The Stack: 3 TB of permissively licensed source code

Anonymous authors
Paper under double-blind review

Abstract

Large Language Models (LLMs) play an ever-increasing role in the field of Artificial Intelligence (AI)–not only for natural language processing but also for code understanding and generation. To stimulate open and responsible research on LLMs for code, we introduce The Stack, a 3.1 TB dataset consisting of permissively licensed source code in 30 programming languages. We describe how we collect the full dataset, construct a permissively licensed subset, present a data governance plan, discuss limitations, and show promising results on text2code benchmarks by training 350M-parameter decoders on different Python subsets. We find that (1) near-deduplicating the data significantly boosts performance across all experiments, and (2) it is possible to match previously reported HumanEval and MBPP performance using only permissively licensed data. We make the dataset available at `ANONYMIZED`, provide a tool called "Am I in The Stack" for developers to search The Stack for copies of their code, and provide a process for code to be removed from the dataset by following the instructions at `ANONYMIZED`.

1 Introduction

Large Language Models (LLMs) have emerged as a powerful tool for Natural Language Processing (NLP) (Brown et al., 2020; Bommasani et al., 2021; Zhang et al., 2022; Chowdhery et al., 2022; BigScience Workshop, 2022). These large transformer models are pre-trained on large internet corpora and have shown impressive zero and few-shot performance on numerous NLP tasks, often by prompting the model with a natural language description of the task at hand (Brown et al., 2020). More recently, researchers have started exploring LLMs for coding applications (Chen et al., 2021; Fried et al., 2022; Nijkamp et al., 2022). Code LLMs are trained on large collections of source code and enable the synthesis of programs from both natural language descriptions and other code snippets. Such models can assist professional developers with programming tasks, for example, by auto-completing code snippets, generating docstrings for a given function signature and body, or suggesting unit tests for a codebase.

One of the challenges faced by researchers working on code LLMs is the lack of openness and transparency around the development of these systems. While a handful of papers on code LLMs has been published, they do not always give full insight into the development process. Some research groups have made their code LLMs available through a paid API service (Chen et al., 2021) or a commercial product[1]. Other groups have published the model weights (Nijkamp et al., 2022; Fried et al., 2022) but did not release the training data. It is, therefore, difficult for researchers to fully reproduce these models because they first need to investigate the nuanced details of the data processing. For example, several groups have reported that (near) deduplication of the training data is an important preprocessing step for both natural language (Kandpal et al., 2022; Hernandez et al., 2022) and source code (Allamanis, 2019). Instead of letting research groups independently iterate over such preprocessing details, we argue that the research community would make progress faster if high-quality pre-training datasets, supported by data cards (Gebru et al., 2021; Bender & Friedman, 2018), were more broadly shared.

[1]For example, Microsoft's Copilot (`https://github.com/features/copilot`) and Amazon's CodeWhisperer (`https://aws.amazon.com/codewhisperer/`).

Language	**The Stack**†	**CodeParrot**†	**AlphaCode**	**CodeGen**	**PolyCoder**†
Assembly	2.36	0.78			
Batchfile	1.00	0.7			
C	222.88	183.83		48.9	55
C#	128.37	36.83	38.4		21
C++	192.84	87.73	290.5	69.9	52
CMake	1.96	0.54			
CSS	145.33	22.67			
Dockerfile	1.95	0.71			
FORTRAN	3.10	1.62			
GO	118.37	19.28	19.8	21.4	15
Haskell	6.95	1.85			
HTML	746.33	118.12			
Java	271.43	107.7	113.8	120.3	41
JavaScript	486.20	87.82	88	24.7	22
Julia	3.09	0.29			
Lua	6.58	2.81	2.9		
Makefile	5.09	2.92			
Markdown	164.61	23.09			
Perl	5.50	4.7			
PHP	183.19	61.41	64		13
PowerShell	3.37	0.69			
Python	190.73	52.03	54.3	55.9 (217.3)	16
Ruby	23.82	10.95	11.6		4.1
Rust	40.35	2.68	2.8		3.5
Scala	14.87	3.87	4.1		1.8
Shell	8.69	3.01			
SQL	18.15	5.67			
TeX	4.65	2.15			
TypeScript	131.46	24.59	24.90		9.20
Visual Basic	2.73	1.91			
Total	3135.95	872.95	715.1	314.1	253.6

Table 1: The size of The Stack (in GB) compared to other source code datasets used for pre-training LLMs. † indicates the dataset is publicly released. The Stack is more than three times the size of CodeParrot, the next-largest released code dataset.

We believe openness around the pre-training data is also important given the ongoing legal discussions around the use of open-source code repositories for training (commercial) LLMs. Some have argued that machine learning models are a derivative work of the training material, and that therefore the resulting models and inferred outputs must comply with the terms of any licenses on the training material. This argument has been made most forcefully by advocates of copyleft licenses (Kuhn, 2022), but has also been made for permissive licenses (Butterick, 2022). Others have taken the position that code LLM developers likely benefit from copyright law exceptions, such as fair use under U.S. copyright law, when using available code repositories for model training purposes (Rothchild & Rothchild, 2022). But even if regulations currently permit the use of public source code for training ML models, it is worth discussing whether these practices align with the ethical values of society and the broader research community. For example, one could argue that code creators should have the rights to meaningfully control whether their data is included in the training set (Jernite et al., 2022). One could also have ethical concerns around the inclusion of Personally Identifiable Information (PII) and malicious code in the training data, as code LLMs might output such sensitive data (Carlini et al., 2021; Kandpal et al., 2022) or unsafe programs (Khlaaf et al., 2022) during

deployment. We believe open datasets benefit from external scrutiny in addressing such data issues, as researchers can investigate the dataset and report issues directly to the maintainers.

