CoTexT: Multi-task Learning with Code-Text Transformer

Long Phan[1], Hieu Tran[2], Daniel Le[1], Hieu Nguyen[1], James Anibal[1], Alec Peltekian[1], and Yanfang Ye[1]

[1]Case Western Reserve University, Ohio, USA
[2]University of Science, VNU-HCM, Vietnam
{*lnp26, yxy1032*}@*case.edu*

Abstract

We present CoTexT, a pre-trained, transformer-based encoder-decoder model that learns the representative context between natural language (NL) and programming language (PL). Using self-supervision, CoTexT is pre-trained on large programming language corpora to learn a general understanding of language and code. CoTexT supports downstream NL-PL tasks such as code summarizing/documentation, code generation, defect detection, and code debugging. We train CoTexT on different combinations of available PL corpus including both ”bimodal” and ”unimodal” data. Here, bimodal data is the combination of text and corresponding code snippets, whereas unimodal data is merely code snippets. We first evaluate CoTexT with multi-task learning: we perform Code Summarization on 6 different programming languages and Code Refinement on both small and medium size featured in the CodeXGLUE dataset. We further conduct extensive experiments to investigate CoTexT on other tasks within the CodeXGlue dataset, including Code Generation and Defect Detection. We consistently achieve SOTA results in these tasks, demonstrating the versatility of our models.

1 Introduction

In recent years, pre-trained language models (LM) have played a crucial role in the development of many natural language processing (NLP) systems. Before the emergence of large LMs, traditional word embedding gives each word/token a global representation. Large pre-trained models such as ELMo (Peters et al., 2018), GPT (Brown et al., 2020), BERT (Devlin et al., 2018), and XLNet (Yang et al., 2020) can derive contextualized word vector representations from large corpora. These methods can learn generalized representations of language and have significantly improved a broad range of downstream NLP tasks. These LMs make use of learning objectives such as Masked Language Modeling (MLM) (Devlin et al., 2018) where random tokens in a sequence are masked and the model predicts the original tokens to learn the context. The success of pre-trained models in NLP has created a path for domain-specific pre-trained LMs, such as BioBERT (Lee et al., 2019a) on biomedical text, or TaBERT (Yin et al., 2020) on NL text and tabular data.

We introduce CoTexT (Code and Text Transfer Transformer), a pre-trained model for both natural language (NL) and programming language (PL) such as Java, Python, Javascript, PHP, etc. CoTexT follows the encoder-decoder architecture proposed by (Vaswani et al., 2017) with attention mechanisms. We then adapt the model to match T5 framework proposed by (Raffel et al., 2019). We test CoTexT by performing exhaustive experiments on multi-task learning of multiple programming languages and other related tasks.

We train CoTexT using large programming language corpora containing multiple programming languages (including Java, Python, JavaScript, Ruby, etc.). Here, we test different combinations of unimodal and bimodal data to produce the best result for each downstream task. We then fine-tune CoTexT on four CodeXGLUE tasks (Lu et al., 2021) including CodeSummarization, CodeGeneration, Defect Detection and Code Refinement (small and medium dataset). Results show that we achieve state-of-the-art values for each of the four tasks. We found that CoTexT outperforms current SOTA models such as CodeBERT (Feng et al., 2020) and PLBART (Ahmad et al., 2021a).

In this paper we offer the following contribution:

- Three different versions of CoTexT that achieve state-of-the-art on the CodeXGLUE's CodeSummarization, CodeGeneration, Defect

Detection and Code Refinement (small and medium dataset) tasks. We publicize our CoTexT pre-trained checkpoints and related source code available for future studies and improvements.

2 Related Work

Recent work on domain adaptation of BERT show improvements compared to the general BERT model. BioBERT (Lee et al., 2019b) is further trained from $\text{BERT}_{\text{BASE}}$ on biomedical articles such as PubMed abstracts and PMC articles. Similarly, SciBERT (Beltagy et al., 2019) is trained on the full text of biomedical and computer science papers. The experimental results of these models on domain-specific datasets show the enhanced performance compared to $\text{BERT}_{\text{BASE}}$.

Relating specfically to our work, CodeBERT is (Feng et al., 2020) trained on bimodal data of NL-PL pairs. This strategy allows CodeBERT to learn general-purpose representations of both natural language and programming language. GraphCodeBERT (Guo et al., 2021) is an extension of CodeBERT that moves beyond syntactic-level structure and uses data flow in the pre-training stage to capture the semantic-level structure of code. More recently, PLBART (Ahmad et al., 2021b) is a pre-trained sequence-to-sequence model for NL and PL. Through denoising autoencoding, this model can perform well on NL-PL understanding and generation tasks.

3 CoTexT

3.1 Vocabulary

Following the example of T5 (Raffel et al., 2019), we use the Sentence Piece Unsupervised Text Tokenizer proposed by (Kudo and Richardson, 2018). The Sentence Piece model extracts the sub-words that contain the semantic context of a sequence. We employ Sentence Piece as a vocabulary model for all of our contributed CoTexT models. However, the special tokens used in code (such as ”[”, ”{”, ”$”, etc) are out-of-vocab for the SentencePiece model [1]. These tokens have a crucial representative context in programming languages. Therefore, to enhance the robustness of the model, we encode all of these missing tokens into a natural language representation during both self-supervised and supervised training.

[1]https://github.com/google/sentencepiece

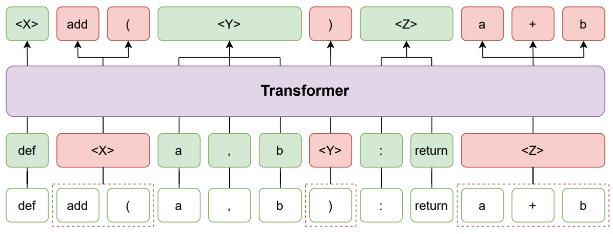

Figure 1: An illustration about Fill-in-the-blank objective

3.2 Pre-training CoTexT

We train CoTexT on both bimodal and unimodal data. Bimodal data contains both code snippets and the corresponding natural text in each sequence, while unimodal data contains only the sequence of code. We use two main datasets during self-supervised training: **CodeSearchNet Corpus Collection** (Husain et al., 2020) and **GitHub Repositories**[2] data. The combinations of corpus used to train CoTexT are listed in Table 1. To save both time and computing resources, we initialized the checkpoints from the original T5 that was trained on the C4 corpus. (Raffel et al., 2019).

3.2.1 CodeSearchNet Corpus Collection

CodeSearchNet Corpus (Husain et al., 2020) contains coded functions from open-source non-forked Github repositories. This dataset spans 6 coding languages (Python, Java, Javascript, PHP, Ruby, Go), which facilitates multi-task learning. CodeSearchNet also contains a natural language description for each function. For bimodal data, we simply concatenate the natural language snippet with the corresponding code snippet to create one input sequence. These data are then processed as described in 3.1.

3.2.2 GitHub repositories

We download a large collection of Java and Python functions from the GitHub repositories dataset available on Google BigQuery. These Java and Python functions are then extracted and the natural language descriptions are obtained using the preprocessing pipeline from (Lachaux et al., 2020). These datapoints also run through a pipeline to replace special tokens (as described in 3.1).

3.3 Input/Output Representations

CoTexT converts all NLP problems into a text-to-text format. This means that during both self-

[2]https://console.cloud.google.com/marketplace/details/github/github-repos

Table 1: Pre-training CoTexT on different combinations of natural language and programming language copora

Model	N-modal	Corpus combination
T5	NL	C4
CoTexT (1-CC)	PL	**C**4 + **C**odeSearchNet
CoTexT (2-CC)	NL-PL	**C**4 + **C**odeSearchNet
CoTexT (1-CCG)	PL	**C**4 + **C**odeSearchNet + **G**ithub Repos

supervised pre-training and supervised training, we use an input sequence and a target sequence. For the bimodal model, we concatenate a sequence of natural language text and the corresponding sequence of programming language text as an input. For the unimodal model, we simply use each coded function as an input sequence. During self-supervised training, spans of the input sequence are randomly masked and the target sequence (Raffel et al., 2019) is formed as the concatenation of the same sentinel tokens and the real masked spans/tokens.

3.4 Model Architecture

CoTexT follows the sequence-to-sequence encoder-decoder architecture proposed by (Vaswani et al., 2017). We initialize the Base T5 model released by (Raffel et al., 2019) which has 220 million parameters. We train the model with a 0.001 learning rate and an input/target length of 1024. With the provided TPU v2-8 on Google Colab, we train with the recommended setting of model parallelism 2 and batch size 128.

3.5 Multi-task Learning

The model is trained with maximum likelihood objective (that is using ”teacher forcing” (Williams and Zipser, 1989)) regardless of the text-code or code-text tasks. Therefore, for CoTexT, we leverage the potential for Multi-Task learning (Raffel et al., 2019) to complete both text-code and code-text generation on CodeSummarization and Code Refinement tasks. To specify the task our model should perform, we simply add a task-specific prefix to the input sequence. For example, when fine-tuning of the CodeSummarization task for each programming language, we simply prepend a prefix for each PL name (i.e., Java) to the input sequence.

4 Experiments

In this section, we will first describe the benchmark dataset for code intelligence CodeXGLUE, then we will explain the experimental setup on the tasks we perform and discuss the results of each task. The evaluation datasets are summarized in Table 3.

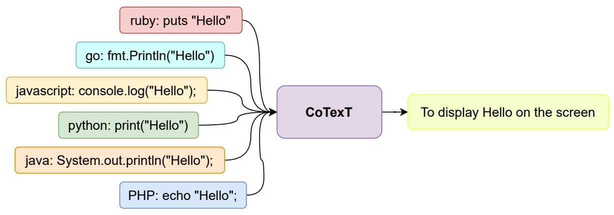
ruby: puts "Hello"
go: fmt.Println("Hello")
javascript: console.log("Hello");
python: print("Hello")
java: System.out.println("Hello");
PHP: echo "Hello";
CoTexT
To display Hello on the screen

Figure 2: An illustration about Multi-task learning

4.1 CodeXGLUE

General Language Understanding Evaluation benchmark for CODE (CodeXGLUE) (Lu et al., 2021) is a benchmark dataset to facilitate machine learning studies on code understanding and code generation problems. This dataset includes a collection of code intelligence tasks (both classification and generation), a platform for model evaluation, and a leaderboard for comparison. CodeXGLUE has 10 code intelligence tasks including code-text, text-code, code-code, and text-text scenarios. For CoTexT, we focus on Code Summarization, Code Generation, Code Refinement, and Defect Detection tasks.

4.2 Evaluation Tasks

We evaluate our programming language and natural language generation tasks on TPU v2-8 with the settings from the original T5 model (Raffel et al., 2019). The input length and target length for each task are described in Table 2.

4.2.1 Code Summarization

For Code Summarization, the objective is to generate a natural language description for a given code snippet. The task includes a CodeSearchNet dataset (Husain et al., 2019) with 6 different programming languages: Python, Java, Javascript, PHP, Ruby, Go. The data comes from public open-source non-fork GitHub repositories and the annotations are ex-

Table 2: The input and target sequence length settings for each self-supervised learning, code summarization, code generation, code refinement, and defect detection task

Task	Dataset	Task Type	Input Length	Target Length
Self-supervised Learning	CodSearchNet Corpus		1024	1024
	GitHub Repositories		1024	1024
Code Summarization	CodeSearchNet	Multi-Task	512	512
Code Generation	CONCODE	Single-Task	256	256
Code Refinement	Bugs2Fix$_{\text{small}}$ Bugs2Fix$_{\text{medium}}$	Multi-Task	512	512
Defect Detection	Devign	Single-Task	1024	5

tracted from function documentation as described in (Husain et al., 2019).

4.2.2 Code Generation

Text-to-Code Generation aims to generate a coded function given a natural language description. This task is completed using the CONCODE dataset (Iyer et al., 2018), a well-known dataset for Java language generation. Within the dataset, there are tuples which contain a natural language description, code environments, ad code snippets. The goal is to generate the correct Java function from the natural language description in the form of Javadoc-style method comments.

4.2.3 Code Refinement

Code Refinement, or Code Repair, aims to automatically correct bugs in Java code. We used the Bug2Fix corpus released by CodeXGLUE (Lu et al., 2021), which divides the task into 2 subsets: SMALL and MEDIUM The small dataset includes only Java code functions with fewer than 50 tokens. The medium dataset includes functions with 50-100 tokens.

4.2.4 Defect Detection

For Defect Detection tasks, we attempt to classify whether a PL snippet contains vulnerabilities that could lead to damaging outcomes such as resource leaks or DoS attacks. The task uses the Devign dataset (Zhou et al., 2019), which contains C programming language from open-source projects. This dataset is labeled based on security-related commits. For details on the annotation process, refer to (Zhou et al., 2019).

4.3 Experimental Setup

4.3.1 Baselines

We compare our model with some well-known pre-trained models:

- CodeGPT, CodeGPT-adapted are based on the architecture and training objective of GPT-2 (Budzianowski and Vulic, 2019). CodeGPT is pre-trained from scratch on CodeSearchNet dataset (Lu et al., 2021) while CodeGPT-adapted learns this dataset starting from the GPT-2 checkpoint.

- CodeBERT (Feng et al., 2020) employs the same architecture as RoBERTa (Liu et al., 2020) but aims to minimize the combined loss from masked language modeling and replaced token detection.

- PLBART (Ahmad et al., 2021b) is a Transformer-based model. BART (Lewis et al., 2019) is trained on PL corpora using three learning strategies: token masking, token deletion, and token infilling.

4.3.2 Performance Metrics

- BLEU (Papineni et al., 2002) is an algorithm which performs automatic evaluation of machine-translated text. This method calculates the n-gram similarity of a candidate translation compared to a set of reference texts. Similar to (Feng et al., 2020) and (Ahmad et al., 2021b), we use smooth BLEU-4 score (**?**) for Code Summarization and corpus-level BLEU score for all remaining tasks.

- CodeBLEU (Ren et al., 2020) is designed to consider syntactic and semantic features of

Table 3: Data statistics about Code Intelligence datasets

Category	Task	Dataset	Size			Language
			Train	Val	Test	
Code-Text	Code Summarization (Lu et al., 2021)	CodeSearchNet	164K	5.1K	10.9K	Java
			58K	3.8K	3.2K	Javascript
			251K	13.9K	14.9K	Python
			241K	12.9K	14K	PHP
			167K	7.3K	8.1K	Go
			24K	1.4K	1.2K	Ruby
Code-Code	Defect Detection (Zhou et al., 2019)	Devign	21K	2.7K	2.7K	C
	Code Refinement (Lu et al., 2021)	Bugs2Fix$_{small}$	46K	5.8K	5.8K	Java
		Bugs2Fix$_{medium}$	52K	6.5K	6.5K	
Text-Code	Code Generation (Iyer et al., 2018)	CONCODE	100K	2K	2K	Java

codes based on the abstract syntax tree and the data flow structure.

- Accuracy is the ratio of the number of generated sequences that harmonise the reference to the total number of observations.

5 Results

5.1 Multi-Task Learning

We first report the result of CoTexT in Multi-Task Learning tasks including Code Summarization and Code Refinement.

5.1.1 Code Summarization

For the Code Summarization task, we perform Multi-Task Learning by using the T5 framework (Raffel et al., 2019) to finetune CoTexT on 6 diferent programming language (Ruby, Javascript, Go, Python, Java, and PHP). The results of the Code Summarization task are shown in Table 5.

First, we observe that the base T5, which is pre-trained only on the general domain corpus (C4), is effective on this task. In fact, base T5 achieves higher overall results on the BLEU-4 metric compared to all other related models on the CodeXGLUE leaderboard. This shows the importance of domain-specific T5 models, which we expect to achieve superior results compared to base T5.

We further observe that CoTexT achieves state-of-the-art (SOTA) on the overall score, the Python-specific score, the Java-specific score, and the Go-specific score. While CoTexT does not significantly outperform other pre-trained models, we observe that CoTexT achieves SOTA on two very common programming languages (Python and Java) while still obtaining competitive results on other programming languages. We attribute this result to the large amount of training data for Python and Java compared to the other languages (training size described in Table 3). Based on this result, CoTeXT has the potential to further surpass competitor models as more training data becomes availible.

5.1.2 Code Refinement

We also tested CoTexT by performing multi-task learning for Code Refinement. In this case, both the small and medium test sets have a task registry with respective prefix prepending to the input sequence.

The Code Refinement results of each model are shown in Table 6. For this task, the base T5, which is pre-trained only on natural language text, does not perform well compared to other transformer-based models. Yet, after the training on a large programming language corpus, the result from CoTexT improves significantly on all metrics for both small and medium test sets. CoTexT achieves SOTA for all metrics on the small test set and on the accuracy metric for the medium test set.

5.2 Single-Task Learning

In addition to multi-task learning, we also evaluate CoTexT performance single-task learning with

Table 4: Test result on Code Generation task

Model	**Text2Code Generation**		
	EM	BLEU	CodeBLEU
PLBART	18.75	36.69	38.52
CodeGPT-adapted	20.10	32.79	35.98
CodeGPT	18.25	28.69	32.71
T5	18.65	32.74	35.95
CoText (1-CCG)	19.45	35.40	38.47
CoText (2-CC)	20.10	36.51	39.49
CoText (1-CC)	**20.10**	**37.40**	**40.14**

Notes: The best scores are in bold and second best scores are underlined. The baseline scores were obtained from the CodeXGLUE's Leaderboard (https://microsoft.github.io/CodeXGLUE/)

Table 5: Test result on Code Summarization task

Model	**All**	**Ruby**	**Javascript**	**Go**	**Python**	**Java**	**PHP**
RoBERTa	16.57	11.17	11.90	17.72	18.14	16.47	24.02
CodeBERT	17.83	12.16	14.90	18.07	19.06	17.65	**25.16**
PLBART	18.32	**14.11**	**15.56**	18.91	19.3	18.45	23.58
T5	18.35	14.18	14.57	19.17	19.26	18.35	24.59
CoTexT (1-CCG)	18.00	13.23	14.75	18.95	19.35	18.75	22.97
CoTexT (2-CC)	18.38	13.07	14.77	**19.37**	19.52	**19.1**	24.47
CoTexT (1-CC)	**18.55**	14.02	14.96	18.86	**19.73**	19.06	24.58

Notes: The best scores are in bold and second best scores are underlined. The baseline scores were obtained from the CodeXGLUE's Leaderboard (https://microsoft.github.io/CodeXGLUE/)

Table 6: Test result on Code Refinement task

	Small test set			**Medium test set**		
Model	BLEU	Acc(%)	CodeBLEU	BLEU	Acc(%)	CodeBLEU
Transformer	77.21	14.70	73.31	89.25	3.70	81.72
CodeBERT	77.42	16.40	75.58	**91.07**	5.16	**87.52**
PLBART	77.02	19.21	/	88.5	8.98	/
T5	74.94	15.3	75.85	88.28	4.11	85.61
CoTexT (1-CCG)	76.87	20.39	77.34	88.58	12.88	86.05
CoTexT (2-CC)	77.28	**21.58**	**77.38**	88.68	13.03	84.41
CoTexT (1-CC)	**77.79**	21.03	76.15	88.4	**13.11**	85.83

Notes: The best scores are in bold and second best scores are underlined. The baseline scores were obtained from the CodeXGLUE's Leaderboard (https://microsoft.github.io/CodeXGLUE/)

Table 7: Test result on Defect Detection task

Model	**Accuracy**
RoBERTa	61.05
CodeBERT	62.08
PLBART	63.18
T5	61.93
CoTexT (1-CCG)	**66.62**
CoTexT (2-CC)	64.49
CoTexT (1-CC)	65.99

Notes: The best scores are in bold and second best scores are underlined. The baseline scores were obtained from the CodeXGLUE's Leaderboard (https://microsoft.github.io/CodeXGLUE/)

a Code Generation Task and a classification task relating to Defect Detection.

5.2.1 Code Generation

In Table 4, we reported our results for the Code Generation task wherein natural language is translated into Java code. The result shows that our proposed model achieves SOTA results based on 3 metrics: Exact Match (EM), BLEU, and CodeBLEU. For each individual metric, CoTexT has only slightly outperformed other models (e.g both CoTexT and CodeGPT-adapted achieve 20.10 for EM). However, our model is consistently superior across the 3 metrics. Prior to CoTexT, CodeGPT-adapted was SOTA for the EM metric and PLBART was SOTA for the BLUE/CodeBLUE metrics. From this result, we infer that CoTexT has the best overall performance on this task and has great potential in the area of code generation.

5.2.2 Defect Detection

The Defect Detection results are shown in Table 7. Specifically, CoText outperforms the previous SOTA model (PLBART) by 3.44%. For this task, extra training on a large programming corpus allows CoTexT to outperform all other models and achieve SOTA results. The Defect Detection dataset consists of code written in the C programming language, which is not contained in our training data. Our model has a strong understanding of similar languages, and is thus able to perform Defect Detection in C with improved results compared to competitor models.

6 Conclusion

In this manuscript, we introduced CoTexT, a pre-trained language representation for both programming language and natural language. CoTexT focused on text-code and code-text understanding and generating. Leveraging the T5 framework (Raffel et al., 2019), we showed that pre-training on a large programming language corpus is effective for a diverse array of tasks within the natural language and programming language domain. CoTexT achieves state-of-the-art results on 4 CodeXGLUE code intelligence tasks: Code Summarization, Code Generation, Code Refinement, and Code Detection. For future work, we plan to test CoTexT on a broader range of programming language and natural language generation tasks, such as autocompletion or code translation.

References

Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021a. Unified pre-training for program understanding and generation.

Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021b. 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*.

Iz Beltagy, Arman Cohan, and Kyle Lo. 2019. Scibert: Pretrained contextualized embeddings for scientific text. *CoRR*, abs/1903.10676.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Pawel Budzianowski and Ivan Vulic. 2019. Hello, it's GPT-2 - how can I help you? towards the use of pre-trained language models for task-oriented dialogue systems. *CoRR*, abs/1907.05774.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. Codebert: A pre-trained model for programming and natural languages. *CoRR*, abs/2002.08155.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie LIU, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. Graphcode{bert}: Pre-training code representations with data flow. In *International Conference on Learning Representations*.

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

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2020. Codesearchnet challenge: Evaluating the state of semantic code search.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2018. Mapping language to code in programmatic context. *CoRR*, abs/1808.09588.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *CoRR*, abs/1808.06226.

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

Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. 2019a. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *CoRR*, abs/1901.08746.

Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. 2019b. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2019. BART: denoising sequence-to-sequence pretraining for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Ro{bert}a: A robustly optimized {bert} pretraining approach.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA. Association for Computational Linguistics.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *CoRR*, abs/1802.05365.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Ronald J. Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.*, 1(2):270–280.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2020. Xlnet: Generalized autoregressive pretraining for language understanding.

Pengcheng Yin, Graham Neubig, Wen tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data.

Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks.