XLCoST: A Benchmark Dataset for Cross-lingual Code Intelligence

Ming Zhu, Aneesh Jain, Karthik Suresh, Roshan Ravindran, Sindhu Tipirneni & Chandan K. Reddy
Department of Computer Science, Virginia Tech, Arlington, VA
{mingzhu, aneeshj, karthiks, roshan14, tsaisindhura}@vt.edu
reddy@cs.vt.edu

Abstract

Recent advances in machine learning have significantly improved the understanding of source code data and achieved good performance on a number of downstream tasks. Open source repositories like GitHub enable this process with rich unlabeled code data. However, the lack of high quality labeled data has largely hindered the progress of several code related tasks, such as program translation, summarization, synthesis, and code search. This paper introduces $XLCoST$, **Cross-L**ingual **Co**de **S**nippe**T** dataset, a new benchmark dataset for cross-lingual code intelligence. Our dataset contains fine-grained parallel data from 8 languages (7 commonly used programming languages and English), and supports 10 cross-lingual code tasks. To the best of our knowledge, it is the largest parallel dataset for source code both in terms of size and the number of languages. We also provide the performance of several state-of-the-art baseline models for each task. We believe this new dataset can be a valuable asset for the research community and facilitate the development and validation of new methods for cross-lingual code intelligence. [1].

1 Introduction

Recent advances in machine learning have benefited a number of code related tasks, such as code translation, code summarization, and code synthesis. Open-source code repository websites like Github provide enormous amount of source code data, which enables the training of large-scale programming language models such as CodeBERT (Feng et al., 2020), PLBART (Ahmad et al., 2021a), TransCoder (Roziere et al., 2020) and CodeT5 (Wang et al., 2021). These extensively pre-trained models have shown superior performance on benchmark datasets like CodeXGLUE (Lu et al., 2021).

Although open-source code data is abundant in quantity, it has several disadvantages when being used as training data for code-related models. First, most of the available code data is unlabeled. For tasks like Code Translation, Code Summarization, and Code Synthesis, high quality parallel data is critical for model training. However, it is difficult to mine parallel data from open-source projects. Second, labeled data is usually small in size. For example, the code translation data introduced in Zhu et al. (2022) only has around 70 programs for testing and 50 programs for validation. Due to the small size of evaluation data, the models trained on this dataset may not be thoroughly evaluated. Moreover, the available labeled datasets usually only cover a limited number of languages. For example, the Code Translation dataset in CodeXGLUE only covers 2 languages, Java and C#. Because of the scarcity of labeled data in some programming languages, code tasks in some low-resource languages remain unexplored.

In this paper, we introduce $XLCoST$, a machine learning benchmark dataset that contains fine-grained parallel data in 7 commonly used programming languages (C++, Java, Python, C#, Javascript, PHP, C), and natural language (English). The data is parallel across 7 languages, at both code snippet level and program level. This means that, given a program in one language, the dataset contains the same program in up to 6 other programming languages. Each program is divided into several code snippets, and programs in all the languages are aligned at the snippet level. Moreover, each of the

[1] https://github.com/reddy-lab-code-research/XLCoST

Table 1: Comparison against other parallel code datasets (Py - Python, JS - JavaScript). Column "Size" refers to the number of parallel data pairs. *This number is for single programs, not pairs.

Dataset	Alignment	Task	Labelling	Size	Languages
CodeNet	Program	Multiple	Solutions to the same problem	13.9M*	55 programming languages
AVATAR	Program	Translation	Solutions to the same problem	57,414	Java, Py
CodeXGLUE	Method	Multiple	Matching function names	11,800	Java, C#
CoST	Snippet	Translation	Matching code comments	132,046	C++, Java, Py, C#, JS, PHP, C
XLCoST	Snippet	Multiple	Matching code comments	1,002,296	C++, Java, Py, C#, JS, PHP, C, English

snippets is accompanied with a comment, and the comment for a particular snippet is the same across all the languages. Table 1 presents a comparative analysis of $XLCoST$ in terms of the number of available parallel data samples against other widely used parallel code datasets. The dataset contains around 1 million parallel snippets and 123K parallel programs in total, which is significantly larger than many available parallel code datasets. We believe that this dataset is a valuable asset for the research community and can potentially benefit a number of code-related research problems.

To further facilitate the development and evaluation of models with a focus on source code, we also introduce 10 different cross-lingual tasks. These tasks can be divided into two categories: Generation and Retrieval. The generation tasks include Code Translation (Code-to-Code), Code Summarization (Code-to-Text), and Code Synthesis (Text-to-Code); the retrieval tasks include NL (Natural Language) Code Search and XL (Cross-Lingual) Code Search. Each task is at both snippet and program level.

To evaluate how challenging the tasks are with the proposed dataset, we run experiments on all the 10 tasks with a number of state-of-the-art baseline models. We also conduct an empirical study to understand how the model design relates with the performance on different tasks with $XLCoST$ dataset. The primary contributions of this paper are as follows:

- We introduce a new dataset which is parallel across 8 languages (7 programming languages and English) at both snippet level and program level. To the best of our knowledge, it is the largest **parallel** dataset for source code in both size and number of languages.
- We formulate 10 different cross-lingual tasks to facilitate the development and evaluation of models in this domain.
- We run experiments for all the 10 tasks on the proposed dataset with a number of state-of-the-art baseline models and provide insights about model design for the new challenges.

2 THE $XLCoST$ DATASET

The data for $XLCoST$ was collected from GeeksForGeeks[2], which is a website that houses thousands of data structures and algorithm problems along with solutions in up to 7 different programming languages - C++, Java, Python, C#, Javascript, PHP, and C. According to GeeksForGeeks, the solution programs for the same problem follow the same structure, down to the variable names. This results in the programs being semantically consistent across the different languages. In most cases, the programs for the same problem share the same set of comments in the same order, which indicates that they are parallel to the snippet level. This is where the fine-grained alignment in $XLCoST$ comes from.

2.1 DEFINITIONS

Problems: The problems are mostly about data structures and algorithms, as they are mainly designed for tutoring and coding interview preparation. Each problem has programs as solutions in up to 7 programming languages.
Programs: A program is a solution to a problem in a specific programming language. Each problem in this dataset may contain up to 7 programs (one for each language). The programs for the same problem share similar logic and structure.
Snippets: The code between two consecutive comments in a program is termed as a snippet (code

[2]https://www.geeksforgeeks.org/

Table 2: The train-valid-test split and basic statistics of $XLCoST$ data. SN - Snippets; PR - Program.

	Snippet-level								Program-level							
Split	**C++**	**Java**	**Py**	**C#**	**JS**	**PHP**	**C**	**Total**	**C++**	**Java**	**Py**	**C#**	**JS**	**PHP**	**C**	**Total**
train	93847	91089	81207	87583	70649	18027	3763	446165	9797	9623	9263	9345	8590	3087	463	50168
valid	4432	4460	3946	4436	3829	930	350	22383	492	494	472	491	475	158	60	2642
test	8118	8154	7293	8013	7033	1682	250	40543	909	911	887	899	886	308	51	4851
total	106397	103703	92446	100032	81511	20639	4363	509091	11198	11028	10622	10735	9951	3553	574	57661
Stats	**C++**	**Java**	**Py**	**C#**	**JS**	**PHP**	**C**	**Avg.**	**C++**	**Java**	**Py**	**C#**	**JS**	**PHP**	**C**	**Avg.**
# lines/code	3.41	3.71	2.41	3.82	3.23	4	4.05	3.37	32.45	34.93	20.54	35.64	26.47	23.23	31.5	29.71
# tokens/code	21.52	24.1	21.63	23.06	22.52	28.14	25.37	22.83	205	227.1	188.5	215.3	184.6	163.5	198	202
# tokens/text	8.25	8.14	7.97	8.23	7.96	8.45	9.67	8.15	10.68	10.67	10.75	10.7	10.87	9.91	8.19	10.66
# SN/PR	–	–	–	–	–	–	–	–	9.52	9.42	8.51	9.33	8.2	5.81	7.77	8.81

before the first comment and after the last comment are also included). On an average, each program contains 8.81 snippets.
Description: Each problem also has a short description, for example, "Maximum Consecutive Increasing Path Length in Binary Tree."
Comments: The comments in each program in this dataset. The programs are well commented and each program has an average of around 9 comments.

2.2 Data Characteristics

The final dataset consists of 11,265 programming problems. As shown in Table 2, there are 57,661 unique programs. Each program consists of 8.81 snippets on average, which results in 509,091 snippets. A detailed statistics table for the translation task is available in Appendix A.2.