In this paper, we take a step towards open and responsible research on code LLMs and release The Stack, a large dataset of permissively licensed source code. By releasing a dataset that can be shared, inspected, and used for pre-training, we hope to make the LLM development process more reproducible and transparent. This work describes how we collected the data from Github and present evidence that the dataset is a useful resource for developing competitive code LLMs. More specifically, we make the following contributions:

- We present The Stack, a large dataset with 3.1 TB of *permissively licensed* source code in 30 programming languages. We release this dataset along with a near-deduplicated version at `ANONYMIZED`.
- We train 350M decoder-only transformers on several python subsets of the data and find that removing *near-duplicates* significantly boosts performance in all experiments. We show it is possible to reproduce text2code performance of Codex (Chen et al., 2021) and CodeGen (Nijkamp et al., 2022) by only using permissively licensed data. We outperform these models by a large margin if we train on the all-license version of the dataset.
- We acknowledge that some developers do not wish their code to be used for pre-training LLMs and, therefore, start experimenting with giving developers the possibility to have their data removed from the dataset. We present the details of this opt-out process in a data governance plan in Section 3.2. We also provide further instructions for removal requests at `ANONYMIZED`.

2 Related Work

Code LLMs A growing body of research has trained large-scale transformer models on source code. Several groups have explored decoder-only models with a causal language modeling objective (Chen et al., 2021; Austin et al., 2021; Nijkamp et al., 2022; Christopoulou et al., 2022; Izadi et al., 2022; Xu et al., 2022) and generally found that larger models are increasingly capable of synthesizing programs from natural language descriptions. A few studies have used such decoder-only models for code-infilling tasks via a causal masking mechanism (Fried et al., 2022; Bavarian et al., 2022). Researchers have also investigated encoder masked language models (Feng et al., 2020; Kanade et al., 2020) and encoder-decoder architectures with various training objectives (Li et al., 2022; Ahmad et al., 2021; Wang et al., 2021; Roziere et al., 2021).

Datasets for pre-training code LLMs GitHub has been a frequently used resource for pretraining large-scale code LLMs. Google BigQuery[2] provides a snapshot of permissively licensed repositories on GitHub and can be filtered through a SQL query. AlphaCode (Li et al., 2022), BLOOM (Laurençon et al., 2022), CodeGen (Nijkamp et al., 2022), and InCoder (Fried et al., 2022) all included this resource in their pre-training dataset. A snapshot of this GitHub data is also publicly available on the HuggingFace hub[3] as the GitHub-Code dataset under the CodeParrot project. PolyCoder (Xu et al., 2022) collected a code dataset by scraping GitHub repositories and released a catalog of their downloaded repositories and files. We compare our work against these code datasets in Table 1 and elaborate more on these differences in Section 3.3.

Another frequently used resource is the CodeSearchNet corpus (Husain et al., 2019). This corpus first collected a large number of public and permissive repositories from GitHub, which were then further processed with treesitter[4] to extract pairs of functions (methods) and docstrings. The dataset contains such pairs for six programming languages: Go, Java, JavaScript, Python, PHP and Ruby. CodeBERT (Feng et al., 2020) and CodeT5 (Wang et al., 2021) used this dataset for pre-training.

Some studies have reported results by training on non-public datasets. Codex (Chen et al., 2021) collected 179 GB of python files from 54M public GitHub repositories, but did not release the data nor disclose information regarding the licensing. CodeGen collected a private GitHub dataset of 217.3 GB of permissively licensed python files. They fine-tuned models on this dataset after pre-training on the Pile (Gao et al., 2020) and

[2] `https://cloud.google.com/blog/topics/public-datasets/github-on-bigquery-analyze-all-the-open-source-code`
[3] `https://huggingface.co/datasets/codeparrot/github-code`
[4] `https://github.com/tree-sitter/tree-sitter`

the GitHub snapshot on BigQuery. While CodeGen open-sourced the model weights, they did not release the private python dataset.

Deduplication Recent studies have shown that deduplicating the training set can significantly improve the performance of LLMs. Lee et al. (2021) show that training corpora for language models contain many near-duplicates, and that LLM performance improves when long repetitive substrings are removed. Hernandez et al. (2022) also found that repeating even a small portion of the training data can significantly hurt model performance, potentially due to a fraction of its capacity being consumed by data memorization. Data duplication is even more present in code datasets, since it is common practice to reuse and clone code repositories of others. Indeed, Lopes et al. (2017) observed that a large proportion of GitHub data consists of clones, resulting in a high ratio of exact and near-duplicates. Allamanis (2019) study the effect of code duplication on machine learning models and show that it can result in highly inflated performance metrics. However, many existing code LLMs (Nijkamp et al., 2022; Xu et al., 2022; Li et al., 2022) only apply exact deduplication, which leaves a large number of near-duplicates in the training data.

3 Dataset

We describe how we collect the code dataset and create permissively licensed and near-deduplicated subsets in Section 3.1. We present a data governance plan in Section 3.2 and further analyze the data in Section 3.3.

3.1 Dataset Creation

Dataset Collection We first collected a set of active GitHub repository names from GHArchive[5], an open-source project working on archiving and releasing the public GitHub timeline. Note that these archives only contain GitHub events, such as forking a repository and commenting on an issue, and not the code repositories. We extracted unique repository names from the event archives published between January 1st, 2015 and March 31st, 2022. This resulted in a list of 220.92M unique repository names. Next, we ran a distributed compute cluster for a couple of months to clone this list of repositories. We did not successfully download all of these repositories because some of them were deleted or made private. In the end, we successfully downloaded 137.36M repositories, resulting in a clone success rate of over 62%. All repositories were downloaded between November 2021 and June 2022.

Note that we do not store binary files as we cannot use this data for pre-training. We provide the exact list of excluded binary file extensions in Appendix C. We also do not store files larger than 1 MB except if the extension is from an approved list of programming language extensions. See Appendix D for the full list. Furthermore, we avoid storing exact file duplicates by using git hashes. All repositories contain 51.76B files, but only 5.28B of them are unique (i.e., slightly more than 10%). The uncompressed size of all *stored* files is 92.36 TB.

