Exploring LLM for Code Explanation

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.

1. Introduction:

Examine Larger Language Models (LLMs) and evaluate their effectiveness in summarising provided code snippets, specifically focusing on their capacity for code explanation. Investigate the applicability of **zero-shot and many-shot** learning techniques within this context. Additionally, explore the distinctions between **code-specific and generic LLMs**. Furnish a comprehensive understanding of the chosen AI tool, elucidating the rationale behind its selection and its broader significance in the field.

2. Datasets Viewed:

- ### IRSE

  Information Retrieval in Software Engineering (IRSE) track at Forum for Information Retrieval Evaluation (FIRE) 2023. 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" .

- ### Conola-train

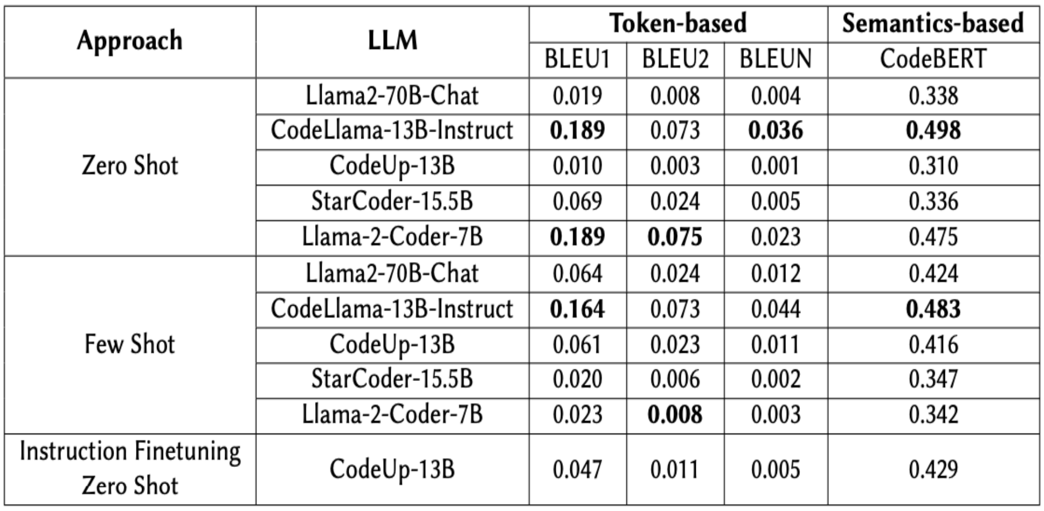
  publicly available conala-train dataset as a data source for few-shot and instruction finetuning. This dataset consists of 1666 unique samples of (*code snippet, code explanation*) pairs.

- ### CodeXGLUE

  CodeXGLUE code summarization benchmark. It should be noted that this dataset is unrelated to the Codex model. CodeXGLUE is originally adapted from the CodeSearchNet dataset. It’s multilingual, with data from six different languages (i.e., Ruby, JavaScript, Java, Go, PHP, Python). Quite a number of papers using the foundation models have been evaluated on this dataset for the code summarization task; so it constitutes a good benchmarks.

3. Models Observed:

- Llama2-70B-Chat
- Code Llama-13B-Instruct
- Llama-2-Coder-7B
- CodeUP-13B
- StarCoder-15.5B



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

This table has been taken from https://arxiv.org/pdf/2310.16673v1.pdf

The table illustrates the performance of five distinct Language Model Models (LLMs) across three methodologies: zero-shot, few-shot, and zero-shot applied to the Instruction fine-tuned model. Notably, CodeLlama-13B-Instruct and Llama-2-Coder-7B exhibit superior zero-shot performance compared to other LLMs. It is noteworthy that, despite being the largest in size (70B), the generic Llama2 model demonstrates poorer performance when juxtaposed with the smaller Code LLM models (13B, 7B). This observation underscores the superiority of domain-specific models over generic counterparts.

4. Code LLMs v/s Generic LLMs

Code LLM:

LLM like OpenAI CodeX, CoPilot these are famous and these are code LLM which means they are code specific and they are not based on guessing the next wods based on previous rather they depend on the code and answer it.

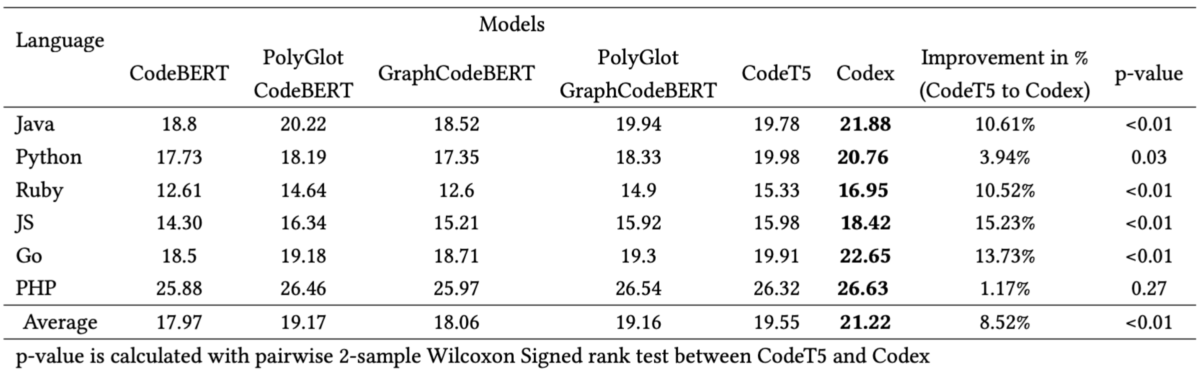
Generic LLM:

A generic Language Model (LLM) refers to a language model that is trained on a diverse range of textual data without specific domain or task-oriented fine-tuning. It is designed to understand and generate human-like text across various topics and contexts. Generic LLMs are pre-trained on large datasets containing a broad spectrum of language, including general knowledge, literature, news, and more.

On experimenting on IRSE dataset we get insights that Code LLM's perform better than generic LLM's.

5. Zero Shot v/s Few Shot

For Zero shot Llama2 models shown good results but once we try few shot Codex models are best they outperform many SOTA models for a 10-shot and the results are based on research paper https://arxiv.org/pdf/2207.04237v2.pdf.



Language	Models							
	CodeBERT	PolyGlot CodeBERT	GraphCodeBERT	PolyGlot GraphCodeBERT	CodeT5	Codex	Improvement in % (CodeT5 to Codex)	p-value
Java	18.8	20.22	18.52	19.94	19.78	**21.88**	10.61%	<0.01
Python	17.73	18.19	17.35	18.33	19.98	**20.76**	3.94%	0.03
Ruby	12.61	14.64	12.6	14.9	15.33	**16.95**	10.52%	<0.01
JS	14.30	16.34	15.21	15.92	15.98	**18.42**	15.23%	<0.01
Go	18.5	19.18	18.71	19.3	19.91	**22.65**	13.73%	<0.01
PHP	25.88	26.46	25.97	26.54	26.32	**26.63**	1.17%	0.27
Average	17.97	19.17	18.06	19.16	19.55	**21.22**	8.52%	<0.01

p-value is calculated with pairwise 2-sample Wilcoxon Signed rank test between CodeT5 and Codex

Table 1: Comparison to existing models, on CodeXGLUE dataset

With 10 samples, Codex outperforms all fine tuned foundation models CodeT5, CodeBERT, GraphCodeBERT, Polyglot CodeBERT, and PolyGlotGraphCodeBERT in all six programming languages, even though the fine-tuned models are trained with thousands of data.

Zero-shot and one-shot training in Codex do not work for code summarization task.

6. Conclusion

Zero-shot prompting proves effective in situations where an insufficient number of examples are available for prompting or fine-tuning. Code summarization shares significant similarities with Neural Machine Translation (NMT), such as English to German translation, where a Language Model (LLM), specifically a decoder-only model, is employed rather than encoder-decoder models for the task.
Utilising LLM for code summarization presents fewer risks compared to code generation, as the summarization involves comments that are not executed as part of the program.