Multilingual: The dataset contains parallel data in 8 languages (7 commonly used programming languages and English).
Parallel: The dataset contains 4 types of parallel data, snippet-to-snippet, program-to-program, snippet-to-comment, program-to-problem (and comments) which further enables 10 different tasks.
Finely-aligned: The data is parallel at both snippet level and program level. To the best of our knowledge, this dataset is the finest-aligned among parallel code datasets.
Large: It is the largest parallel dataset for source code in terms of both size and number of languages.
Simple: Each program in this dataset is standalone without dependency on other programs. It ensures that the complexity of the tasks is controllable.

2.3 Data Collection and Processing

The data was scraped from different sub-pages of the GeeksForGeeks website. A majority of the problems on this site fall under two categories - Data Structures and Algorithms. More details are included in Appendix A.3. The IP policies and regulations for GeeksForGeeks were carefully followed and we confirm that no data privacy policy was violated when collecting the data.

After collecting the data, we first removed duplicate problems, as some problems might be presented in multiple subcategories. Then we extracted problem description and solution programs in each available language from the page. Each program was sliced into code snippets by splitting at the comments, after which the comments and docstrings were removed from the programs. Any personal information such as the name of the code's contributor, was also removed from both the comments and the codes at this time. Eventually, we get 4 types of information from one page: 1) Problem Description; 2) Parallel programs in different languages; 3) Code Snippets; 4) Code Comments.

2.3.1 Data Alignment

The snippet-level alignment was done by matching comments in the solution programs (for the same problem) across different languages. As mentioned earlier, GeeksForGeeks programs follow a standard template, because of which the comments in different language programs (for the same problem) align parallelly in most cases. This yields parallel snippets that have the same functionality across different languages.

Table 3: An overview of the tasks. All the tasks have pairwise data at both snippet-level and program-level in 7 programming languages, C++, Java, Python, C#, Javascript, PHP, and C. The tasks can be divided into two categories, generation and retrieval. The generation tasks include Code Translation, Code Summarization and Code Syntheis; the retrieval tasks include NL (natural language) Code Search and XL (Cross-Lingual) Code Search.

Category		Task	Data	Description
Generation	Code Translation (Code-to-Code)	Snippet Translation	872K/47K/83K	Translate code snippet across programming languages
		Program Translation	106K/6K/11K	Translate program across programming languages
	Code Summarization (Code-to-Text)	Snippet Summarization	446K/22K/41K	Generate comment for given code snippet
		Program Summarization	50K/3K/5K	Generate problem description for given program
	Code Synthesis (Text-to-Code)	Snippet Synthesis	446K/22K/41K	Generate code snippet giving comment
		Program Synthesis	50K/3K/5K	Generate program giving problem description and comments
Retrieval	NL Code Search	Comment-to-Snippet Search	446K/22K/41K	Retrieve code snippet for given comment
		Problem-to-Program Search	50K/3K/5K	Retrieve program for given problem description
	XL Code Search	Snippet-to-Snippet Search	872K/47K/83K	Retrieve code snippets in other languages for given snippet
		Program-to-Program Search	106K/6K/11K	Retrieve programs in other languages for given snippet

Misalignment detection: In some cases, the comments in different solution programs are not aligned. The misalignment can come from different numbers of comments, and the differences in the comment content. This is usually due to some solution program not strictly following the guidelines and templates. For solution programs with the same number of comments, we evaluate the alignment by calculating the average similarity score of each pair of comments in the two programs (using Python *difflib.SequenceMatcher*[3])). If the average score is below a certain threshold (80% in our case), it would be categorized as misalignment and manual checking would be needed. Solution programs with different number of comments were automatically categorized as misaligned and sent for manual checking.

Manual checking and aligning: Manual checking was performed by two of the authors with good knowledge of programming languages and their functionalities. Based on the differences in number of comments, the misaligned programs were split into the following categories:
Category 0: The programs have the same number of comments. The type of misalignment usually only is due to different wording in the comments and can be easily fixed.
Category k: The difference in number of comments is k. When $k < 3$, extra comments needed to be discarded in some cases and code from these comments was moved to appropriate snippets to preserve the alignment with other languages. In some cases, there were also missing comments which had to be added along with the moving of the appropriate code block as in the previous case. When $k >= 3$, the programs will be discarded.

2.3.2 DATA SPLITTING

Since the parallel programs are within each problem, splitting the data at problem level can naturally avoid data leakage. However, during the data processing, we noticed that some problems are very similar. For example, "Check if a large number is divisible by 3 or not" and "Check whether a large number is divisible by 53 or not". If one problem goes to the training set and the other goes to the test set, it can lead to potential data leakage and bias. To address this concern, we first clustered all the similar problems into groups, and make the split at the group-level. In this way, we can ensure that similar problems go to the same split. To do so, we first calculate the similarity score (using Python *difflib.SequenceMatcher*) between every two pairs of problem descriptions, and group all the problems using various similarity score thresholds (60%-80%) based on length of the descriptions. The final split ratio in the data is around 85-5-10 for train-validation-test sets. The detailed steps for data splitting are included in Appendix A.4.

[3]https://docs.python.org/3/library/difflib.html

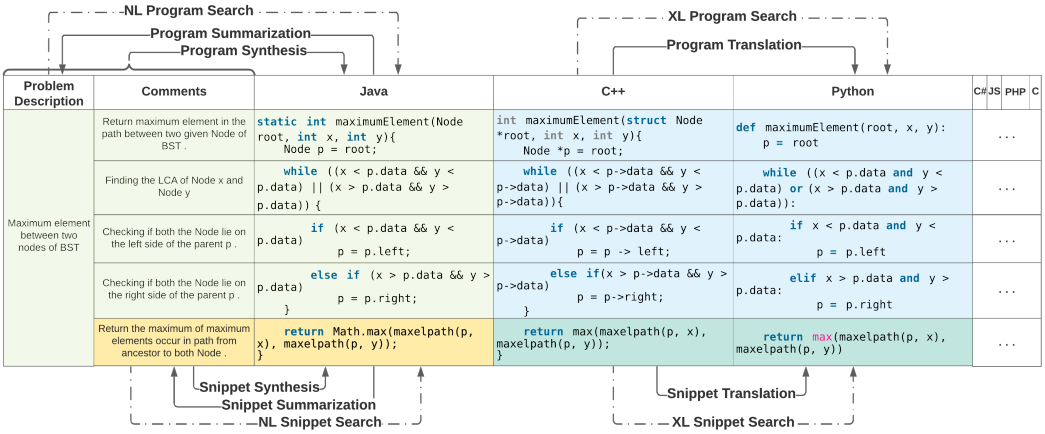

Figure 1: An illustration of the data and the tasks. The first column is the Problem Description; each cell in the second column is a Comment; each cell from the third column is a code Snippet. The combination of all the code snippets in a column is a Program (truncated due to space limitation). The arrows show the input and output data for each task. Solid lines are for generation tasks and dashed lines are for retrieval tasks. Note that the Program Synthesis task uses both Problem Description and Comments as input.

3 CODE TASKS

The tasks can be divided into two categories: generation and retrieval. The generation tasks include Code Translation, Code Summarization, and Code Synthesis. The retrieval tasks include NL (natural language) Code Search and XL (Cross-Lingual) Code Search. All the tasks are at both snippet-level and program-level. Figure 1 shows the input and output data for each of the tasks. Table 3 summarizes all the tasks introduced and some aggregate data statistics corresponding to each task.

Code Translation (Code-to-Code): Code Translation is the problem of converting source code from one programming language to another. Efficient and accurate code translation is valuable in scenarios like legacy code migration, software platform adaptation, etc. The proposed $XLCoST$ dataset provides parallel data in 7 common programming languages, supporting translation for 42 language pairs at both snippet and program level.

Code Summarization (Code-to-Text): The objective of Code Summarization task is to generate natural language descriptions of the code that is given as input. We perform this task under two settings, generating snippet level summary by leveraging the comment-snippet pairings, and generating problem level summary using the problem description and program code pairings. Applications of this task include increasing the comprehensibility of uncommented or unfamiliar code to first time viewers and making it easier to collaborate as well as educate.

Code Synthesis (Text-to-Code): The Code Synthesis task focuses on generating source code from text inputs. It includes Snippet Synthesis and Program Synthesis. We use the comment of each code snippet as input to generate the code snippet for the Snippet Synthesis task, since they are of similar length (as shown in Table 2). However, programs are usually much longer (Avg. 202 tokens) than problem descriptions (Avg. 11 tokens). To generate programs, it is necessary that the input text is detailed and informative. Therefore, we use a combination of problem description and step-by-step comments as input to generate the entire program. Since the programs in $XLCoST$ are well commented (9 comments/snippets per program on an average) this ensures that the models have enough information to synthesize the whole program.

Code Search: The NL (Natural Language) Code Search in this paper refers to using text input to retrieve relevant code. The snippet and program level task use Comment and Problem Description as query, respectively. XL (Cross-lingual) Code Search is the task of retrieving code that performs similar functions in multiple other languages given a piece of code in one particular language. Unlike

NL code search, using code as queries to search for similarly functioning code in a multilingual setting is relatively unexplored task. This task also includes both snippet and program level. To account for multiple correct answers, we use a modified *MRR* (Mean Reciprocal Rank) for evaluation (details in Appendix A.6).

4 EXPERIMENTS

All the baselines were initialized with the pretrained weights and default configuration (including hyper-parameters) released by the corresponding original authors of the works. We changed the source and target sequence lengths to align with the dataset based on the task. The models were trained using 4 RTX 8000 GPUs with 48GB memory on each GPU. The code for training and evaluation is released in the GitHub repository of the dataset.

4.1 EVALUATION METRICS AND BASELINES

We use the following metrics to evaluate different tasks proposed in this work: (i) BLEU (Papineni et al., 2002) score to evaluate code-to-text generation tasks;, (ii) BLEU and CodeBLEU[4] (Ren et al., 2020) to evaluate code-to-code and text-to code generation tasks, and (iii) Mean Reciprocal Rank (MRR) to evaluate retrieval tasks.
We use the following models/methods for our comparison:
Naive Copy Lu et al. (2021) directly copies the input source code as the output, which shows how similar two programming languages are. It is only used for translation tasks.
RoBERTa (Liu et al., 2019) is a robustly optimized version of BERT pretrained on huge natural language corpora. We use it only for retrieval tasks.
CodeBERT (Feng et al., 2020) uses the BERT (Devlin et al., 2019) architecture pretrained on CodeSearchNet (Husain et al., 2019) data. We use the encoder-only version for retrieval tasks and encoder-decoder version (the decoder is randomly initialized) for generation tasks.
PLBART (Ahmad et al., 2021a) is initialized with mBART (Liu et al., 2020) and further pretrained on a large-collection of Java and Python functions and natural language descriptions from Github and StackOverflow with denoising auto-encoding objective.
CodeT5 (Wang et al., 2021) employs T5 (Raffel et al., 2020) architecture and is pretrained on corpora of 8 programming languages (Java, Python, C#, JS, PHP, C, Ruby, Go) with identifier-aware objective.

4.2 RESULT ANALYSIS

Table 4 shows the performance of baseline models for Code Translation, Code Synthesis, Code Summarization, and Code Search tasks.

Effect of Sequence-to-Sequence Pretraining: In Table 4, on an average, CodeBERT performs significantly worse than PLBART and CodeT5 on almost all the generation tasks (refer to the first three sections of the table). Different from PLBART and CodeT5, which are both encoder-decoder models pretrained with sequence-to-sequence objectives, only the encoder in CodeBERT is pretrained, and the decoder weights are randomly initialized for sequence-to-sequence tasks. Experimental results show that encoder-decoder architecture and sequence-to-sequence pretraining are better aligned with generation tasks and thus can potentially achieve superior performance.

Effect of Pretraining on Specific Languages: CodeBERT is pretrained on CodeSearchNet, which contains data from 6 programming languages, Java, Python, Javascript, PHP, Ruby, and Go. PLBART is pretrained on Java and Python from GitHub data. CodeT5 is trained on the 6 languages from CodeSearchNet and additional C and C#. In Table 4, CodeT5 consistently outperforms the other two models for almost all generation tasks. When the source or target language is C, CodeT5 outperforms the other two by a wide margin. Pre-training on specific languages can potentially benefit the generation tasks with these languages as either input or output.

Performance on Low-Resource Languages: In Table 4, most models performs significantly worse on C compared to other languages, both when C is source or target language, in almost all the tasks (except for Code Search). As shown in Table 2, C has the least number of samples for all the tasks. It shows that tasks in low-resource languages are potentially more challenging.

[4]We extended the CodeBLEU metric to support C and C++. Related code is released in the GitHub repo.

Table 4: From top to bottom, the table contains results for Code Translation, Code Synthesis, Code Summarization, and Code Search at the snippet-level and program-level. CodeBLEU scores are reported for Code Generation tasks (Translation and Synthesis). For Translation, the language column on the left represents the source language and the row on the top represents the target language. BLEU scores are reported for Summarization and MRR for Search.

		Snippet-level							**Program-level**						
CodeBLEU	**Model**	**C++**	**Java**	**Py**	**C#**	**JS**	**PHP**	**C**	**C++**	**Java**	**Py**	**C#**	**JS**	**PHP**	**C**
C++	Naive Copy	–	64.56	34.79	63.19	53.16	42.56	84.2	–	57.36	17.68	58.02	53.16	18.97	75.91
	CodeBERT	–	84.94	74.55	84.99	82.79	68.56	45.46	–	74.73	24.96	76.35	72.95	50.4	21.84
	PLBART	–	83.85	74.89	84.57	83.19	68.62	83.95	–	75.26	70.13	78.01	61.85	67.01	72.59
	CodeT5	–	**86.35**	**76.28**	**85.85**	**84.31**	**69.87**	**90.45**	–	**80.03**	**71.56**	**81.73**	**79.48**	**70.44**	**85.67**
Java	Naive Copy	70.85	–	35	78.43	57.81	42.49	69.74	64.25	–	39.87	72.68	57.81	42.51	62.48
	CodeBERT	87.27	–	58.39	92.26	84.63	67.26	39.94	79.36	–	8.51	84.43	76.02	51.42	21.22
	PLBART	87.31	–	58.3	90.78	85.42	67.44	72.47	81.41	–	66.29	83.34	80.14	67.12	63.37
	CodeT5	**88.26**	–	**74.59**	**92.56**	**86.22**	**69.02**	**82.78**	**84.26**	–	**69.57**	**87.79**	**80.67**	**69.44**	**78.78**
Python	Naive Copy	39.22	31.89	–	31.79	38.34	36.02	37.79	37.47	29.78	–	27.59	38.42	35.48	35.66
	CodeBERT	80.46	58.5	–	54.72	57.38	65.14	10.7	68.87	28.22	–	17.8	23.65	49.3	18.32
	PLBART	80.15	74.15	–	73.5	73.2	66.12	62.15	74.38	67.8	–	66.03	69.3	64.85	29.05
	CodeT5	**81.56**	**78.61**	–	**78.89**	**77.76**	**67.54**	**68.67**	**78.85**	**73.15**	–	**73.35**	**71.8**	**67.5**	**56.35**
C#	Naive Copy	69.78	78.71	34.77	–	57.85	42.53	66.73	64	73.63	40.09	–	57.79	42.96	60.87
	CodeBERT	86.96	90.15	56.92	–	84.38	67.18	40.43	78.52	82.25	10.82	–	75.46	51.76	21.63
	PLBART	84.98	6.27	69.82	–	85.02	67.3	75.74	80.17	81.37	67.02	–	79.81	67.12	57.6
	CodeT5	**88.06**	**91.69**	**73.85**	–	**85.95**	**68.97**	**81.09**	**83.59**	**85.7**	**69.52**	–	**80.5**	**69.63**	**77.35**
JS	Naive Copy	60.82	59.25	38.84	64.27	–	41.56	55.84	53.81	51.77	42.31	54.86	–	42.11	49.04
	CodeBERT	84.38	84.42	52.57	84.74	–	66.66	33.29	75.43	72.33	9.19	75.47	–	52.08	19.79
	PLBART	84.45	84.9	69.29	85.05	–	67.09	72.65	80.19	76.96	64.18	78.51	–	67.24	67.7
	CodeT5	**85.06**	**85.48**	**73.15**	**85.96**	–	**68.42**	**80.49**	**82.14**	**79.91**	**68.42**	**81.77**	–	**68.76**	**74.57**
PHP	Naive Copy	36.33	35.61	24.62	36.67	35.55	–	35.95	34.62	31.33	25.68	32.81	32.26	–	33.45
	CodeBERT	82.58	81.57	69.29	80.96	79.94	–	28.45	50.13	46.81	16.92	49.75	48.12	–	22.19
	PLBART	83.87	81.66	71.17	78	82.94	–	57.39	79.4	72.77	61.26	74.16	44.26	–	56.23
	CodeT5	**86.33**	**85.12**	**73.22**	**84.56**	**83.56**	–	**79.3**	**85.55**	**82.09**	**72.26**	**83.79**	**81.72**	–	**65.86**
C	Naive Copy	83.93	65.46	38.49	63.05	55.55	41.85	–	78.4	59.41	20.2	59.83	53.54	19.75	–
	CodeBERT	45.84	39.69	13.55	39.71	29.85	38.88	–	21.7	21.27	21.1	19.5	15.64	31.71	–
	PLBART	82.53	72.35	49.16	75.78	75.05	60.86	–	78.42	13.45	5.53	45.15	31.47	25.17	–
	CodeT5	**90.26**	**81.81**	**63.81**	**83.05**	**79.73**	**66.32**	–	**88.17**	**76.12**	**56.32**	**80.2**	**76.5**	**64.28**	–
CodeBLEU	**Model**	**C++**	**Java**	**Py**	**C#**	**JS**	**PHP**	**C**	**C++**	**Java**	**Py**	**C#**	**JS**	**PHP**	**C**
Code Synthesis	CodeBERT	22.7	25.53	12.26	23.44	23.87	36.47	10.63	26.51	31.14	24.5	33.37	29.09	39.84	18.08
	PLBART	34.89	32.23	4.62	29.36	29.63	**37.56**	**22.88**	44.09	41.55	33.77	40.7	38.33	43.01	6.72
	CodeT5	**35.48**	**33.51**	**21.1**	**30.64**	**29.99**	36.37	21.93	**45.18**	**42.73**	**35.02**	**43.6**	**38.66**	**45.02**	**34.88**
BLEU	**Model**	**C++**	**Java**	**Py**	**C#**	**JS**	**PHP**	**C**	**C++**	**Java**	**Py**	**C#**	**JS**	**PHP**	**C**
Code Summarization	CodeBERT	14.4	13.13	3.96	14.07	11.81	11.25	5.84	7.68	5.47	2.04	7.58	7.67	7.5	6.64
	PLBART	14.77	13.76	8	14.37	10.93	9.07	**7.5**	7.65	6.35	4.86	**9.23**	6.78	6.03	4.14
	CodeT5	**17.36**	**16.69**	**10.76**	**17.44**	**14.34**	**13.42**	6.63	**9.62**	**8.82**	**6.32**	7.75	**8.23**	**10.5**	**12.84**
MRR	**Model**	**C++**	**Java**	**Py**	**C#**	**JS**	**PHP**	**C**	**C++**	**Java**	**Py**	**C#**	**JS**	**PHP**	**C**
NL Code Search	RoBERTa	25.77	25.85	27.08	25.64	26.78	33.47	36.14	51.47	50.4	48.98	52.24	50.05	62.01	**56.34**
	CodeBERT	**29.77**	**29.41**	**30.94**	**29.08**	**31.2**	**38.75**	**41.56**	**59.13**	**56.07**	**57.97**	**56.65**	**54.37**	**65.13**	47.13
XL Code Search	RoBERTa	41.73	41.25	36.16	41.18	43.17	**41.17**	37.1	48.28	47.66	46.11	46.4	47.6	43.76	40.15
	CodeBERT	**42.11**	**41.71**	**36.98**	**41.52**	**43.41**	41.09	**37.87**	**48.71**	**48.33**	**47.24**	**47.96**	**47.66**	**44.02**	**40.43**

Effect of Transfer Learning from Snippet-level Training: From Table 4, first section, we noticed that models perform significantly better at snippet-level than program-level on most language pairs in the translation task. This is because 1) Snippets are much shorter than programs. As shown in Table 2, the average length of snippets is 1/7 of the programs. 2) Snippet data is much more than program data. As shown in Table 3, the amount of pairwise snippet data is 8 times of program data. Motivated by this, we employ transfer learning from snippet-level training to improve the Program

Table 5: Transfer learning from Snippet-Level training for Program Translation task on low resource language C. ST - Snippet Transfer.

Model	C-C++	C-Java	C-Py	C-C#	C-JS	C-PHP	C++-C	Java-C	Py-C	C#-C	JS-C	PHP-C
CodeBERT	21.67	21.27	21.1	19.48	15.68	31.71	21.87	21.27	18.32	21.57	19.79	22.19
CodeBERT+ST	**38.85**	**37.55**	**19.79**	**33.52**	**27.1**	**37.61**	**31.99**	**30.52**	**24.07**	**34.16**	**29.67**	**28.35**
PLBART	78.42	13.43	5.53	45.14	31.42	25.17	72.61	63.4	29.01	57.6	67.71	56.15
PLBART+ST	**81.1**	**70.78**	**44.26**	**72.68**	**73.27**	**60.71**	**79.72**	**77.3**	**47.48**	**74.09**	**72.6**	**64.64**
CodeT5	88.17	76.15	56.3	80.2	76.42	64.28	85.67	78.76	56.44	77.38	74.56	65.8
CodeT5+ST	**89.06**	**79.04**	**62.61**	**80.53**	**78.59**	**68.31**	**88.96**	**82.08**	**60.97**	**80.93**	**79.58**	**77.58**

Table 6: Top compilation errors in each target language (Javascript not included).

Language	Top-3 Compilation Errors in Each Target Language		
C++	expected '}' at end of input	stray '#' in program	'define' does not name a type
Java	';' expected	not a statement	unclosed character literal
Python	SyntaxError: invalid syntax	SyntaxError: unexpected EOF while parsing	IndentationError: expected an indented block
C#	Too many characters in character literal	Unexpected symbol 'end-of-file'	Newline in constant
PHP	Syntax error, unexpected '}',expecting EOF..	Syntax error, unexpected ')'..	Syntax error, unexpected EOF on line 1
C	expected declaration or statement at end of input	expected '=',',',';','asm' ... before ')' token	expected statement before ')' token

Translation performance on low-resource language C. Table 5 shows the performance of each model with and without the transfer learning. For example, CodeBERT is trained only on program data; "CodeBERT + ST" (ST is short for Snippet Transfer) model is first trained on the snippet data, and then on program data. All the models' performances improve by a wide margin on all the language pairs after snippet-level transfer learning, both when C is the source or target language.

Top Compilation Errors in Generated Programs: Table 6 shows the top compilation error types from compiling the generated programs from the Program Translation task. We aggregated the results of generated programs from all the baselines by the target language, because 1) the top error types of each baseline are very similar and 2) the space is limited. From this table, we can see that the top error types are mostly syntactic errors, such as bracket mismatch (C++, PHP, C), indentation mismatch (Python), missing ';' (Java). This indicates that the models need improvement in capturing the structure of the programs.

4.3 Limitations and Future Work

From our analysis of the results, we can conclude that Sequence-to-Sequence pretraining tasks, multilingual pretraining data, and Snippet-level Transfer Learning can potentially improve the performance on multiple tasks and low resource languages. This is an important insight for the design and development of future models in this domain. A good code generation model should also be able to learn and preserve the structure of the code since the current models mostly make syntactic errors in generation. For the evaluation of code generation tasks, we use CodeBLEU as metric, which evaluates the code syntax and semantics along with n-gram matching (as in BLEU). However, the evaluation can be further improved by using test cases. Automated test case generation can be explored in future work. The tasks we introduce aim to rigorously evaluate code models with the parallel data from the dataset. Therefore, not all the tasks have practical applications in real-world, especially the snippet-level tasks. One future direction is to make use of the comments and snippets to iteratively generate programs.

5 Conclusion

In this paper, we introduce a new dataset which is parallel across 8 languages (7 programming languages and 1 natural language) at both snippet level and program level. To the best of our knoweldge, it is the largest parallel dataset for source code in terms of both size and number of languages. We also introduce 10 different cross-lingual tasks to facilitate the development and evaluation of models in this domain. Moreover, we run experiments for all the 10 tasks on the proposed dataset with a number of state-of-the-art baseline models and provided insights about model design for the new challenges. We believe that this dataset will be of significant value to the research community and can potentially benefit a number of code-related research problems.

REFERENCES

Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. Unified pre-training for program understanding and generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2655–2668, 2021a.

Wasi Uddin Ahmad, Md Golam Rahman Tushar, Saikat Chakraborty, and Kai-Wei Chang. Avatar: A parallel corpus for java-python program translation. *arXiv preprint arXiv:2108.11590*, 2021b.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*, 2019.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1536–1547, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.139. URL https://aclanthology.org/2020.findings-emnlp.139.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pp. 4171–4186, 2019.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 7871–7880, 2020.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. 2019.

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742, 2020.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. Codexglue: A machine learning benchmark dataset for code understanding and generation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pp. 311–318, 2002.

Ruchir Puri, David S Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladmir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, et al. Project codenet: A large-scale ai for code dataset for learning a diversity of coding tasks. *arXiv preprint arXiv:2105.12655*, 2021.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*, 2020.

Baptiste Roziere, Marie-Anne Lachaux, Lowik Chanussot, and Guillaume Lample. Unsupervised translation of programming languages. In *NeurIPS*, 2020.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 8696–8708, 2021.

Ming Zhu, Karthik Suresh, and Chandan K Reddy. Multilingual code snippets training for program translation. In *36th AAAI Conference on Artificial Intelligence (AAAI)*, 2022.

Daniel Z¨ugner, Tobias Kirschstein, Michele Catasta, Jure Leskovec, and Stephan G¨unnemann. Language-agnostic representation learning of source code from structure and context. In *International Conference on Learning Representations (ICLR)*, 2021.

A APPENDIX

A.1 RELATED WORK

Parallel Code Data CodeXGLUE (Lu et al., 2021) is a popular benchmark that includes 14 datasets for 10 code related tasks. The tasks include clone detection, code translation, natural language code search, etc. However, this benchmark does not contain datasets with parallel codes from more than 2 languages. CoST Zhu et al. (2022) is a code translation dataset for 7 programming languages. However, it is relatively small and only supports translation task. AVATAR (Ahmad et al., 2021b) presents another parallel dataset for Java-Python translation. The authors collect multiple solutions for problems scraped from competitive programming websites and then form n^2 possible combinations of parallel data. This is also constrained to only 2 languages. Project CodeNet (Puri et al., 2021) has an abundance of parallel programs in a wide range of languages. However, the programs are significantly different in logic and structure, thus the alignment is of low quality.

Cross-Lingual Code Tasks Several tasks in the code domain are related to our work, including Code Translation, Code Summarization, Code Synthesis, and Code Search. CodeBERT (Feng et al., 2020) pre-trained a BERT (Kenton & Toutanova, 2019) based encoder on the source code, and then added a decoder to perform end-to-end training on code translation. CodeBERT is also used for Code Search tasks. PLBART (Ahmad et al., 2021a) utilized an existing natural language translation model, BART (Lewis et al., 2020), and also pre-trained it with source code. CodeTransformer (Z¨ugner et al., 2021) uses language agnostic features computed from the source code and its abstract syntax tree for code summarization. OpenAI's Codex (Chen et al., 2021) framework makes use of GPT (Radford et al.) language models fine-tuned on publicly available code from GitHub for code related downstream tasks. However, most of the models only explored a limited number of languages, due to the scarcity of multilingual parallel data.

A.2 BLEU SCORES FOR CODE GENERATION

Due to space constraints in the main document, we are including the BLEU results for Code Translation and Synthesis tasks in Table 7 in the appendix.

BLEU score has been the defacto evaluation metric used to evaluate natural language translation tasks. It measures the similarity between the generated translation and a set of reference texts. However, different from natural languages, programming languages have more rigorous syntax and semantics. A minor change in the code sequence, such as addition or removal of a bracket, may not affect the BLEU score by much, but it can potentially alter the structure and functionality of the code substantially. Therefore, in the main paper, we use CodeBLEU as the evaluation metric for code generation task, as it takes into consideration the Abstract Syntax Tree (AST) matching and Data Flow Graph (DFG) matching, which measure the syntax and semantics of the code, respectively.

Due to the fact that BLEU only measures the n-gram matching and ignores code syntax and semantics, it can be observed from our translation results (see Table 7) that BLEU scores clearly over estimate the model performance. Almost all BLEU score values can be found to be greater than CodeBLEU scores presented in the main paper. The effect is more pronounced for the program level tasks than for the snippet level tasks. This shows that maintaining long term structure in full programs, which are much longer than snippets, is harder.

However, from our Code Synthesis results (see Table 7), we observe that the BLEU scores are lower than CodeBLEU scores. This is because the Code Synthesis results are much lower (in absolute value) than translation results to begin with, and taking into account AST and DFG matching increases the overall scores.

A.3 DATASET STATISTICS

A detailed breakdown of the data statistics for the translation task is provided in this section due to space limitation in the main paper. Table 8 summarizes the number of aligned code pairs contained in the train, validation, and test sets for all possible language pair combinations both at the snippet and program level.

A.4 More Details about Data Collection

The data was scraped from different sub-pages of the GeeksForGeeks website. A majority of the problems on this site belong to the following two categories: Data Structures and Algorithms. These two categories have different sub-categories within them. For example, the Data Structures page has the hyperlinked sub-categories of Array, Linked Lists, Stack, Queue, etc. which when clicked direct the user to all the problems relating to that specific sub-category and their corresponding solutions in different programming languages. The same goes for the Algorithms page.

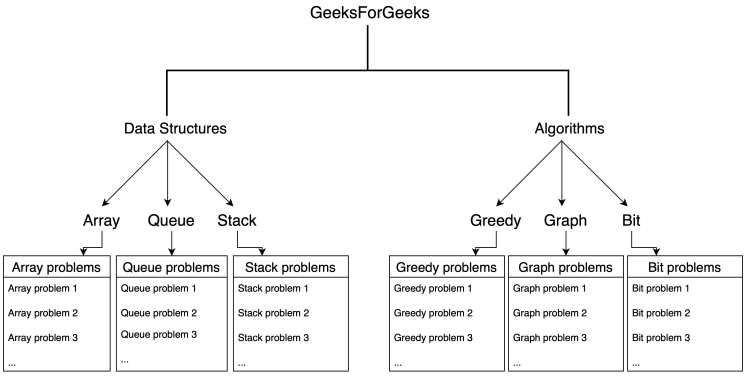

Figure 2: A breakdown of the problem sets included in the data and their organization on the GeeksForGeeks portal. Only three sub-categories per category is shown here for purposes of brevity.

For scraping the data, we used python scripts and some external libraries, the most important of them being BeautifulSoup4[5] and Selenium[6]. BeautifulSoup4 is a python library that facilitates the acquisition of data from HTML and XML files. Selenium is a python package that, in our case, is used to automate web browser functions to navigate through the GeeksForGeeks page directories. Each page in GFG that houses a problem and its solution has a uniform HTML page structure. This structure allows us to extract specifically targeted sections which are needed for our dataset using BeautifulSoup4.

Using the directory structure of the GeekForGeeks website, the ability of Selenium to navigate through these pages and the utility of BeautifulSoup4 for content extraction from these pages, we could extract the data from the website with practically no manual intervention. Every problem page can have one or more solutions to the same problem in different languages. For example, if there is a problem statement "Given a number n, print n-th Fibonacci Number", there can be different logics to solve the same problem. One solution may use recursion-based logic, another can utilize dynamic programming and yet another can use a space complexity optimized method to solve the same problem. Each logic has code in different languages and code pertaining to each logic resides in separate sections of the page which can be identified via their HTML tags. We extracted every possible problem statement and solution from the above-mentioned two categories and did not put any filter on what category, type, difficulty, etc. the solutions belong to at collection time.

A.5 Creating Data Splits

A natural way to generate train-validation-test sets from the data is to split at the problem level. However, the number of programs in different languages are imbalanced. Only 5.1% of the problems have C programs, and it is 31.5% for PHP programs. Random splitting at the problem level can exacerbate this problem, resulting in very small test/validation dataset for C and PHP. Moreover, only a small number of problems have programs in all the 7 languages. It is beneficial to use these problems for evaluation, as they can provide a fair comparison across all the languages. Satisfying these two constraints, we take the following steps to create the data split:

[5]https://www.crummy.com/software/BeautifulSoup/

[6]https://www.selenium.dev/

1. Out of all the problems that have programs (solutions) in all 7 languages, we randomly sample the test and validation sets for C. We start out with creating the splits for C in particular since it represents the smallest proportion of the dataset.
2. Next, we first remove all the problems that have C programs. Out of all the problems that have programs in the remaining 6 languages (excluding C), we randomly sample the partial test and validation sets for PHP, so that the combined problems from this step and the previous one can be used as final test and validation sets of the PHP programs.
3. Finally, we remove all the problems that have C programs or PHP programs. Since the remaining 5 languages have approximately same number of programs, we randomly sample the partial test and validation sets and use them for all the 5 languages. The final test and validation set for each of the 5 languages is the combination of these problems and problems from the previous two steps. This allows us to maintain a split ratio of approximately 85-5-10 (train-val-test) for all the 7 languages.

Our splitting strategy provides a balanced split across languages and ensures there is no overlap between any evaluation set(test or validation) and any training set across all languages.

A.6 More Details about XL Code Search

For this task, we create 7 different datasets, one for each language where the chosen language is the query language and all other languages form the candidate solutions. For example, let us consider the dataset for C++ which contains entries like "1057-C++-1/1057-C#-1". This basically represents the datapoint where the first snippet of problem ID 1057 in C++ is the query and the corresponding answer snippet is in C#. However, this is not the only correct pairing, the dataset contains all the possible correct pairings which include {1057-C#-1, 1057-C-1, 1057-Python-1, 1057-Javascript-1, 1057-PHP-1, 1057-Java-1}. When any of these solutions are present, the output candidate list is considered as a correctly chosen candidate. It should also be noted that all queries do not have exhaustively all other languages as candidate solutions.

A.7 More Details about Evaluation Metrics

- **BLEU:** Given an input code sample, we use BLEU (Papineni et al., 2002) score to evaluate the n-gram overlap between the generated and the ground-truth target text and code.
- **CodeBLEU:** CodeBLEU (Ren et al., 2020) is designed for automatic evaluation of code synthesis. Besides n-gram match (as in BLEU), it also evaluates the code syntax via abstract syntax trees (AST) and code semantics via data-flow. We use CodeBLEU for code generation tasks like Code Translation and Code Synthesis. The original CodeBLEU does not support C and C++. We extend the CodeBLEU code to include these two languages. The related code is included in the GitHub repo.
- **Mean Reciprocal Rank (MRR):** The reciprocal rank is defined as the inverse of the rank of the first correct candidate for a given query. MRR is the mean of the reciprocal rank for all the queries in the test set. In order to evaluate our XL Code Search task, we modified the traditional definition of the **MRR** metric to account for the possibility of multiple correct candidate solutions. We modify it in the following manner:

  Given a query q_i from the set of queries $Q = \{q_1, q_2 \dots q_m\}$, candidate set $C_i = \{c_{i1}, c_{i2}, \dots, c_{in}\}$ corresponding to q_i and the answer set $A_i = \{a_{i1}, a_{i2}, \dots, a_{ik}\}$ where $k \in [1, 6]$, r_{ij} is the reciprocal rank of the j^{th} candidate c_{ij} for $c_{ij} \in A_i$.

$$MRR_{q_i} = \frac{1}{k} \sum_{\substack{j=1 \\ c_{ij} \in A_i}}^{n} r_{ij}$$

$$MRR_Q = \frac{1}{m} \sum_{i=1}^{m} MRR_{q_i}$$

B DATASET INFORMATION

The dataset and the code used in this paper will be made publicly available upon acceptance.

B.1 MOTIVATION

As described in the main paper, the primary motivation behind the creation and release of this data is to potentially facilitate and foster research in the domain of Deep Learning for Software Engineering. Code related tasks have garnered a lot of attention by the community in the past few years but it has been our observation that the availability of high quality, parallel data across multiple languages which is required to be able to produce advances in this domain, is still limited. We discuss in the main paper as well, how most of the widely used datasets are either limited to just a few language pairs or are limited in size. With the release of this dataset we aim to fill both of those gaps, and to give the research community better tools in order to solve code-related tasks.

B.2 INTENDED USE

The primary intended uses of the dataset are to encourage development and validation of models/methods for code related tasks such as translation, summarization, synthesis, and search. The link to the dataset can be found in the README of the GitHub repository. Code required to reproduce results and baseline scores can also be found in the GitHub README file. Readers will need to cite the original dataset when using it in their experiments or making modifications to it.

B.3 AUTHOR STATEMENT

The IP policies and regulations for GeeksForGeeks were carefully followed and we confirm that no data privacy policy was violated when collecting the data. We bear all responsibility in case of violation of rights. We confirm that the dataset is distributed under CC BY-SA License 4.0.

B.4 MAINTENANCE

The dataset will be actively maintained by the authors. Issues can be reported via raising an issue on GitHub or e-mail to one of the authors. The dataset will be hosted on Google Drive since its large size is not supported by GitHub. Any changes to hosting will be reflected in the links on the GitHub repository. The authors may also update the dataset by adding more datapoints, or in case issues are reported by other parties or are found by the authors themselves. Any such updates to the data will be documented on GitHub.

B.5 SOCIETAL IMPACT

Since deep learning models have become larger, the amount of computational power needed to train and maintain them has also increased. An unintended consequence of this has been the increased carbon footprint of deep learning research as a result of running large number of experiments to validate hypotheses. As our dataset aims to facilitate further research in the domain, it would also end up having this societal impact, albeit indirectly so. We would encourage users to use compute and memory efficient methods when carrying their research using this dataset.

Table 7: BLEU scores for two tasks (Translation and Code Synthesis) separated by a double horizontal rule. Above the double rule are presented BLEU scores of the Translation task for the 42 programming language pairs in the $XLCoST$ dataset. Below the double rule are the BLEU scores for the Code Synthesis task. Column headers represent target languages. For translation, row headers represent source languages.

		Snippet-level							**Program-level**						
BLEU	**Model**	**C++**	**Java**	**Py**	**C#**	**JS**	**PHP**	**C**	**C++**	**Java**	**Py**	**C#**	**JS**	**PHP**	**C**
C++	Naive Copy	–	64.57	37.29	65.89	59.73	37.44	84.44	–	64.47	34.48	65.98	58.09	38.13	84
	CodeBERT	–	85.03	79.72	85.64	84.61	87.18	44.48	–	80.09	15.43	81.24	78.14	50.68	11.7
	PLBART	–	84.02	80.12	84.86	85.34	87.31	85.26	–	81.23	77.5	83.96	69.6	83.94	77.94
	CodeT5	–	**86.45**	**81.73**	**86.55**	**86.24**	**89.84**	**91.82**	–	**84.29**	**78.69**	**85.69**	**83.75**	**90.88**	**93.64**
Java	Naive Copy	64.48	–	34.04	76.95	60.42	35.02	65.67	64.88	–	31.28	78.09	58.23	34.35	65.48
	CodeBERT	88.18	–	61.45	92.64	86.42	84.57	38.47	83.24	–	2.51	88.31	81.16	52.7	12.27
	PLBART	87.69	–	55.05	91.63	87.17	84.92	71.43	85.83	–	73.16	89	84.62	84.18	66.1
	CodeT5	**89.2**	–	**79.37**	**92.9**	**87.95**	**88.12**	**84.29**	**87.67**	–	**76.92**	**90.5**	**85.38**	**88.86**	**85.64**
Python	Naive Copy	37.36	34	–	35.06	42.64	22.05	38.5	34.43	30.81	–	31.93	39.57	21.22	35.68
	CodeBERT	82.02	57.17	–	53.94	57.64	80.32	9.01	72.56	25.24	–	8.39	17.97	48.46	4.5
	PLBART	81.17	69.17	–	69.29	68	82.27	57.8	78.4	73.1	–	71.14	73.73	79.56	27.07
	CodeT5	**83.07**	**76.61**	–	**78.59**	**78.16**	**85.14**	**70.64**	**81.8**	**77.9**	–	**78.21**	**76.25**	**85**	**51.89**
C#	Naive Copy	65.71	77.11	34.99	–	61.13	35.25	66.63	66.13	78.04	32.2	–	59.09	36.08	66.39
	CodeBERT	87.76	89.99	59.63	–	86.13	84.41	39.19	82.48	86.86	4.68	–	80.8	53.39	11.48
	PLBART	87.03	4.97	70.85	–	86.88	84.66	76.83	84.77	87	73.94	–	84.55	84.2	60.27
	CodeT5	**88.81**	**91.67**	**77.64**	–	**87.75**	**88.01**	**83.54**	**87.18**	**89.28**	**76.81**	–	**85**	**89.25**	**83.72**
JS	Naive Copy	59.84	60.36	42.63	61.23	–	33.05	56.07	57.99	57.26	39.58	58.54	–	34	53.8
	CodeBERT	84.97	84.1	54.19	84.69	–	83.37	31.68	79.54	77.97	3.36	80.73	–	54.02	11.04
	PLBART	84.75	84.44	70.99	84.85	–	84.19	73.57	84.24	82.46	70.68	83.86	–	84.4	69.46
	CodeT5	**85.74**	**85.22**	**77.46**	**85.96**	–	**86.91**	**82.03**	**85.72**	**84.14**	**75.46**	**85.44**	–	**87.5**	**77.92**
PHP	Naive Copy	37.39	35.03	22.08	35.2	33.17	–	36.62	38.15	34.32	21.36	36.08	34.44	–	37.53
	CodeBERT	84.56	82.35	74.68	81.93	82.82	–	27.94	52.19	50.94	10.87	54.61	53.13	–	8.42
	PLBART	84.81	82.13	76.42	78.86	85.43	–	53.62	84.56	78.61	68.9	80.26	44.71	–	56.16
	CodeT5	**88.59**	**85.88**	**79.47**	**85.67**	**86.46**	–	**83.05**	**89.96**	**86.95**	**80.29**	**87.77**	**87.01**	–	**68.74**
C	Naive Copy	84.34	65.65	38.47	66.64	56.19	36.67	–	83.99	65.29	35.94	66.4	54.52	37.53	–
	CodeBERT	45.55	38.79	9.83	39.09	26.85	27.03	–	15.51	17.77	6.01	14.92	13.06	8.53	–
	PLBART	83.01	72.21	44.76	76.26	78.8	72.37	–	84.88	10.65	4.02	38.53	18.6	0.2	–
	CodeT5	**91.76**	**82.12**	**65.89**	**84.06**	**82.16**	**82.82**	–	**93.15**	**83.08**	**54.6**	**85.39**	**82.42**	**78.7**	–
BLEU	**Model**	**C++**	**Java**	**Py**	**C#**	**JS**	**PHP**	**C**	**C++**	**Java**	**Py**	**C#**	**JS**	**PHP**	**C**
Code Synthesis	CodeBERT	17.19	18.78	7.82	18.58	19.53	**22.54**	5.16	21.39	27.16	19.58	30.83	25.53	29.6	8.85
	PLBART	24.01	28.12	1.31	26.61	17.27	20.16	**12.9**	39.94	41.01	29.92	38.92	35.95	35.91	3.53
	CodeT5	**28.52**	**29.65**	**12.87**	**28.16**	**21.54**	18.7	12.47	**41.53**	**42.37**	**31.9**	**42.14**	**36.31**	**39.97**	**28.64**

Table 8: Number of pairwise code-code data in training, validation, and testing splits for each language-pair. The upper triangle (in bold font) shows the number of parallel code snippets, and the lower triangle shows the number of parallel programs. This data is used for the Code Translation and XL Code Search tasks. (**Py** is short for Python. **JS** is short for Javascript.)

Lang		C++	Java	Py	C#	JS	PHP	C
C++	train	**–**	**89040**	**80100**	**85662**	**69507**	**17811**	**3386**
	val	**–**	**4419**	**3913**	**4408**	**3808**	**923**	**352**
	test	**–**	**8059**	**7228**	**7922**	**6965**	**1647**	**222**
Java	train	9450	–	**77759**	**87065**	**69341**	**17853**	**2996**
	val	490	–	**3938**	**4437**	**3826**	**929**	**353**
	test	901	–	**7259**	**8011**	**7005**	**1672**	**238**
Py	train	9139	8991	–	**75843**	**67219**	**17616**	**2478**
	val	468	471	–	**3922**	**3750**	**923**	**311**
	test	878	882	–	**7215**	**6861**	**1655**	**203**
C#	train	9187	9301	8826	–	**68093**	**17873**	**2958**
	val	488	491	470	–	**3826**	**928**	**352**
	test	890	898	877	–	**6961**	**1668**	**238**
JS	train	8482	8470	8182	8367	–	**17117**	**1875**
	val	472	475	459	475	–	**921**	**309**
	test	878	881	864	877	–	**1617**	**200**
PHP	train	3056	3068	3003	3071	2971	–	**856**
	val	157	158	153	158	157	–	**271**
	test	303	307	304	307	302	–	**183**
C	train	402	409	380	394	308	170	–
	val	59	59	59	59	59	55	–
	test	45	49	48	49	49	43	–

Table 9: Example of the parallel alignment of the code data in four languages. The programs given here checks if a given number is divisible by 3 or not.

C++	Java	Python	PHP

`/* C++ program to find if a number is divisible by 3 or not */`
 	
`#include<bits/stdc++.h>`
`using namespace std;`
	`/* Java program to find if a number is divisible by 3 or not */`
	
	`class IsDivisible`
	`{`
		`''' Python program to find if a number is divisible by 3 or not '''`	`/* PHP program to find if a number is divisible by 3 or not */`
			
			`<?php`

`/* Function to find that number divisible by 3 or not */`
 	
`int check(string str)`
`{`
	`/* Function to find that number divisible by 3 or not */`
	
	`static boolean check(String str)`
	`{`
		`''' Function to find that number divisible by 3 or not '''`
		
		`def check(num) :`
			`/* Function to find that number divisible by 3 or not */`
			
			`function check($str)`
			`{`

`/* Compute sum of digits */`
 	
`int n = str.length();`
`int digitSum = 0;`
`for (int i=0; i<n; i++)`
`digitSum += (str[i]-'0');`
	`/* Compute sum of digits */`
	
	`int n = str.length();`
	`int digitSum = 0;`
	`for (int i=0; i<n; i++)`
	`digitSum += (str.charAt(i)-'0');`
		`''' Compute sum of digits '''`
		
		`digitSum = 0`
		`while num > 0 :`
		`rem = num % 10`
		`digitSum = digitSum + rem`
		`num = num / 10`
			`/* Compute sum of digits */`
			
			`$n = strlen($str);`
			`$digitSum = 0;`
			`for ($i = 0; $i < $n; $i++)`
			`$digitSum += ($str[$i] - '0');`

`/* Check if sum of digits is divisible by 3 */`
 	
`return (digitSum % 3 == 0);`
`}`
	`/* Check if sum of digits is divisible by 3 */`
	
	`return (digitSum % 3 == 0);`
	`}`
		`''' Check if sum of digits is divisible by 3 '''`
		
		`return (digitSum % 3 == 0)`
			`/* Check if sum of digits is divisible by 3 */`
			
			`return ($digitSum % 3 == 0);`
			`}`

`/* Driver code */`
 	
`int main()`
`{`
`string str = ""1332"";`
`check(str)? cout << ""Yes"" : cout << ""No "";`
`return 0;`
`}`
	`/* main function */`
	
	`public static void main (String[] args)`
	`{`
	`String str = ""1332"";`
	`if(check(str))`
	`System.out.println(""Yes"");`
	`else`
	`System.out.println(""No"");`
	`}`
	`}`
		`''' main function '''`
		
		`num = 1332`
		`if(check(num)) :`
		`print ""Yes""`
		`else :`
		`print ""No""`
			`/* Driver code */`
			
			`$str = "1332";`
			`$x = check($str) ? "Yes" : "No ";`
			`echo($x);`
			`?>`

Table 10: Example of the parallel alignment of the code data in four languages. The programs given here aim to find the LCM of two given numbers

C#	JavaScript	PHP	C

`/* C# program to find LCM of two numbers */`
 	
`using System;`
`class GFG {`
	`/* Javascript program to find LCM of two numbers */`	`/* PHP program to find LCM of two numbers */`
		
		`<?php`
			`/*C program to find LCM of two numbers*/`
			
			`#include <stdio.h>`

`/* Recursive method to return gcd of a and b */`
 	
`static int gcd(int a, int b)`
`{`
`  if (a == 0)`
`  return b;`
`  return gcd(b % a, a);`
`}`
	`/* Recursive function to return gcd of a and b */`
	
	`function gcd(a, b)`
	`{`
	`if (b == 0)`
	`  return a;`
	`return gcd(b, a % b);`
	`}`
		`/* Recursive function to return gcd of a and b */`
		
		`function gcd( $a, $b )`
		`{`
		`  if ($a == 0)`
		`  return $b;`
		`  return gcd($b % $a, $a);`
		`}`
			`/* Recursive function to return gcd of a and b */`
			
			`int gcd(int a, int b)`
			`{`
			`  if (a == 0)`
			`  return b;`
			`  return gcd(b % a, a);`
			`}`

`/* method to return LCM of two numbers */`
 	
`static int lcm(int a, int b)`
`{`
`  return (a / gcd(a, b)) * b;`
`}`
	`/* Function to return LCM of two numbers */`
	
	`function lcm(a, b)`
	`{`
	`  return (a / gcd(a, b)) * b;`
	`}`
		`/* Function to return LCM of two numbers */`
		
		`function lcm( $a, $b )`
		`{`
		`  return ($a / gcd($a, $b)) * $b;`
		`}`
			`/* Function to return LCM of two numbers */`
			
			`int lcm(int a, int b)`
			`{`
			`  return (a / gcd(a, b)) * b;`
			`}`

`/* Driver method */`
 	
`public static void Main()`
`{`
`int a = 15, b = 20;`
`Console.WriteLine("LCM of " + a + " and " + b + " is " + lcm(a, b));`
`}`
`}`
	`/* Driver program to test above function */`
	
	`let a = 15, b = 20;`
	`document.write("LCM of " + a + " and " + b + " is " + lcm(a, b));`
		`/* Driver Code */`
		
		`$a = 15;`
		`$b = 20;`
		`echo "LCM of ",$a, " and " ,$b, " is ", lcm($a, $b);`
		`?>`
			`/* Driver program to test above function */`
			
			`int main()`
			`{`
			`int a = 15, b = 20;`
			`printf("LCM of %d and %d is %d ", a, b, lcm(a, b));`
			`return 0;`
			`}`

Table 11: Example of the parallel alignment of the code data in all seven languages. The Programs given here aim to find two elements whose sum is closest to zero.

C++

```
/* C++ code to find
     Two elements
     whose sum is
     closest to zero
     */

# include <bits/stdc
    ++.h>
# include <stdlib.h>
# include <math.h>
using namespace std;
void minAbsSumPair(
     int arr[], int
     arr_size)
{
int inv_count = 0;
int l, r, min_sum,
     sum, min_l,
     min_r;
```

```
/* Array should have
     at least two
     elements */

if (arr_size < 2)
{
   Console.Write("
        Invalid Input"
        );
   return;
}
```

```
/* Initialization of
     values */

min_l = 0;
min_r = 1;
min_sum = arr[0] +
     arr[1];
for(l = 0; l <
     arr_size - 1; l
     ++)
{
   for(r = l + 1; r <
        arr_size; r
        ++)
   {
   sum = arr[l] + arr
        [r];
   if(abs(min_sum) >
        abs(sum))
   {
      min_sum = sum;
      min_l = l;
      min_r = r;
   }}}}
```

```
/* Driver Code */

int main()
{
   int arr[] = {1, 60,
        -10, 70, -80,
        85};
   minAbsSumPair(arr,
        6);
   return 0;
}
```

Java

```
/* Java code to find
     Two elements
     whose sum is
     closest to zero
     */

import java.util.*;
import java.lang.*;
class Main
{
static void
     minAbsSumPair(
     int arr[], int
     arr_size)
{
int inv_count = 0;
int l, r, min_sum,
     sum, min_l,
     min_r;
```

```
/* Array should have
     at least two
     elements */

if(arr_size < 2)
{
   document.write("
        Invalid
        Input");
   return;
}
```

```
/* Initialization of
     values */

min_l = 0;
min_r = 1;
min_sum = arr[0] +
     arr[1];
for(l = 0; l <
     arr_size - 1; l
     ++)
{
for(r = l+1; r <
     arr_size; r++)
{
 sum = arr[l] + arr
      [r];
 if(Math.abs(
      min_sum) >
      Math.abs(sum))
 {
  min_sum = sum;
  min_l = l;
  min_r = r;
 }}}}
```

```
/* main function */

public static void
     main (String[]
     args)
{
   int arr[] = {1,
        60, -10, 70,
        -80, 85};
   minAbsSumPair(arr
        , 6);
}
}
```

Python

```
''' Python3 code to
     find Two
     elements whose
     sum is
     closest to
     zero '''

def minAbsSumPair(
     arr,arr_size):
   inv_count = 0
```

```
''' Array should
     have at least
     two elements
     '''

   if arr_size < 2:
      print("Invalid
           Input")
      return
```

```
''' Initialization
     of values '''

   min_l = 0
   min_r = 1
   min_sum = arr[0]
        + arr[1]
   for l in range
        (0, arr_size
        - 1):
      for r in range
           (l + 1,
           arr_size):

         sum = arr[l
              ] +
              arr[r]
         if abs(
              min_sum
              ) >
              abs(
              sum):
            min_sum
                 =
                 sum
            min_l =
                 l
            min_r =
                 r
```

```
''' Driver program
     to test above
     function '''

arr = [1, 60, -10,
     70, -80, 85]
minAbsSumPair(arr,
     6);
```

C#

```
/* C# code to find
     Two elements
     whose sum is
     closest to zero
     */

using System;
class GFG
{
static void
     minAbsSumPair(
     int []arr, int
     arr_size)
{
int l, r, min_sum,
     sum, min_l,
     min_r;
```

```
/* Array should have
     at least two
     elements */

if (arr_size < 2)
{
   Console.Write("
        Invalid
        Input");
   return;
}
```

```
/* Initialization of
     values */

min_l = 0;
min_r = 1;
min_sum = arr[0] +
     arr[1];
for (l = 0; l <
     arr_size - 1; l
     ++)
{
   for (r = l+1; r <
        arr_size; r
        ++)
   {
      sum = arr[l] +
           arr[r];
      if (Math.Abs(
           min_sum)
           > Math.
           Abs(sum))
      {
         min_sum =
              sum;
         min_l = l;
         min_r = r;
      }}}}
```

```
/* main function */

public static void
     Main ()
{
   int []arr = {1,
        60, -10, 70,
        -80, 85};
   minAbsSumPair(arr
        , 6);
}
}
```

JavaScript	PHP	C	

/* JavaScript code to find Two elements whose sum is closest to zero */
 	
function minAbsSumPair(arr, arr_size)
{
var inv_count = 0;
var l, r, min_sum, sum, min_l, min_r;
	/* PHP program to find the Two elements whose sum is closest to zero */
	
	function minAbsSumPair($arr, $arr_size)
	{
	$inv_count = 0;
		/* C code to find Two elements whose sum is closest to zero */
		
		# include <stdio.h>
		# include <stdlib.h>
		# include <math.h>
		void minAbsSumPair(int arr[], int arr_size)
		{
		int inv_count = 0;
		int l, r, min_sum, sum, min_l, min_r;	

/* Array should have at least two elements */
 	
if(arr_size < 2)
{
document.write(" Invalid Input");
return;
}
	/* Array should have at least two elements */
	
	if($arr_size < 2)
	{
	echo "Invalid Input";
	return;
	}
		/* Array should have at least two elements */
		
		if(arr_size < 2)
		{
		printf("Invalid Input");
		return;
		}	

/* Initialization of values */
 	
min_l = 0;
min_r = 1;
min_sum = arr[0] + arr[1];
for(l = 0; l < arr_size - 1; l ++)
{
for(r = l + 1; r < arr_size; r ++)
{
sum = arr[l] + arr[r];
if(Math.abs(min_sum) > Math.abs(sum))
{
min_sum = sum;
min_l = l;
min_r = r;
}}}}
	/* Initialization of values */
	
	$min_l = 0;
	$min_r = 1;
	$min_sum = $arr[0] + $arr[1];
	for($l = 0; $l < $arr_size - 1; $l++)
	{
	for($r = $l+1; $r < $arr_size ; $r++)
	{
	$sum = $arr[$l] + $arr[$r];
	if(abs($min_sum) > abs($sum))
	{
	$min_sum = $sum;
	$min_l = $l;
	$min_r = $r;
	}}}}
		/* Initialization of values */
		
		min_l = 0;
		min_r = 1;
		min_sum = arr[0] + arr[1];
		for(l = 0; l < arr_size - 1; l ++)
		{
		for(r = l+1; r < arr_size; r++)
		{
		sum = arr[l] + arr [r];
		if(abs(min_sum) > abs(sum))
		{
		min_sum = sum;
		min_l = l;
		min_r = r;
		}}}}	

/* Driver Code */
 	
arr = new Array(1, 60, -10, 70, -80, 85);
minAbsSumPair(arr, 6) ;
	/* Driver Code */
	
	$arr = array(1, 60, -10, 70, -80, 85);
	minAbsSumPair($arr, 6);
	
	?>
		/* Driver program to test above function */
		
		int main()
		{
		int arr[] = {1, 60, -10, 70, -80, 85};
		minAbsSumPair(arr, 6);
		getchar();
		return 0;
		}	