0.012	0.424
	CodeLlama-13B-Instruct	**0.164**	0.073	0.044	**0.483**
	CodeUp-13B	0.061	0.023	0.011	0.416
	StarCoder-15.5B	0.020	0.006	0.002	0.347
	Llama-2-Coder-7B	0.023	**0.008**	0.003	0.342
Instruction Finetuning Zero Shot	CodeUp-13B	0.047	0.011	0.005	0.429

We load the CodeUp-13B-Chat model with 4-bit quantization using QLoRA [26] and bitsandbytes [27] methods. We then perform parameter-efficient finetuning (PEFT) [28] of the model using the above prepared dataset.

6. Results

Table 3 shows the performance of the 5 different LLMs over three approaches – zero-shot, few-shot and zero-shot over the Instruction finetuned model. CodeLlama-13B-Instruct and Llama-2-Coder-7B have the best zero-shot performance over the other LLMs. Note that although the generic Llama2 model is the largest in size (70B), it has poor performance when compared to the smaller Code LLM models (13B, 7B). This shows that domain specific models perform better than generic ones.

While the few shot strategy is expected to give better performance than zero-shot, in this study we find that the performance is worse. This is mainly because the few shot examples had been selected from the conala-train set. As discussed in Section 3 and Table 1 the code explanation lengths in the IRSE dataset and the conala-train dataset vary hugely. Since the LLMs see few shot examples from the conala-train, it generates shorter length code explanations for input samples coming from the IRSE dataset. This train-test distribution mismatch causes the models to perform worse in the few shot scenario as compared to the zero-shot.

Similar arguments can be drawn for the Instruction finetuning+Zero shot approach, as the training data comes from the conala-train dataset which is different from the IRSE dataset.

7. Conclusion

In this work we explore the performance of 5 LLMs, both generic and code-specifc, for the task of code explanation. We use zero-shot, few shot and instruction finetuning approaches over the LLMs and assess their performance. We find that Code LLMs perform better than larger generic LLMs. Also, zero-shot prompting works well in the scenario where we do not have enough examples to prompt/finetune the model.

References

[1] Y. Liang, K. Zhu, Automatic generation of text descriptive comments for code blocks, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 32, 2018.

[2] R. Sharma, F. Chen, F. Fard, Lamner: code comment generation using character language model and named entity recognition, in: Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension, 2022, pp. 48–59.

[3] T. Ahmed, P. Devanbu, Few-shot training llms for project-specific code-summarization, in: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, 2022, pp. 1–5.

[4] I. Ozkaya, Application of large language models to software engineering tasks: Opportunities, risks, and implications, IEEE Software 40 (2023) 4–8.

[5] J. Jiang, S. Kim, Codeup: A multilingual code generation llama2 model with parameter-efficient instruction-tuning, https://huggingface.co/deepse, 2023.

[6] M.-F. Wong, S. Guo, C.-N. Hang, S.-W. Ho, C.-W. Tan, Natural language generation and understanding of big code for AI-assisted programming: A review, Entropy 25 (2023) 888. URL: https://doi.org/10.3390%2Fe25060888. doi:10.3390/e25060888.

[7] M. Schäfer, S. Nadi, A. Eghbali, F. Tip, An empirical evaluation of using large language models for automated unit test generation, 2023. arXiv:2302.06527.

[8] 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). URL: https://huggingface.co/meta-llama/Llama-2-70b-chat-hf.

[9] R. L. et.al., Starcoder: may the source be with you!, arXiv preprint arXiv:2305.06161 (2023). URL: https://huggingface.co/bigcode/starcoder.

[10] S. Majumdar, S. Paul, D. Paul, A. Bandyopadhyay, B. Dave, S. Chattopadhyay, P. P. Das, P. D. Clough, P. Majumder, Generative ai for software metadata: Overview of the information retrieval in software engineering track at fire 2023, in: Forum for Information Retrieval Evaluation, ACM, 2023.

[11] S. MacNeil, A. Tran, A. Hellas, J. Kim, S. Sarsa, P. Denny, S. Bernstein, J. Leinonen, Experiences from using code explanations generated by large language models in a web software development e-book, in: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, 2023, pp. 931–937.

[12] S. Iyer, I. Konstas, A. Cheung, 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.

[13] X. Hu, G. Li, X. Xia, D. Lo, Z. Jin, Deep code comment generation, in: Proceedings of the 26th Conference on Program Comprehension, Association for Computing Machinery, 2018, p. 200–210.

[14] S. Haiduc, J. Aponte, L. Moreno, A. Marcus, On the use of automated text summarization techniques for summarizing source code, in: 2010 17th Working conference on reverse engineering, IEEE, 2010, pp. 35–44.

[15] B. P. Eddy, J. A. Robinson, N. A. Kraft, J. C. Carver, Evaluating source code summarization techniques: Replication and expansion, in: 2013 21st International Conference on Program

Comprehension (ICPC), IEEE, 2013, pp. 13–22.

[16] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, K. Vijay-Shanker, Automatic generation of natural language summaries for java classes, in: 2013 21st International conference on program comprehension (ICPC), IEEE, 2013, pp. 23–32.

[17] J. Yang, H. Jin, R. Tang, X. Han, Q. Feng, H. Jiang, B. Yin, X. Hu, Harnessing the power of llms in practice: A survey on chatgpt and beyond, arXiv preprint arXiv:2304.13712 (2023).

[18] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, T. B. Hashimoto, Alpaca: A strong, replicable instruction-following model, Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html 3 (2023) 7.

[19] G. Penedo, Q. Malartic, D. Hesslow, R. Cojocaru, A. Cappelli, H. Alobeidli, B. Pannier, E. Almazrouei, J. Launay, The RefinedWeb dataset for Falcon LLM: outperforming curated corpora with web data, and web data only, arXiv preprint arXiv:2306.01116 (2023).

[20] B. Rozière, 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). URL: https://huggingface.co/codellama.

[21] Manuel Romero, llama-2-coder-7b (revision d30d193), 2023. URL: https://huggingface.co/mrm8488/llama-2-coder-7b. doi:10.57967/hf/0931.

[22] M. Geng, S. Wang, D. Dong, H. Wang, G. Li, Z. Jin, X. Mao, X. Liao, Large language models are few-shot summarizers: Multi-intent comment generation via in-context learning (2024).

[23] P. Yin, B. Deng, E. Chen, B. Vasilescu, G. Neubig, Learning to mine aligned code and natural language pairs from stack overflow, in: International Conference on Mining Software Repositories, ACM, 2018, pp. 476–486. URL: https://conala-corpus.github.io/.

[24] K. Papineni, S. Roukos, T. Ward, W.-J. Zhu, Bleu: A method for automatic evaluation of machine translation, Association for Computational Linguistics, USA, 2002, p. 311–318.

[25] 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, 2020, pp. 1536–1547.

[26] T. Dettmers, A. Pagnoni, A. Holtzman, L. Zettlemoyer, Qlora: Efficient finetuning of quantized llms, 2023. arXiv:2305.14314.

[27] T. Dettmers, M. Lewis, S. Shleifer, L. Zettlemoyer, 8-bit optimizers via block-wise quantization, 9th International Conference on Learning Representations, ICLR (2022).

[28] S. Mangrulkar, S. Gugger, L. Debut, Y. Belkada, S. Paul, B. Bossan, Peft: State-of-the-art parameter-efficient fine-tuning methods, https://github.com/huggingface/peft, 2022.