License detection We describe how we extract license information for each repository. GHArchive provides license information when the repository owner explicitly sets the code license through the web interface. We find that licenses from GHArchive are available for 26.4M repositories. For the remaining 110.9M repositories, we run the go-license-detector[6]. This detector scans the repository and returns a list of predictions with the associated file, the SPDX license identifier[7], and a confidence score. We found that the detector did not detect a license for more than 80% of the repositories. MIT and Apache 2.0 are the most frequently detected licenses for 9.6% and 2.7% of the total repositories, respectively. We present the top-20 predicted SPDX identifiers in Table 2.

Permissive license dataset We develop a dataset of source code with only permissive licenses, i.e., with minimal restrictions on how the software can be copied, modified, and redistributed. We first provide the list of licenses which we classified as permissive in Appendix A. Note that we intentionally exclude copyleft

[5] https://gharchive.org
[6] https://github.com/src-d/go-license-detector
[7] https://spdx.org/licenses/

SPDX identifier	Number of repos (in M)	Percentage
not_found	112.51	81.91
MIT	13.16	9.58
Apache-2.0	3.72	2.71
BSD-3-Clause	0.76	0.55
error	0.58	0.42
GPL-3.0-only	0.55	0.4
GPL-3.0-or-later	0.55	0.4
deprecated_GPL-3.0+	0.55	0.4
deprecated_GPL-3.0	0.55	0.4
GPL-3.0	0.52	0.38
Unlicense	0.38	0.28
CC0-1.0	0.29	0.21
GPL-2.0-or-later	0.28	0.2
deprecated_GPL-2.0	0.28	0.2
BSD-2-Clause	0.24	0.17
CC-BY-4.0	0.2	0.15
CC-BY-3.0	0.13	0.1
GPL-2.0	0.11	0.08
MPL-2.0	0.1	0.07
AGPL-3.0	0.09	0.06

Table 2: The top 20 detected licenses in the collected repositories. We took the highest confidence prediction per repository.

licenses like GPL, as this community has strongly expressed the concern of machine learning models and inferred outputs violating the terms of their licenses Kuhn (2022).

After running all our experiments, it was brought to our attention that licenses such as MPL, LGPL, and EPL were erroneously labeled as permissive when they are in fact weak copyleft licenses. We have removed these weak copyleft license files from The Stack and will release the updated version to the community. The weak copyleft-licensed data is only a small part of the overall dataset (below 0.5% for the Python subset), hence we decided to not rerun experiments as we expect findings to remain unchanged. For the new version of The Stack, we rely on the Blue Oak Council[8] to classify the set of permissive licenses - see Appendix B for the full list and the updated programming language statistics.

One challenge in compiling the permissive license dataset is that each file might be part of multiple repositories. We opt to include a file in the permissive license dataset if at least one the repositories containing the file has a permissive license. With this procedure, we keep permissively-licensed files that were copied into non-permissively licensed repositories. However, it is possible that non-permissively licensed files are part of the dataset if a developer erroneously copied a non-permissively licensed file into a permissively licensed repository. We encourage users of the dataset to report files that might have been misclassified as permissively licensed.

We mark an individual repository as permissive by using the list of license predictions as follows. We first take the highest confidence prediction for each *separate* file that the license detector flags as a potential license. We then check whether all predictions are permissive licenses (see the list in A). If so, we classify the repository as permissive. Additionally, empty files, files larger than 1 MB, and files that could not be decoded are removed from the dataset. Finally, we point out that we give developers the opportunity to have their data removed from this dataset by following the instructions at `ANONYMIZED`.

[8] `https://blueoakcouncil.org/list`

	All-licenses		**Permissive**		**Perm. + near-dedup**	
Language	Size (GB)	Files (M)	Size (GB)	Files (M)	Size (GB)	Files (M)
Assembly	36.04	1.34	2.36	0.32	1.55	0.24
Batchfile	31.05	2.82	1.00	0.42	0.33	0.28
C	1461.23	95.57	222.88	19.88	73.21	10.95
C#	644.28	105.96	128.37	20.54	56.75	12.79
C++	1106.54	62.72	192.84	13.54	185.60	7.23
CMake	11.25	3.59	1.96	0.56	0.68	0.25
CSS	1040.53	50.47	145.33	5.73	35.60	3.54
Dockerfile	3.89	3.74	1.95	1.27	0.55	0.65
FORTRAN	26.67	1.21	3.10	0.24	1.77	0.17
GO	271.92	23.34	118.37	12.08	34.87	6.17
Haskell	15.79	2.06	6.95	0.92	3.29	0.64
HTML	9491.23	267.81	746.33	32.31	279.37	15.55
Java	1311.99	279.16	271.43	43.01	119.19	26.05
JavaScript	5820.23	209.51	486.20	39.28	220.94	25.22
Julia	21.75	0.88	3.09	0.47	1.78	0.34
Lua	88.39	5.18	6.58	0.81	3.60	0.58
Makefile	39.36	6.34	5.09	1.09	1.69	0.62
Markdown	706.8	135.69	164.61	28.97	73.09	20.91
Perl	49.21	2.74	5.50	0.55	2.16	0.31
PHP	779.66	115.53	183.19	34.18	90.21	22.65
PowerShell	13.26	1.39	3.37	0.52	1.66	0.33
Python	737.89	106.91	190.73	23.58	80.38	15.03
Ruby	78.63	30.74	23.82	6.39	22.39	4.12
Rust	78.97	6.3	40.35	3.09	13.65	1.73
Scala	28.37	6.06	14.87	2.64	6.04	1.55
Shell	71.56	14.01	8.69	3.66	3.98	2.49
SQL	1438.73	10.2	18.15	1.27	11.47	1.00
TeX	69.4	4.01	4.65	0.45	3.56	0.38
TypeScript	4145.01	75.8	131.46	19.44	120.19	12.90
Visual Basic	28.57	1.97	2.73	0.2	1.20	0.12
Total	29648.2	1633.05	3135.95	317.41	1450.75	194.79

Table 3: An overview of the amount of data we collected for 30 popular programming languages. We show the size and number of files for different splits of the data: the all-license, permissive license, and permissive license with near-deduplication.

Near-deduplication We implement near-deduplication[9] in our pre-processing pipeline on top of exact deduplication. We first split the files into words/tokens based on non-alphanumeric characters and remove files with fewer than 10 tokens. Next, we compute the MinHash (Broder, 2000; Lee et al., 2021) with 256 permutations of all documents, and use Locality Sensitive Hashing (Har-Peled et al., 2012) to find clusters of duplicates. We further reduce these clusters by ensuring that each file in the original cluster is similar to at least one other file in the reduced cluster. We consider two files similar when their Jaccard similarity exceeds 0.85. We find that in the permissive license dataset, 38.6% of the files are just near-duplicates of other files and are removed, they also represent 53.7% of the volume of the dataset. See Table 3 for a breakdown per programming language.

[9]See `ANONYMIZED`

3.2 Data Governance

One of the goals of this project is to give developers agency over their source code and let them decide whether or not it can be used to develop and evaluate LLMs, as we acknowledge that not all developers may wish to have their data used for that purpose. Our first step to that end was to select source code with permissive licenses, i.e. those with minimal restrictions on how the software can be copied, modified and redistributed (see Section 3.1). In addition, we are giving developers the ability to have their code removed from the dataset upon request. The process for submitting and enacting removal requests will keep evolving throughout the project as we receive feedback and build up more data governance tools. The following FAQ presents the current state of this process, as well as the planned next steps.

How do I know my data is in The Stack? We have a developed a tool to help users understand whether their data is in The Stack. See `ANONYMIZED`.

How can I request that my data be removed from The Stack? In order to request that data from your repositories be removed from The Stack, we ask that you first fill out the following form with your GitHub username and the email address associated with your git activity. After submitting the form, we will invite you to a private repository on our Github organisation and ask you to open an issue with the topic "remove my Github repositories from The Stack". This will verify your Github username and we will mark all public repositories under your username for removal in the next dataset release cycle. The verification process is manual at the moment but we are looking into ways to fully automate it.

What data can I request be removed from The Stack? Currently, you can request that we remove all public repositories under the provided username. In future work, we will expand the scope of data removal requests to address requests at a finer granularity (specific repositories, specific files) and to a greater range of contribution types (for example, based on whether a file or repository contains push events associated with your username according to GHArchive).

Can I also prevent my data from being included in future versions of The Stack? The removal request form will be used to validate removal requests and remove appropriate data. Validated requests and associated code pointers will also be stored in order to ensure that the code does not appear in future versions of The Stack.

What happens to my data once I have requested its removal? For as long as we are maintaining The Stack dataset, we will provide regular updates to the dataset to remove data that has been flagged since the last version. The current plan is to update the dataset every 3 months, although the schedule may change based on the volume of requests received. If we are not in a position to continue maintaining the dataset, we plan to stop distributing it in its current format and update its terms of use to limit its range of applications further, including for training new LLMs. Finally, we require that people who download the dataset agree to use the most recent allowed version in order to incorporate the removal requests.

3.3 Dataset Analysis

To gain more insight into the collected dataset, we first analyze the amount of data per programming language, then compare against other code datasets, and finally present a more extensive analysis on the python subset.

Data per programming language We present the amount of data available for 30 programming languages in Table 3. We see that the all-license dataset contains over 29 TB of data. Only selecting permissively licensed files reduces the dataset to 3.1 TB, i.e. only roughly 10% of the dataset is kept. For certain programming languages (e.g. SQL, Batchfile, TypeScript), we find that less than 4% of data has a permissive license. We might be able to increase this percentage by adding more licenses to the permissive licenses list (see Appendix A. If we further apply near-deduplication to the permissive license dataset, we end up with 1.4 TB, a reduction of more than 50%. For example, only 37% of HTML is kept when we near-duplicate

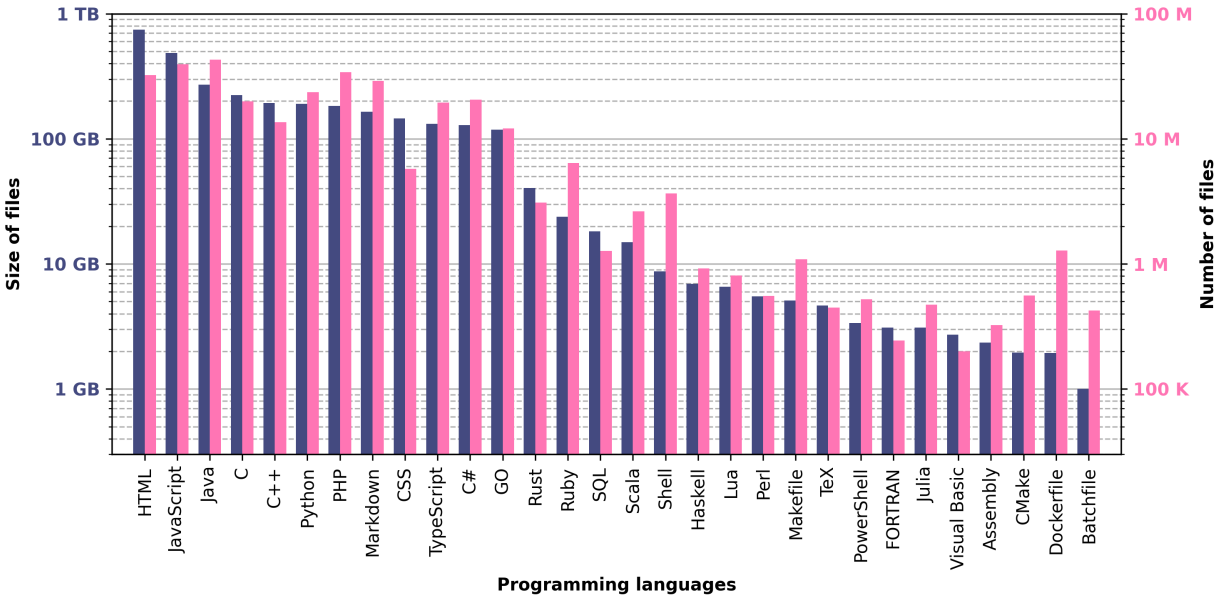

Figure 1: Histogram of the amount of data per programming language for the permissive license dataset. Note that we plot the dataset size on a log scale.

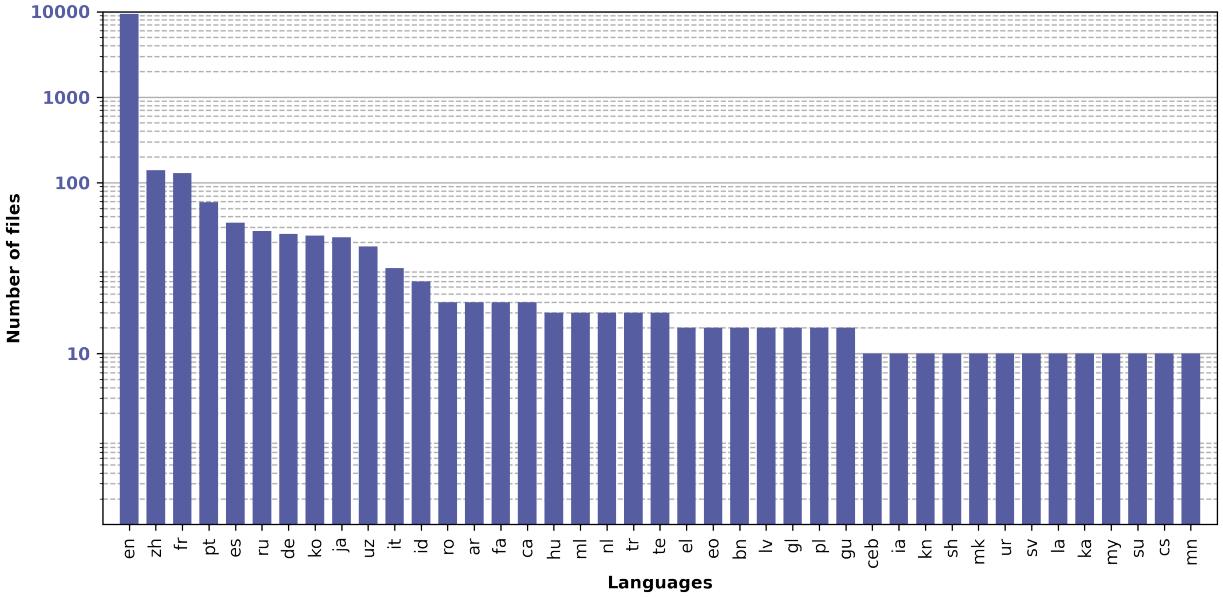

Figure 2: Histogram of natural languages in docstrings and comments from 10,000 Python files. Note that we plot the number of files on a log scale.

this data. As can be seen in Figure 1, there are a few programming languages that make up the majority of the dataset. For the permissive license dataset, the four biggest languages–HTML (746 GB), Javascript (486 GB), Java (271 GB), and C (222 GB)–consume more than 55% of the dataset size.

Comparison with other code datasets We compare The Stack with CodeParrot[10], AlphaCode(Li et al., 2022), CodeGen(Nijkamp et al., 2022), PolyCoder(Xu et al., 2022) in Table 1. Note that AlphaCode and CodeGen did not release their data but provided statistics on the amount of data per programming language. It is also worth mentioning that CodeParrot includes files with a copyleft license (e.g. GPL) while our permissive license dataset does not. PolyCoder does not filter for license information and also likely contains files with copyleft license. While The Stack and CodeParrot provide source code for 30 programming languages, AlphaCode, PolyCoder, and CodeGen only provide data for 12, 12, and 6 of the languages, respectively. Furthermore, we observe that our dataset is more than 3x the size of CodeParrot, the next largest publicly available code dataset. We also see that our dataset is bigger in size than CodeParrot for each individual programming language.

Python subset analysis First, we investigate how many configuration and test files are in the Python subset of the permissive license dataset. These config files are abundant in GitHub repositories but do not capture the complex features of source code. To detect them, we look for mentions of keywords such as "configuration file" and "test file" in the first 5 lines. If these keywords are not present, we see if the occurrence number of either *config* or *test* literals is higher than 5% of the number of lines in the file. This filter selects 15% of the files which represent 23% of the data by volume.

Next, we estimate the number of valid Python files by using the `py_compile` module on 10,000 samples from our dataset. We find that only 0.7% of the files do not compile due to syntax errors. Specifically, we find that around 0.1% of the Python files had tokenization errors (e.g., due to inconsistent use of tabs and spaces).

Finally, we analyze the docstrings and comments in the files. This data is essential for applications such as documentation generation and natural language to code translation (Chen et al., 2021; Feng et al., 2020). We analyze a subset of 10,000 files. We first use the `ast` and `tokenize` modules to extract docstrings and comments. We find that they make up 18% of the volume of the subset and that 20% of the files had little (less than 20 characters) to no natural text.

To identify the language of the extracted docstrings and comments, we use a model from the *fasttext* library[11]. We find that 94% of the files are in English. Among the other languages, Chinese and French are popular with over 100 samples. We show the full distribution of natural languages in Figure 2. It is important to note that this language detection is imperfect since docstrings often include code examples with English keywords.

4 Experiments

To better understand the quality of the collected dataset, we investigate the performance of transformer models trained from scratch on several versions of the python subset. We evaluate the models on HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) and compare against CodeGen (Nijkamp et al., 2022), InCoder (Fried et al., 2022), and Codex (Chen et al., 2021) models of similar size. We discovered late in the research process that some of the pretraining sets were contaminated with examples from the evaluation benchmarks. As running experiments is expensive and takes long, we decided to re-run only a few experiments to investigate the impact of the data contamination. In addition, we did not rerun experiments after finding out a labelling mistake in the permissive licenses (see Appendix A for full details). The results we present for permissively licensed Python data, therefore, do contain a small fraction (less than 0.5%) of weak copyleft licensed data. However, we think our main findings remain unchanged.

4.1 Experimental setup

Datasets We experiment with 4 versions of our python dataset, as well as the python subset of CodeParrot. For our datasets we either use python files from all repositories (*all-license*) or from repositories with permissive licenses (*permissive-license*). To this data we either apply near-deduplication (*near-dedup*)

[10] `https://huggingface.co/datasets/codeparrot/github-code`
[11] See `ANONYMIZED`

Dataset	Filter	HumanEval	MBPP
Python-all-license	None	338	1
	Near-dedup	291	1
Python-permissive-license	None	336	0
	Near-dedup	292	0
CodeParrot	Near-dedup	0	0

Table 4: Data contamination per training set. We report the number of files in which we find a natural language prompt of a test example.

or no further filtering (*none*). It is worth stressing that the near deduplication process is on top of the exact deduplication that we applied during the dataset collection. For CodeParrot we only experiment with the near-deduplicated version[12].

For all datasets, we follow the filtering methods of Codex (Chen et al., 2021) and remove files with:

- an average line length greater than 100 characters
- a maximum line length above 1,000 characters
- less than 25% of the characters being alphanumeric characters
- keywords in the first few lines of the file indicating that the file was likely automatically generated

Data contamination After training the models, we found that some of the training sets contained examples from the evaluation benchmarks. We detected this contamination issue by searching for exact string matches of the natural language prompts of HumanEval and MBPP. Table 4 shows how many contaminated files were found for each of the subsets. All our subsets contain HumanEval examples but only the python-all-license subsets contain MBPP examples. To investigate the impact of the data contamination, we rerun experiments on the near-deduplicated versions of the python-all-license and python-permissive-license datasets. To this end, we remove the contaminated files from these datasets. Note that we only eliminate files with exact copies of the prompts and thus do not detect paraphrases.

Evaluation We evaluate models on the HumanEval (Chen et al., 2021) and MBBP (Austin et al., 2021) benchmarks. HumanEval is a text2python benchmark containing 164 programming problems. Each problem consists of a function signature, docstring, body, and some test cases that allow to verify the correctness of the generated program. Models are evaluated with the pass@k metric: k samples are generated for each problem, a problem is solved if at least one of the samples passes the test cases, and the total fraction of problem solved is measured. In practice, we sample 200 programs for each problem, and estimate pass@k as proposed by Chen et al. (2021). We use nucleus sampling with top-$p = 0.95$, temperature $T \in \{0.2, 0.8\}$ and report the pass@k for the best temperature.

We also evaluate on MBPP (Austin et al., 2021), a text2python benchmark consisting of crowdsourced programming problems. Each problem consists of a short natural language description and 3 unit tests. We evaluate on the test set of 500 examples. While Austin et al. (2021) include all three unit tests in the prompt, Fried et al. (2022) only include a single unit test because they observed that smaller language models (i.e., with a few billion parameters) did not benefit from more unit tests. We follow Fried et al. (2022) and only add the first unit test to the natural language prompt. Besides pass@1, we also evaluate pass@10 and pass@100 by sampling 200 programs for each problem. Similarly to HumanEval, we use nucleus sampling with top-$p = 0.95$, temperature $T \in \{0.2, 0.8\}$ and report the scores for the best temperature.

Training details We experiment with decoder-only transformers trained via a causal language modeling objective. We opt for a 350M parameter model with 24 layers, a hidden dimension of 1024, 16 attention

[12] `https://hf.co/datasets/codeparrot/codeparrot-train-near-deduplication`

Dataset	Filtering	Pass@1	Pass@10	Pass@100
Codex (300M)		13.17	20.17	36.27
CodeGen (350M)		12.76	23.11	35.19
Python all-license	None	13.11	21.77	36.67
	Near-dedup	16.60	27.82	44.00
	+ Decontamination	17.34	27.64	45.52
Python permissive-license	None	10.99	15.94	27.21
	Near-dedup	13.94	22.36	37.00
	+ Decontamination	12.89	22.26	36.01
CodeParrot	Near-dedup	11.23	18.16	30.37

Table 5: HumanEval performance of a 350M model on different training sets.

heads, and a sequence length of 2048. The model is trained for 300K iterations with a global batch size of 384 using Adam (Kingma & Ba, 2015) with $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$ and a weight decay of 0.1. The learning rate set to 3×10^{-4} is warmed up for 175 steps, then follows a cosine decay. The model processes 235.9B tokens during training. The Byte-Pair Encoding tokenizer was trained on a 50-50 mixture of the Pile (Gao et al., 2020) and Python files from The Stack. We use a fork[13] of Megatron-LM (Shoeybi et al., 2019) for training.

4.2 Discussion of results

We report the HumanEval and MBPP results in Table 5 and 6, respectively.

Near-deduplication improves performance Applying near-deduplication to the training data yields a dramatic improvement, on both the all-license and permissive-license datasets. On the permissive-license dataset, near-deduplication improves HumanEval performance from 27.21% to 37.00% pass@100 and MBPP performance from 44.99% to 54.69% pass@100. We see a similar boost in results for the all-license dataset, where HumanEval performance improves from 36.67% to 44.00% pass@100 and MBPP performance from 53.59% to 61.00% pass@100.

Reproducing performance with permissive licenses We are able to reproduce text2python results of previous work with only permissively licensed source code. On HumanEval, we observe that, without near-deduplication, the performance of the permissive license dataset (27.21% pass@100) is significantly below Codex (36.27% pass@100) and CodeGen(35.19% pass@100). However, we match Codex and CodeGen performance after applying near-deduplication (37.00% pass@100). On MBPP, we observe similar findings. Without near-deduplication, the performance of the python-permissive dataset (44.99% pass@100) is significantly below CodeGen (51.80% pass@100). However, the near-deduplicated version (54.69% pass@100) surpasses the CodeGen results.

Comparison to CodeParrot We see that training on CodeParrot—another released code dataset—achieves 30.37% pass@100 on HumanEval, which is significantly below the performance of our released dataset (37.00% pass@100). On MBPP, CodeParrot also underperforms our released dataset (45.44% vs 54.69% pass@100). CodeParrot consists of data extracted from BigQuery and is not enough to obtain a competitive model on HumanEval and MBPP.

Impact of data contamination Surprisingly, we find that removing contaminated files has very little impact on the text2python results. On HumanEval, there is very small drop for the permissive-license dataset (37.00% vs 36.01% pass@100), while there is a small gain for the all-license dataset (44.00% vs 45.52% pass@100). We also see a small impact for the all-license dataset on MBPP (61.00% vs 58.28%

[13] ANONYMIZED

Model	Filtering	Pass@1	Pass@10	Pass@100
InCoder-1B		9.36	23.37	45.80
CodeGen (350M)		14.09	30.07	51.80
Python all-license	None	17.41	33.09	53.59
	Near-dedup	22.99	39.62	61.00
	+ Decontamination	21.82	37.55	58.28
Python permissive-license	None	11.60	23.13	44.99
	Near-dedup	15.94	31.70	54.69
CodeParrot	Near-dedup	6.31	21.50	45.44

Table 6: MBPP performance of a 350M model on different training sets.

pass@100). We speculate that the impact of data contamination was minimal because (1) small models are unlikely to memorize training data and (2) there were few contaminated files.

5 Conclusion and Future Work

We introduce The Stack, a large dataset of more than 3 TB of permissively licensed source code. This paper described the details of the dataset collection, presented a brief dataset analysis, and showed promising results on the HumanEval benchmark. Our experimental results show that near-deduplication is an important pre-processing step for achieving competitive results on text2code benchmarks. We release all permissively licensed files for 30 common programming languages, along with a near-deduplicated version. In future work, we would like to further improve the released dataset. We are open to releasing data of other programming languages, plan to work on methods for removing PII and malicious code, and start experimenting with giving developers the possibility to have their data removed from the dataset. We hope The Stack will be a useful resource for open and responsible research on Code LLMs.

6 Limitations

Social Impact of the Dataset The Stack is an output of ANONYMIZED. The project aims to be responsible by design and by default. The project is conducted in the spirit of Open Science, focused on the responsible development of LLMs for code.

With the release of The Stack, we aim to increase access, reproducibility, and transparency of code LLMs in the research community. Work to de-risk and improve on the implementation of ethical best practices of code LLMs is conducted in the Legal, Ethics, and Governance working group. This group has explored topics such as licensing (including copyleft and the intended use of permissively licensed code), attribution of generated code to original code, rights to restrict processing, the inclusion of Personally Identifiable Information (PII), and risks of malicious code, among other topics. This work is ongoing as of October 25th, 2022.

We expect code LLMs to enable people from diverse backgrounds to write higher quality code and develop low-code applications. Mission-critical software could become easier to maintain as professional developers are guided by code-generating systems on how to write more robust and efficient code. While the social impact is intended to be positive, the increased accessibility of code LLMs comes with certain risks such as over-reliance on the generated code and long-term effects on the software development job market.

We refer the reader to Section 7 of Chen et al. (2021) for a broader impact analysis of Code LLMs, as well as Khlaaf et al. (2022) for an in-depth risk assessment and hazard analysis of this emerging technology.

Biases of the Dataset The code collected from GitHub does not contain demographic information or proxy information about the demographics. However, it is not without risks, as the comments within the code may contain harmful or offensive language, which could be learned from the models.

Widely adopted programming languages like C and Javascript are overrepresented compared to niche programming languages like Julia and Scala. We found that some programming languages such as SQL, Batchfile, TypeScript are less likely to be permissively licensed (4% vs the average 10%). This may result in a biased representation of those languages. Permissively licensed files also tend to be longer.

We also found that the English language is over represented in the docstrings and comments, as it makes up 96% of the data for Python files.

HTML not WCAG-compliant One of the current limitations of The Stack is that scraped HTML for websites may not be compliant with Web Content Accessibility Guidelines (WCAG). This could have an impact on HTML-generated code that may introduce web accessibility issues.

Personally Identifiable Information We noted that Personally Identifiable Information (PII) such as names and email addresses are contained in the data. This PII is already exposed to the public on GitHub, however, we do plan to remove PII in future work.

Malicious code In the context of cyber-security, there is risk that large language models trained on datasets containing ransomware, malware, or other malicious code, could be used to build harmful applications. The ability to spawn new harmful applications by prompting the model using known indicators of compromise (IOCs) could reduce the experience needed for hackers to learn these techniques[14] and increase the risk of Ransomware-as-a-Service kits being developed and distributed, not on the Dark Web, but in the general public domain by citizen-hackers and activists. While more advantaged companies and communities may have resources to manage these new threats, the negative social impact of ransomware on disadvantaged communities and society in general, along with the potential legal consequences for companies that pay ransom to bad actors, is something that requires more consideration.

A handful of repositories were removed because they triggered an internal security tool during the downloading phase. However, we did not fully scan the dataset for malware and, as such, warn future users of the

[14]See this article `https://www.trendmicro.com/en_us/research/22/a/codex-exposed-helping-hackers-in-training.html`.

potential for malicious code bases in The Stack. Please report your findings of malicious code to ANONYMIZED so that it can be removed in a future release.

Data licensing While we did our best to filter for permissively licensed source code, it is possible that the license detector incorrectly classified a number of repositories. Please reach out to ANONYMIZED in case you encounter files that do not have a permissive license. We also stress that The Stack is a compilation of source files—including attribution and license notices—in the form of a dataset. Each source file in The Stack carries its own permissive license which should be respected by users of the dataset.

Model limitations We show promising text2code results for smaller models on the python dataset. More research is necessary to find out whether strong results can be obtained for larger models and other programming languages than Python.

References

Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. Unified pre-training for program understanding and generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2655–2668, Online, June 2021. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2021.naacl-main.211.

Miltiadis Allamanis. The adverse effects of code duplication in machine learning models of code. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2019, pp. 143–153, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450369954. doi: 10.1145/3359591.3359735. URL https://doi.org/10.1145/3359591.3359735.

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

Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. Efficient training of language models to fill in the middle, 2022. URL https://arxiv.org/abs/2207.14255.

Emily M. Bender and Batya Friedman. Data statements for natural language processing: Toward mitigating system bias and enabling better science. *Transactions of the Association for Computational Linguistics*, 6:587–604, 2018. doi: 10.1162/tacl_a_00041. URL https://aclanthology.org/Q18-1041.

BigScience Workshop. BLOOM (revision 4ab0472), 2022. URL https://huggingface.co/bigscience/bloom.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen Creel, Jared Quincy Davis, Dorottya Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, and et al. On the opportunities and risks of foundation models. *CoRR*, abs/2108.07258, 2021. URL https://arxiv.org/abs/2108.07258.

Andrei Z Broder. Identifying and filtering near-duplicate documents. In *Annual symposium on combinatorial pattern matching*, pp. 1–10. Springer, 2000.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Matthew Butterick. This CoPilot is stupid and wants to kill me. https://matthewbutterick.com/chron/this-copilot-is-stupid-and-wants-to-kill-me.html, 2022.

Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, Alina Oprea, and Colin Raffel. Extracting training data from large language models. In *USENIX Security Symposium*, 2021. URL https://arxiv.org/abs/2012.07805.

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. Evaluating large language models trained on code. 2021.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *CoRR*, abs/2204.02311, 2022. doi: 10.48550/arXiv.2204.02311. URL https://doi.org/10.48550/arXiv.2204.02311.

Fenia Christopoulou, Gerasimos Lampouras, Milan Gritta, Guchun Zhang, Yinpeng Guo, Zhongqi Li, Qi Zhang, Meng Xiao, Bo Shen, Lin Li, Hao Yu, Li Yan, Pingyi Zhou, Xin Wang, Yuchi Ma, Ignacio Iacobacci, Yasheng Wang, Guangtai Liang, Jiansheng Wei, Xin Jiang, Qianxiang Wang, and Qun Liu. Pangu-coder: Program synthesis with function-level language modeling, 2022. URL https://arxiv.org/abs/2207.11280.

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

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen-tau Yih, Luke Zettlemoyer, and Mike Lewis. Incoder: A generative model for code infilling and synthesis, 2022. URL https://arxiv.org/abs/2204.05999.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800GB dataset of diverse text for language modeling, 2020.

Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé Iii, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12): 86–92, 2021.

Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. 2012.

Danny Hernandez, Tom Brown, Tom Conerly, Nova DasSarma, Dawn Drain, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Tom Henighan, Tristan Hume, et al. Scaling laws and interpretability of learning from repeated data. *arXiv preprint arXiv:2205.10487*, 2022.

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

Maliheh Izadi, Roberta Gismondi, and Georgios Gousios. Codefill: Multi-token code completion by jointly learning from structure and naming sequences. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, pp. 401–412, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392211. doi: 10.1145/3510003.3510172. URL https://doi.org/10.1145/3510003.3510172.

Yacine Jernite, Huu Nguyen, Stella Biderman, Anna Rogers, Maraim Masoud, Valentin Danchev, Samson Tan, Alexandra Sasha Luccioni, Nishant Subramani, Isaac Johnson, Gerard Dupont, Jesse Dodge, Kyle Lo, Zeerak Talat, Dragomir Radev, Aaron Gokaslan, Somaieh Nikpoor, Peter Henderson, Rishi Bommasani, and Margaret Mitchell. Data governance in the age of large-scale data-driven language technology. In *2022 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '22, pp. 2206–2222, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393522. doi: 10.1145/3531146.3534637. URL https://doi.org/10.1145/3531146.3534637.

Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. Learning and evaluating contextual embedding of source code. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

Nikhil Kandpal, Eric Wallace, and Colin Raffel. Deduplicating training data mitigates privacy risks in language models. *arXiv preprint arXiv:2202.06539*, 2022.

Heidy Khlaaf, Pamela Mishkin, Joshua Achiam, Gretchen Krueger, and Miles Brundage. A hazard analysis framework for code synthesis large language models. *arXiv preprint arXiv:2207.14157*, 2022.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Bradley M. Kuhn. If Software is My Copilot, Who Programmed My Software? https://sfconservancy.org/blog/2022/feb/03/github-copilot-copyleft-gpl/, 2022.

Hugo Laurençon, Lucile Saulnier, Thomas Wang, Christopher Akiki, Albert Villanova del Moral, Teven Le Scao, Leandro Von Werra, Chenghao Mou, Eduardo González Ponferrada, Huu Nguyen, Jörg Frohberg, Mario Šaško, Quentin Lhoest, Angelina McMillan-Major, Gérard Dupont, Stella Biderman, Anna Rogers, Loubna Ben allal, Francesco De Toni, Giada Pistilli, Olivier Nguyen, Somaieh Nikpoor, Maraim Masoud, Pierre Colombo, Javier de la Rosa, Paulo Villegas, Tristan Thrush, Shayne Longpre, Sebastian Nagel, Leon Weber, Manuel Romero Muñoz, