CoDesc: A Large Code–Description Parallel Dataset

Masum Hasan[1]*, **Tanveer Muttaqueen**[1]*, **Abdullah Al Ishtiaq**[1]
Kazi Sajeed Mehrab[1], **Md. Mahim Anjum Haque**[1], **Tahmid Hasan**[1]
Wasi Uddin Ahmad[2], **Anindya Iqbal**[1], **Rifat Shahriyar**[1†]

[1]Bangladesh University of Engineering and Technology (BUET)
[2]University of California, Los Angeles

masum@ra.cse.buet.ac.bd, mahimanzum@gmail.com
{1505002.tm, 1505080.aai, 1505025.ksh}@ugrad.cse.buet.ac.bd
{tahmidhasan, anindya, rifat}@cse.buet.ac.bd
wasiahmad@cs.ucla.edu

Abstract

Translation between natural language and source code can help software development, by enabling developers to comprehend, ideate, search, and also write computer programs in natural language. Despite growing interest from the industry and the research community, this task is often difficult for the lack of large standard datasets suitable for training modern deep neural models, standard noise removal methods, and standard benchmarks. This leaves researchers to collect new small-scale datasets, resulting in inconsistencies across published works. In this study, we present CoDesc - a large parallel dataset containing 4.2 million Java methods and natural language descriptions. With extensive empirical analysis, we identify and remove prevailing noise patterns from the dataset. We demonstrate the proficiency of CoDesc in two complementary tasks for code–description pairs: code summarization and code search. We show that the dataset helps improve code search by up to 22% and achieve the new state-of-the-art in code summarization. We also show its effectiveness in pretrain-finetune setup, opening the possibility of building pretrained language models for source code. We release the dataset, a data processing tool, and a benchmark to encourage future research.[1]

1 Introduction

Neural models for natural language processing have benefited from large datasets and standard evaluation benchmarks (Wang et al., 2019b,a; Rajpurkar et al., 2016; Hermann et al., 2015; CommonCrawl). However, the programming language counterpart is lagging behind for the lack of such large datasets and benchmarks. To put this into perspective, the original Transformer network (Vaswani et al., 2017) was trained on WMT'14 English–German and English–French datasets (Bojar et al., 2014) containing 4.5 million and 36 million parallel sentences, respectively, whereas a similar network that achieved state-of-the-art results in source code summarization has been trained on only 69 thousand code-description pairs (Ahmad et al., 2020). We argue that the existing models used for programming language tasks in the literature have a significant scope of improvement given a large, good-quality dataset, and such a dataset is the missing link for effectively applying deep learning methods on programming languages.

In this work, we collect and release a large (4.2 million) Java source code - natural language (NL) parallel dataset along with denoising methods and baseline results. We apply our dataset to established works in both training from scratch and pretraining-finetuning setting and we demonstrate a notable performance gain in both settings. We gain 10% to 22% improvement over baseline code search models using CoDesc, and attain performances comparable to models having 8X more parameters. We achieve a new state-of-the-art BLEU score of 45.89 in code summarization by pretraining a Transformer network (Vaswani et al., 2017) with our dataset for two epochs. With extensive empirical analysis, we propose a set of noise removal techniques for the source code and the NL descriptions in our dataset.

Our work brings together several datasets and multiple tasks on the intersection of Natural Language Processing (NLP) and Software Engineering (SE), such as code summarization, code search and code synthesis, and allows researchers to compare their methods on the same benchmark. It also opens the door for building large pretrained models to jointly learn code and NL representations that can be leveraged in downstream tasks that do not

*Equal contribution.
†Corresponding author.
[1]https://github.com/code-desc/CoDesc

have adequate data, such as, code refactoring, clone detection, etc. as done by Feng et al. (2020).

2 Related Works

With the advent of deep learning, researchers have developed data-driven code search techniques (Gu et al., 2018; Sachdev et al., 2018; Cambronero et al., 2019; Husain et al., 2019; Sachdev et al., 2018). Husain et al. (2019) introduced CODESEARCHNET (CSN) CHALLENGE, a benchmark for code search techniques with 2.1 million code-NL parallel data in 6 programming languages, 6.4 million monolingual code data, a leader-board, and baseline results with 5 code search techniques. Feng et al. (2020) released CodeBERT, a pretrained BERT (Devlin et al., 2019) model trained on CSN data with Masked Language Modeling (MLM) and Replaced Token Detection (RTD) (Clark et al., 2020) objective that achieved a high performance in the CSN benchmark.

CODE-NN (Iyer et al., 2016) is a pioneering work in Deep Learning based code summarization. CONCODE (Iyer et al., 2018), DEEPCOM (Hu et al., 2018a), FunCom (LeClair and McMillan, 2019) are some notable dataset papers for the task. Wei et al. (2019) showed that both code summarization and code generation can be used to improve each other. Ahmad et al. (2020) achieved state-of-the-art results in source code summarization using a Transformer network (Vaswani et al., 2017). Clement et al. (2020) released a parallel corpus of 26 million Python methods and 7.7 million method-docstring pairs. They presented PyMT5, a text to text transformer that achieved a BLEU score of 8.59 for method generation and 16.3 for code summarization.

3 CoDesc Dataset

3.1 Data Sources

We collect our data from several sources and formulate rules for data cleaning. 5 of the authors spent 45-50 man-hours manually going over the dataset to identify patterns of noises in different data sources. Upon group discussion, common patterns were identified and a noise removal method was established.

One of the datasets used in CoDesc is CODESEARCHNET (CSN) CHALLENGE[2] (Husain et al., 2019) - a parallel method-description dataset for code search. Furthermore, other datasets used are DeepCom[3] (Hu et al., 2018a), CONCODE[4] (Iyer et al., 2018), FunCom[5] (LeClair and McMillan, 2019) - datasets created for code summarization. The CODESEARCHNET dataset originally contained 6 programming languages, from which the Java methods are directly used in CoDesc, however, the Python methods are used after being automatically translated to Java. We combine all aforementioned datasets to create CoDesc. More details about the individual datasets, their noise patterns, and examples are provided in Appendix A and B. Table 1 describes our data sources and their characteristics in detail.

CSN Python to Java Translation. To utilize maximum possible data from the CSN CORPUS, we translate the Python dataset to Java using TransCoder (Lachaux et al., 2020), a state-of-the-art, self-supervised neural source-to-source compiler. We modified and re-released the open-source implementation of TransCoder[6], enabling it to translate data in batches instead of one at a time, and resulting in a 16X faster translation. Upon empirical inspection, we found that the converted Java codes are human-readable and bear a strong resemblance to the original Python source. The converted codes seem correct to the human eye and their syntax matches with that of Java. However, the transcompiler suffers in some cases when converting to Java library methods, and converting from Python coding conventions that are not relevant to Java (e.g. use of SELF). These conversion errors, however, were not severe enough to affect our model to learn the NL-source code mapping.

3.2 Data Cleaning and Noise Removal

We created an easy-to-use, parameterized data processing tool for removing the different types of noise that we observed in our dataset. From the natural language descriptions, we remove symbols and characters that does not carry a meaning in a natural language description, such as, comment tags (e.g., `//`, `/*`, `*/`), stray code characters (e.g., `@`, `#`, `{`, `}`, etc.), HTML and XML tags, non-ASCII and escape characters, and some patterns of autogenerated tags (e.g., `@param`, `@return`, `@throws`, `@link`, `@code`,

[2]https://github.com/github/CodeSearchNet

[3]https://github.com/xing-hu/DeepCom

[4]https://github.com/sriniiyer/concode

[5]http://leclair.tech/data/funcom/

[6]https://github.com/code-desc/TransCoder.git

Name	#Projects	#Raw data	#Clean data	Code			Description		
				#Unique tokens	Avg len	≤200 (%)	#Unique tokens	Avg len	≤50 (%)
CSN-Java	N/A	542,991	490,169	284,214	140.41	83.42	168,507	25.14	89.42
DeepCom	9,714	588,108	424,028	306,422	128.35	84.04	91,933	17.80	94.76
FunCom	28,000	2,149,121	2,130,247	469,354	51.30	99.83	399,338	15.52	95.87
CONCODE	33,000	2,184,310	733,040	131,852	33.75	99.99	166,239	14.87	96.27
CSN-Py2Java	N/A	456,000	434,032	414,018	163.78	72.32	223,277	57.11	68.69
CoDesc (All)	N/A	5,920,530	4,211,516	1,128,909	77.97	93.53	813,078	21.04	92.28
Balanced *train-valid-test* split for CoDesc data									
train	-	-	3,369,218	991,395	78.01	93.53	718,204	21.05	92.28
valid	-	-	421,149	269,435	77.73	93.51	188,145	21.08	92.26
test	-	-	421,149	269,318	77.88	93.55	187,230	20.97	92.33

Table 1: Characteristics of CoDesc datasets and a balanced train-valid-test split. ≤200 indicates percent of data where source code is smaller than 200 tokens.

`@inheritdoc`, etc.). From the source code, we remove the comments, and the non-ASCII and escape characters. In previous studies, many meaningful data are discarded due to having some noisy patterns/characters either in the code or NL (Husain et al., 2019; Iyer et al., 2018; LeClair and McMillan, 2019). We identify and remove the noise only and preserve the data. This helps us reduce data loss in preprocessing.

For both source code and NL description, we split CamelCase and snake_case code tokens into subtokens (e.g., Camel Case, snake case) and separate linked alphabets and numbers (e.g., var0 to var 0) (Ahmad et al., 2020; LeClair and McMillan, 2019). After the aforementioned processing, we remove the data points where the source code is less than 3 tokens, or the description contains less than 2 alphabets (Husain et al., 2019). We lowercase the natural language as the case is not necessary for describing codes. We release our data processing tool along with the CoDesc dataset for applying the dataset to diverse tasks.

3.3 Dataset Characteristics

After the previous steps, we are left with nearly 4.2 million Java method and description parallel data. Table 1 describes characteristics of our dataset. The combined CoDesc dataset consists of more than one million unique tokens, which is significantly larger than natural language vocabulary (Chen et al., 2019). This can be partially attributed to inseparable multi-words (e.g. 'updateproductvariationlocalizeddeltaprice') in our dataset. Hence, we perform BPE (Sennrich et al., 2016) tokenization in our preprocessing pipeline. We also see that although the average token length of Java source codes vary in the different dataset sources, the natural language descriptions have a relatively uniform length. We create a balanced, deduplicated, and representative *train–valid–test* dataset by splitting each source-dataset in 8:1:1 ratio (Table 1).

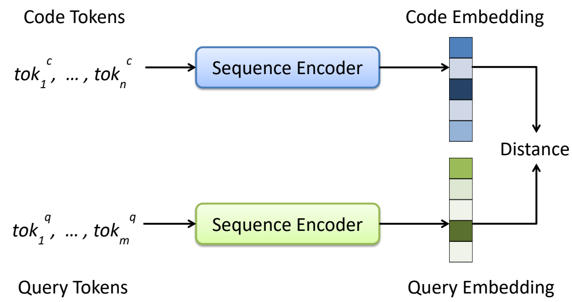

Figure 1: Code search model architecture; code and NL (query) encoders jointly train to reduce their embedded distance. During search, we select the code that is closest to the query in their shared embedding space.

4 Experiments

We evaluate our code-description parallel corpus from both directions – natural language to source code and vice versa. We use two well-known tasks: natural language code search and source code summarization. In this section, we demonstrate that models trained on CoDesc bring about a noticeable improvement over two established baselines in code search and code summarization. Each baseline task contains a separate tokenization scheme, and validation and test data. Considering the magnitude of the CoDesc dataset, it is likely that some of the test and validation data will be present in the CoDesc. Hence, prior to both Code Search and Code Summarization, all data is tokenized and formatted in the corresponding scheme, and repetition with test or validation data is removed from the train set.

Model	#Param	CSN test MRR	
		CSN-Java	CoDesc
NBOW	11.6 M	0.589	0.683
RNN	12.6 M	0.582	0.679
Sel-attn	13.6 M	0.583	0.723
1D Conv	16.4 M	0.520	0.686
Conv self-attn	16.0 M	0.509	**0.729**
State-of-the-art models			
RoBERTa (code)	125 M	0.721	
CodeBERT	125 M	**0.748**	

Table 2: Baseline models trained with default dataset and CoDesc, along with, comparison with SoTA pretrained models in CODESEARCHNET CHALLENGE Java test set.

Methods	BLEU	METEOR	ROUGE-L
Transformer	44.58	26.43	54.76
CoDesc pretrained	**45.89**	**28.01**	**56.59**

Table 3: Code summarization with SoTA Transformer network without and with pretraining with CoDesc.

4.1 Natural Language Code Search

We use the code search models used by Husain et al. (2019) that jointly trains a source code and an NL encoder networks to minimize their encoded vector distance (Figure. 1). We apply our dataset on the CODESEARCHNET (CSN) CHALLENGE (Husain et al., 2019) – a well-studied benchmark in the semantic code search literature. We train 5 different encoder networks (Table. 2) with the CSN Java dataset, and CoDesc respectively. We compare our results with CodeBERT and RoBERTa (code) (Feng et al., 2020), two pretrained models achieving state-of-the-art results in CSN Benchmark. They are trained with a Masked Language Modeling (MLM) (Devlin et al., 2019) objective on 2.1 million bimodal code-NL data, and 6.4 million unimodal data released with CODESEARCHNET.

Results. We use Mean Reciprocal Rank (MRR) – the standard evaluation metric for code search (Husain et al., 2019; Sachdev et al., 2018; Cambronero et al., 2019) as the evaluation criteria for code search. The results are reported in Table 2. We also show the performance of state-of-the-art models (Liu et al., 2019; Feng et al., 2020) that have nearly 8-10 times more parameters than the baseline networks and a more complex training objective. We achieve remarkably close performance with the state-of-the-art models with much simpler and smaller networks.

4.2 Source Code Summarization

For this task, we follow the methodology proposed by Ahmad et al. (2020). They used a seq2seq Transformer (Vaswani et al., 2017) network with 77M parameters with relative positional encoding (Shaw et al., 2018) and copying mechanism (See et al., 2017) and achieved state-of-the-art results.

Data preparation. Ahmad et al. (2020) used a Java dataset released by Hu et al. (2018b) and preprocessed by Wei et al. (2019) consisting of *training*, *validation*, and *test* datasets of size 69,708, 8,714, and 8,714 respectively. We refer to this training data as *train-small*. We create a new dataset *CoDesc-train* by combining *train-small* with CoDesc. We replace all literals as Wei et al. (2019) and tokenize the dataset using Character BPE Tokenization (Radford et al., 2019) to create the same size vocabulary as Ahmad et al. (2020).

Training. Similar to the original implementation, we train a Transformer model with *CoDesc-train* dataset using Adam optimizer (Kingma and Ba, 2014) with an initial learning rate of 10^{-4}, mini-batch size of 32, and dropout rate 0.2, vocabulary size 50k for code and 30k for output. However, we use maximum input length of 200 token instead of 150 from our observation of Table 1. Each epoch of the model took nearly 8 hours in an NVIDIA V100 16GB GPU. In comparison, the train-small dataset took 8.5 minutes only. For limitation of computational resource, we saved the network weights after training it with the large dataset for two epochs, and to be consistent with the original implementation, we trained the saved weights further with the *train-small* dataset for a maximum of 198 more epochs and perform an early stop if the validation performance does not improve for consecutive 20 epoch. This method of transfer learning provides the network parameters a more favorable initialization than random, and helps the network find better local minima.

Results. Table 3 shows that our two-stage training with CoDesc significantly outperforms state-of-the-art code summarization methods in all three evaluation metrics – BLEU (Papineni et al., 2002), METEOR (Banerjee and Lavie, 2005), and ROUGE-L (Lin, 2004). We observe that the pretrained model often generates more descriptive summary even when it achieves lower BLEU score (Fig. 2). We believe the model has more room for improvement if pre-trained with the CoDesc dataset further. We do not validate this for resource constraints, and we leave this as a future work.

```
public void makeImmutable(){
   if(mutable){
      if(results ! = null){
         int length = results.size();
         for(int i = NUM; i < length; i + +){
            Result result = (Result)results.get(i); result.makeImmutable();
         } results = Collections.unmodifiableList(results);
      } mutable = BOOL; } }
```

Human written: makes the object immutable
Transformer prediction (BLEU: 1.0): makes the object immutable

Figure 2: CoDesc pretrained model generates more descriptive summary, even in cases it achieves lower score.

Dataset	Raw data	Clean data	Inc. (%)
CSN (Java)	0.5870	0.6427	5.57
DeepCom	0.4677	0.6069	13.92
FunCom	0.5379	0.6366	9.87
CONCODE	0.5444	0.6234	7.90
CSN (Python2Java)	0.5081	0.5546	4.65
CoDesc (All)	0.5852	0.6826	9.74

Table 4: MRR of individual datasets (Section 3.1) before and after noise removal.

4.3 Ablation Study

To quantify the effect of individual data sources and our noise removal methodology, we train each dataset before and after our data cleaning using an NBOW model and test them in the CSN benchmark using their released test set (Table 4).

Although our collected data was already cleaned by the respective authors, Table 4 shows that the performance of every dataset improves drastically after our noise removal. Interestingly, without our extra layer of data cleaning, CoDesc dataset performs worse than training with only CSN data while being significantly larger. This shows the importance of a standard cleaning and processing method. Moreover, CSN (Java) appears to have the highest accuracy, which can be attributed to the fact that it came from the same distribution of data as the evaluation test set, and hence contains similar tokens and patterns (Husain et al., 2019). We can see from Table 4 that the model trained with CSN (Python2Java) achieves an MRR score of 0.5548 in CSN benchmark. Although this score is lower than other datasets, it is still a good indication that the translated data is helping the model is to learn NL-code association.

4.4 New Benchmark Results in Code Search

We provide a new set of benchmark results for CoDesc dataset in natural language code search. We train, validate, and test an NBOW, an RNN, and a Self-attn code search network with the balanced *train*, *validation*, and *test* data shown in Table 1. The three models achieve MRR score of **0.812**, **0.766**, and **0.839** respectively.

5 Discussion and Conclusion

In this work, we have accumulated CoDesc – a large code-description parallel dataset and established a baseline results. CoDesc brings a noteworthy improvement in two established tasks: code search and code summarization. We believe that this dataset and benchmark will serve as a base for future studies on code-description joint tasks. We also show that automatically translated source code from a source-to-source compiler can be applied in a code-NL parallel task, suggesting that, translating our Java dataset to other programming languages can also be helpful.

The most striking finding of our study is that, by training with 2X larger parallel data, we achieve equivalent performance to models having 8X parameters (Feng et al., 2020) in code search. This raises an interesting question: are we fully utilizing the model capacities in code–description studies? From our pretraining results in code summarization, it can be reasonably assumed that pretraining with our large dataset the larger models will also improve further. Given that the focus of this study is to propose and validate the CoDesc dataset, we refrain from proposing new methodologies. In future works, we wish to apply new techniques for code search, code summarization, along with exploring our dataset for general-purpose code synthesis, where the best models are still struggling in accuracy (Wei et al., 2019; Yin and Neubig, 2017).

References

Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2020. A transformer-based approach for source code summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4998–5007, Online. Association for Computational Linguistics.

Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics.

Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. 2014. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA. Association for Computational Linguistics.

Jose Cambronero, Hongyu Li, Seohyun Kim, Koushik Sen, and Satish Chandra. 2019. When deep learning met code search. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, pages 964–974.

Wenhu Chen, Yu Su, Yilin Shen, Zhiyu Chen, Xifeng Yan, and William Yang Wang. 2019. How large a vocabulary does text classification need? a variational approach to vocabulary selection. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3487–3497, Minneapolis, Minnesota. Association for Computational Linguistics.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *ICLR*.

Colin Clement, Dawn Drain, Jonathan Timcheck, Alexey Svyatkovskiy, and Neel Sundaresan. 2020. PyMT5: multi-mode translation of natural language and python code with transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9052–9065, Online. Association for Computational Linguistics.

CommonCrawl. Common crawl. `https://commoncrawl.org/`. Accessed: 2021-01-31.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

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. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online. Association for Computational Linguistics.

Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep code search. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, page 933–944.

Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS 2015)*, pages 1693–1701, Montreal, Canada.

Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018a. Deep code comment generation. In *Proceedings of the 26th Conference on Program Comprehension*, ICPC '18, page 200–210.

Xing Hu, Ge Li, Xin Xia, David Lo, Shuai Lu, and Zhi Jin. 2018b. Summarizing source code with transferred api knowledge. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 2269–2275. International Joint Conferences on Artificial Intelligence Organization.

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

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2073–2083, Berlin, Germany. Association for Computational Linguistics.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2018. Mapping language to code in programmatic context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1643–1652, Brussels, Belgium. Association for Computational Linguistics.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Marie-Anne Lachaux, Baptiste Roziere, Lowik Chanussot, and Guillaume Lample. 2020. Unsupervised translation of programming languages. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (NeurIPS 2020)*.

Alexander LeClair and Collin McMillan. 2019. Recommendations for datasets for source code summarization. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3931–3937, Minneapolis, Minnesota. Association for Computational Linguistics.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

C. Lopes, S. Bajracharya, J. Ossher, and P. Baldi. 2010. UCI source code data sets.

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 of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8).

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Saksham Sachdev, Hongyu Li, Sifei Luan, Seohyun Kim, Koushik Sen, and Satish Chandra. 2018. Retrieval on source code: a neural code search. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2018, page 31–41.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS 2017)*, page 6000–6010, Long Beach, California, USA.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019a. SuperGLUE: A stickier benchmark for general-purpose language understanding systems. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems (NeurIPS 2019)*, pages 3266–3280, Vancouver, Canada.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019b. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Bolin Wei, Ge Li, Xin Xia, Zhiyi Fu, and Zhi Jin. 2019. Code generation as a dual task of code summarization. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems (NeurIPS 2019)*, pages 6563–6573, Vancouver, Canada.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver, Canada. Association for Computational Linguistics.

A Dataset Details

CODESEARCHNET (CSN) CORPUS. Husain et al. (2019) released the CSN CORPUS[7], a code search dataset for 6 programming languages. Despite the authors' effort for data cleaning, in our observation, CSN CORPUS is found to be one of the noisiest. The dataset contained duplicate descriptions, inseparable multi-words (e.g. updateproductvariationlocalizeddeltaprice, updatelocationinventory), XML tags (e. g. <tt>, <soup>, <sub>, <pre>, <p>), non-English documentation, non-ASCII characters, escape characters, unwanted symbols (e.g. @, #, {, }, etc.), deprecated methods and descriptions, comments inside code, annotations (e.g. @link, @code, @inheritdoc) inside description, etc. Some datapoints (27,471) in our Python to Java translation are truncated by TransCoder, which have been marked with a special flag in our released dataset.

DeepCom. Hu et al. (2018a) released a dataset of 588,108 Java method and documentation pairs collected from 9,714 GitHub projects for code summarization[8]. Similar to CODESEARCHNET (Husain et al., 2019), they considered the first sentence of a documentation as the summary of the method as it typically describes the functionalities of Java methods. They filter out empty and single world descriptions and the setter, getter, constructors, and test methods, since they are easy for a model to summarize. In our manual analysis, we found HTML tags (e.g. <tt>, <p>, <p class = ...> , <li>, <ul>), comment tags, annotations, escape characters inside descriptions, empty parentheses as descriptions, repetitive and non-meaningful descriptions, comments inside source code, etc. Despite the authors' claim, we found numerous test methods in the dataset, which were mostly meaningful data.

CONCODE. Iyer et al. (2018) released a dataset named CONCODE, collected by mining nearly 33,00 GitHub repositories[9]. In their preprocessed dataset, they replaced the names of the identifier and method names with generic terms, (e.g., arg0, loc0, function, etc.) and replaced all string literals with constants. This created a discrepancy with the other datasets, hence, we opted for their unprocessed dataset rather than the processed version. The unprocessed dataset released with CONCODE contained approximately 2.1 million Java methods and lowercased Javadoc-style document pairs. Upon duplicate removal, we were left with 733,878 datapoints.

Although some noises were present in this dataset, we found this data to be least noisy in manual observation. We find that because of lower casing the documentations, some CamelCase tokens became inseparable. The dataset also contained non-English comments with English alphabets (mostly Italian). We found these documents hard to identify and remove.

FunCom. LeClair and McMillan (2019) released a dataset of over 2.1 million pairs of Java methods and one-sentence method descriptions from over 28k Java projects[10]. They collected this dataset by filtering over 51 million Java methods from UCI Source Code datasets (Lopes et al., 2010). In their preprocessing step, LeClair and McMillan (2019) removed all datapoints where the method is more than 100 tokens long, or the method description is over 13 tokens or below 3 tokens.

In our observation of this dataset, we found method descriptions containing HTML tokens (e.g. <tt>, annotations (e.g., @link, @param), comment tokens, unwanted symbols, solely non-alphabetic characters, etc. It also contained comments inside methods, and a large portion of the data were getter, setter, tester, and toString methods.

B Sample Data

```
Object getBean(String beanName){
   if(null == beanName){
      return null;
   }
   return
       applicationContext.getBean(beanName);
}
```

Description: this method is used to retrieve a bean by its name. note that this may result in new bean creation if the scope is set to "prototype" in the bean configuration file. (CONCODE)

```
@Exported
   public boolean isIdle() {
      lock.readLock().lock();
      try {
         return workUnit == null &&
             executable == null;
      } finally {
         lock.readLock().unlock();
      }
   }
```

[7] https://github.com/github/CodeSearchNet

[8] https://github.com/xing-hu/DeepCom

[9] https://github.com/sriniiyer/concode

[10] http://leclair.tech/data/funcom/

Description: returns true if this executor is ready for action. (CodeSearchNet)

```
public static void dbCommand(ParserArgs
    args){
  final Synergy synergy
      =(Synergy)args.get("synergy");
  if(synergy.reset){
    synergy.resetDb();
    synergy.update = true;
  }
  if(synergy.update){
    synergy.updateDb();
  }
}
```

Description: manages synergy db state (CodeSearchNet-Python to Java)

```
public void sort (boolean
    transformChanged) {
   if (list Size > 1){
      if (tlist == null || tlist.length
          != list.length){
         tlist = list.clone();
      } else {
         System.arraycopy(list, 0,
             tlist, 0, list.length);
      }
      if (transform Changed) {
         for(int i = 0; i < listSize;
             i++) {
            list[i]
            .computeLastDistance(owner);
         }
      }
      SortUtil.msort(tlist, list, 0,
          list Size - 1, c);
   }
}
```

Description: sorts the elements in the list acording to their comparator. there are two reasons why lights should be resorted. first, if the lights have moved, that means their distance to the spatial changed. second, if the spatial itself moved, it means the distance from it to the individual lights might have changed. (FunCom)

```
protected void hideTabs(){
  if (getPageCount() <= 1) {
    setPageText(0,"");
    if (getContainer() instanceof
        CTabFolder) {
      ((CTabFolder)getContainer())
      .setTabHeight(1);
      Point point =
          getContainer().getSize();
      getContainer().setSize(point.x,point.y
          + 6);
    }
  }
}
```

Description: if there is just one page in the multi - page editor part , this hides the single tab at the bottom. (DeepCom)