CodeMod: The Code Modification Dataset

Anonymous ACL submission

Abstract

The continuous emergence of large language models specially capable to deal with programming languages makes crucial the development of better benchmarks that appraise them in terms of their skills. In this paper we introduced CodeMod, the first benchmark dataset for code modification. This dataset is evaluated both in zero-shot and fine-tuned configurations utilizing the most recent Large Language Models (LLMs) for code. We also demonstrate its usefulness by evaluating the performance of fine tuned models in terms of code synthesis performance. We show up to 5 points of improvement on pass@1 performance on the HumanEval benchmark. This new dataset will be a new addition to the code benchmark landscape.

1 Introduction

The advent of Large Language Models (LLMs) has redrawn the landscape of the world of Artificial Intelligence. The availability of larger and open-source models trained with huge amounts of data have pushed capabilities of these models to limits that were out of reach just a few years ago. Nowadays, LLMs are dealing with more diverse data, not limited to general natural language, but also domain specific data such as medical (Lee et al., 2020), legal (Chalkidis et al., 2020), or financial (Araci, 2019), diverse modalities such as images (Trinh et al., 2019; Chen et al., 2021c), videos (Sun et al., 2019), stocks prices (Wang et al., 2022), proteins (Brandes et al., 2022), etc.

More recently, there has been an tremendous growth in the area of LLMs which are trained with data from the programming language domain (Chen et al., 2021a; Nijkamp et al., 2023; Li et al., 2023; Si et al., 2023). Several close- and open-source models have rushed the community showing impressive skill. These models are capable of tackling several aspects of the coding domain,

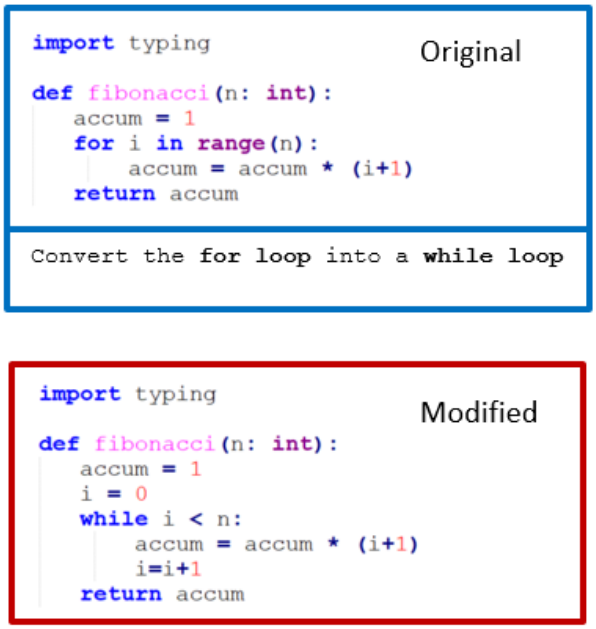

Figure 1: An excerpt of the expected behaviour of the code modification task.

such as code synthesis, code translation, code fixing, code search, among others.

With the growth of code related LLMs, also grows the interests of the development of datasets and benchmarks intended to assess the capabilities of these models. Example of such benchmarks can are CONCODE (Iyer et al., 2018) for code search, CodeSearchNet (Husain et al., 2019) for search an summarisation, HumanEval (Chen et al., 2021b) and MBPP (Austin et al., 2021), for program synthesis), XLCoST (Zhu et al., 2022) (code translation), BigCloneBench (Svajlenko et al., 2014) (clone detection), Bugs2Fix (Tufano et al., 2018) (code fixing), and CodeXGlue (Lu et al., 2021), a suite that cluster many of these datasets.

Existing datasets that involve code generation fall into this definition: given an input in some modality, generate a piece of code that fits the expected output. Examples of that include text-to-

code generation, code translation, program synthesis based on unit tests, etc. On another side, code repair datasets such as Bugs2Fix, aim at modifying the input code by fixing an existing where a bug to fix that should be obvious given the context. None of these existing datasets provide both training and evaluation sets for where the output is a piece of code that should result from the input code and a natural language description of the modification to apply.

We propose CodeMod, "code modification" dataset which is intended to modify an functioning piece of code given a description of the expected behaviour. This dataset is closer to multi step code synthesis than other text-to-code generation datasets such as HumanEval or MBPP.

Our contributions are threefold: (1) we introduce CodeMod, a new dataset for training and evaluation of Language models for code modification 2) we benchmark well known LLMs in both zero-shot and fine-tuned configurations 3) We further demonstrate that our dataset can be also used to improve models capable of program synthesis.

2 Related Work

From the past few years there has been a growth in the interest to find new and better ways to assess the strength of models able to perform program synthesis. There have been many datasets aim at mapping different forms of natural language code. Examples of that are Magic the Gathering (MTG) and Hearthstone (HS) introduced by Ling et al. (2016), which generated code for cards in Trading Card Games, Django (Oda et al., 2015; Ling et al., 2016), which was introduced to generate pseudo-code from actual Python code, and NAPS (Zavershynskyi et al., 2018) and SPoC (Kulal et al., 2019) which do the opposite.

Along with the advent of Large Language Models for code, the discipline was popularised among the Natural Language Processing community. An example of that was the presentation of CodeBLEU (Ren et al., 2020), an extension of the popular metric BLEU (Papineni et al., 2002) but adapted to be used in the coding domain. CodeBLEU extends the assessment of a generated piece of code not only in terms of word overlap against a reference solution, but also considering the syntax and the semantics of code, comparing it in terms of the Abstract Syntax Tree (AST) and the data-flow of the reference solution. Moreover, inspired in recent general language understanding benchmarks such as GLUE (Wang et al., 2018) and XTREME (Hu et al., 2020), a new suite, CodeXGlue, was introduced by Lu et al. (2021), CodeXGlue consists in a set of benchmarks for both code understanding a generation, ,including 14 tasks, among them the task of text-to-code generation or program synthesis. The dataset consists in 100k training instances, and it's evaluated in terms of BLEU and CodeBLUE. Chen et al. (2021a) introduced HumanEval. This dataset measures functional correctness for synthesizing programs from docstrings. It consists of 164 original programming problems written in Python, assessing language comprehension, algorithms, and simple mathematics, with some comparable to simple software interview questions. It benchmarks models based on the capacity to pass unit tests that check for the correct execution of the function. Soon after the introduction of HumanEval, MBPP was presented (Austin et al., 2021). The Mostly Basic Programming Problems (MBPP) dataset, is similar is similar to the former, but include a much larger set of problems (974 in total) and in addition to the ability to be used both in a few-shot and fine-tuned configuration. More recently Nijkamp et al. (2023) introduced MTPB (Multi-Turn Programming Benchmark). MTPB consists of 115 handcrafted problems, each of which includes a multi-step descriptions in natural language (prompt). Contrary to the previous datasets, to solve a problem, a model needs to synthesize functionally correct subprograms following the description at the current step and considering descriptions and synthesized subprograms at previous steps.

3 Dataset

In this section we explain the process we applied to create the CodeMod dataset. It consists of several filtering steps to guarantee the quality and usefulness of the resulting instances.

3.1 Filtering

The raw data for CodeMod consists of git commits taken from StarCoder (Li et al., 2023), which is gathered from BigQuery[1]. Only single-file commits of repositories with the same licenses and file extension as used in The Stack (Kocetkov et al., 2022) are used. All repositories from users that

[1] https://cloud.google.com/bigquery/public-data/

Model	BLEU
baseline	88.15
zero-shot	
starcoderbase-1b	28.88
santacoder	20.98
codet5p-770m	4.51
fine-tuned	
starcoderbase-1b	**92.47**
santacoder	81.83
codet5p-770m	76.79

Table 1: Zero shot and finetuned evaluation on the CodeMod Dataset

Model	BLEU
baseline	**88.15**
zero-shot	
Starchat-beta	66.19
CodeLlama-34b-Instruct-hf	85.43
GPT3.5	71.21

Table 2: Zero shot evaluation of Larger models on the CodeMod Dataset

have opted out of The Stack are removed. StarCoder already employs some filters to the commits as described in their paper. Notably, excessively long files are removed, and JSON, YAML, XML and HTML are subsampled. These languages are mostly removed from this dataset, due to the abundance of documentation using these formats.

By employing additional filtering processes to this data, a higher quality corpus of code pairs is obtained. These pairs consist of before code and some modified after code, with a related commit message. The modification made should be a change in functionality to working code, not a fix/repair to broken or dysfunctional code.

A number of filters targeting certain words are added to remove non-useful messages i.e. "fix" to remove examples of code repair. Additionally some basic filters e.g. the maximum length of the file are applied. A summary of the basic filters and the removed terms, with a short justification, is provided in the Appendix.

Commit messages of less than 9 words are removed to improve quality, as longer messages are more verbose and tend to include more specific information describing the change than those with less words.

The similarity (Reimers and Gurevych, 2019) of the before and after code is measured, and filtered so that the before and after code are not too dissimilar. In addition, a minimum of 1 and a maximum of 15 lines of code can be added or modified (complete removal of a line does not count as a modified). Overly large refactoring of code is considered to be closer to generation of new code than a modification to functionality.

In order to remove messages that hold low information, a few metrics are taken.

The perplexity (Meister and Cotterell, 2021) of the commit message is measured against the corresponding 'after' example of code. If the commit message describes the change, the perplexity should be lower due to sharing relevant terms and semantics.

Similarly, a metric that measures the similarity docstrings and codes is used (Husain et al., 2019; Lu et al., 2021). In this work, the similarity between the commit message and the code is used. The commit message, if high in quality, should show some resemblance to a natural language query that someone might use to search for the associated piece of code. This includes features such as use of specific words/names or numbers and description of functionality.

The commit message is also compared to some chosen sample sentences via SentenceTransformer (see below).

```
'Fixed a bug in the code'
'initial commit'
'Updated date and time'
```

If the commit message shows very high similarity to the provided sentences, they are removed for lack of relevant information, as a message with more specific knowledge would separate itself from those with low semantic value. This is effective at removing trivial commits but not narrowing down to higher quality samples.

A few other metrics were measured, e.g. surprisal (Oh et al., 2021) of the commit message and the number of modified lines, but these provided little information as evident by their very sharp dis-

Model	ZS	FT
codet5p-770m	15.24	18.90
starcoderbase-1b	15.17	20.12
santacoder	18.29	**21.95**

Table 3: Performance on HumanEval pass@1, **ZS**: zero-shot and **FT**, after finetuning with CodeMod

tributions, which provide no meaningful threshold from which to filter. Correlation between metrics is also measured to investigate whether any of them used are performing identical tasks. None of the metrics show statistically significant correlation toward one another and so the chosen metrics should be justified in their combined use as separate values.

3.2 The Final Dataset

The final dataset consists of **52,171** examples from the original 7.6 million from the raw data. It is packaged in a parquet in an identical format to the original data, shown here:

The distribution of programming languages in the final dataset is uncertain due to the raw data not providing filenames or any other relevant metadata. Some approximation is made utilising GuessLang [2], with the most common languages being Python, followed by Java, Javacript and C.

4 Experiments

To validate the quality / usefulness of the dataset, 3 models were finetuned using a shortened version of the dataset. The shortening consists of taking only examples in which the after code is under 1000 characters in length. This is done to accomodate the relatively small context windows of the models. The models include, starcoderbase-1b (Li et al., 2023), santacoder (Si et al., 2023) and codet5p-770m (Wang et al., 2021). In all cases the entire model was trained. We show improved BLEU results using the dataset for evaluation against a baseline using the before code and the after code.

The results show such a large improvement over the zero shot as the before and after code shares much similarity. As such the model will greatly improve its results as it learns to copy over code. Due to the commit message, the model does make changes to the code, in many cases to its own detriment. StarCoder (1 billion parameters) manages to improve over the baseline by adding useful information from the commit message into the code. While a small improvement, it is significantly better than santacoder and codet5p-770m. Santacoder may be a more capable model than the results suggest, as it is trained only on Python, Java and Javascript which may explain the discrepancy. They are all trained for the same number of steps, but due to their performance compared to baseline may be undertrained, or simply failing to learn due to the consistently large length of the corpus.

The zero shot evaluation is done on a smaller randomly sampled subset of the data, and may need a better prompting strategy as sometimes the model will not reproduce the entire code but only return a changed snippet. The larger models do a much better job of maintaining the original code, and adding less unnecessary changes. They do however, especially gpt3.5, add comments where there otherwise would not, in turn lowering the BLEU score.

The HumanEval (Python) scores improve by a few points across the board, it is worth noting that all 3 models are non-instruction tuned and that SantaCoder is trained on Python, Java and Javascript only.

5 Conclusion

In this work we comment about the existing datasets for evaluation of Large Language models for code. We assess their limitation and present CodeMod, a new dataset and benchmark for the evaluation of code synthesis task where the expected output is a modified version of the input. Our contributions are: i) We describe the process for extraction and filtering of the CodeMod dataset based on existing commit information extracted from open source repositories. ii) We show that our benchmark is useful both in zero-shot and fine tune configurations, showing that is challenging for established LLM for code. iii) we proved that our dataset is also useful as training signal for the task of code synthesis, showing up to 5 points of improvement on HumanEval performance.

[2] https://github.com/yoeo/guesslang

6 Limitations

Presented in a short paper, the current work is limited in all the experimentation that could have been included. The main limitation we see is the evaluation of CodeMod is limited to be benchmarked in terms of BLEU score. While Pass@k metric, popularized by Chen et al. (2021a) is the dominant metric these days, the lack of proper unit test for most of the original source code made the evaluation in those terms, unfeasible.

7 Ethical Considerations

This research piece introduces a new resource meant to improve performance of Large Language Models for Code. Although this work do not deal with datasets or tasks which might be ethically concerning the medical domain, the authors recognize that any work that aims to automatize tasks which are currently carried out almost exclusively by humans should be of particular ethical interest. We recognize that better language models, in particular on the coding domain, might constitute a risk for future jobs positions, in particular for those who are just starting a career in software engineering.

References

Dogu Araci. 2019. Finbert: Financial sentiment analysis with pre-trained language models. *CoRR*, abs/1908.10063.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Nadav Brandes, Dan Ofer, Yam Peleg, Nadav Rappoport, and Michal Linial. 2022. Proteinbert: a universal deep-learning model of protein sequence and function. *Bioinformatics*, 38(8):2102–2110.

Ilias Chalkidis, Manos Fergadiotis, Prodromos Malakasiotis, Nikolaos Aletras, and Ion Androutsopoulos. 2020. LEGAL-BERT: The muppets straight out of law school. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2898–2904, Online. Association for Computational Linguistics.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021a. Evaluating large language models trained on code.

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. 2021b. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Xiangning Chen, Cho-Jui Hsieh, and Boqing Gong. 2021c. When vision transformers outperform resnets without pre-training or strong data augmentations. *arXiv preprint arXiv:2106.01548*.

Junjie Hu, Sebastian Ruder, Aditya Siddhant, Graham Neubig, Orhan Firat, and Melvin Johnson. 2020. Xtreme: A massively multilingual multi-task benchmark for evaluating cross-lingual generalization. *CoRR*, abs/2003.11080.

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.

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.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. 2022. The stack: 3 tb of permissively licensed source code.

Sumith Kulal, Panupong Pasupat, Kartik Chandra, Mina Lee, Oded Padon, Alex Aiken, and Percy S Liang. 2019. Spoc: Search-based pseudocode to code. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. 2020. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim,

Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. Starcoder: may the source be with you!

Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, Fumin Wang, and Andrew Senior. 2016. Latent predictor networks for code generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 599–609, Berlin, Germany. Association for Computational Linguistics.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. 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. *CoRR*, abs/2102.04664.

Clara Meister and Ryan Cotterell. 2021. Language model evaluation beyond perplexity.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. Codegen: An open large language model for code with multi-turn program synthesis.

Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Learning to generate pseudo-code from source code using statistical machine translation. In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 574–584.

Byung-Doh Oh, Christian Clark, and William Schuler. 2021. Surprisal estimators for human reading times need character models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3746–3757, Online. Association for Computational Linguistics.

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.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks.

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. *CoRR*, abs/2009.10297.

Shuzheng Si, Zefan Cai, Shuang Zeng, Guoqiang Feng, Jiaxing Lin, and Baobao Chang. 2023. SANTA: Separate strategies for inaccurate and incomplete annotation noise in distantly-supervised named entity recognition. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 3883–3896, Toronto, Canada. Association for Computational Linguistics.

Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid. 2019. Videobert: A joint model for video and language representation learning. *CoRR*, abs/1904.01766.

Jeffrey Svajlenko, Judith F Islam, Iman Keivanloo, Chanchal K Roy, and Mohammad Mamun Mia. 2014. Towards a big data curated benchmark of inter-project code clones. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 476–480. IEEE.

Trieu H. Trinh, Minh-Thang Luong, and Quoc V. Le. 2019. Selfie: Self-supervised pretraining for image embedding. *CoRR*, abs/1906.02940.

Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. 2018. An empirical study on learning bug-fixing patches in the wild via neural machine translation. *CoRR*, abs/1812.08693.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Chaojie Wang, Yuanyuan Chen, Shuqi Zhang, and Qiuhui Zhang. 2022. Stock market index prediction using deep transformer model. *Expert Systems with Applications*, 208:118128.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. 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, pages 8696–8708, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Maksym Zavershynskyi, Alexander Skidanov, and Illia Polosukhin. 2018. NAPS: natural program synthesis dataset. *CoRR*, abs/1807.03168.

Ming Zhu, Aneesh Jain, Karthik Suresh, Roshan Ravindran, Sindhu Tipirneni, and Chandan K. Reddy. 2022. Xlcost: A benchmark dataset for cross-lingual code intelligence.

8 Appendix

The specifics of the filtering are listed below:

Code including the following regex expression are removed:

```
---
===
\[(.*)\]\((.*)\)
</p>
</div>
\.\.(.*)\:\:
copyright
```

After a preliminary cleaning of commit numbers, commit messages including the following regex expressions are removed:

```
#
->
version
bump
documentation
(.*)\.(.*)\.(.*)
\*\*\* empty log message \*\*\*
readme
fix
error
refactor
commit
(.*):(.*):(.*)PM
(.*):(.*):(.*)AM
license
```

The after code is filtered to be between 5 and 500 lines (inclusive). The number of added must be a minimum of 1 and a maximum of 15.

Similarity is calculated via sentence-transformers/all-MiniLM-L6-v2 from huggingface, and the before code and after code must be greater or equal to 0.8

The commit message sentence comparison is also done with SentenceTransformer and filtered to below 0.4

CodeSearchNet metric uses the implementation from Nokia[3] and is filtered to examples above 0.25

Perplexity is calculated using GPT-2 as in huggingface[4] and is filtered to examples under 15.

Surprisal[5] was calculated using GPT-2, but was not utilised in the final process.

[3] https://github.com/nokia/codesearch

[4] https://huggingface.co/docs/transformers/perplexity

[5] https://github.com/aalok-sathe/surprisal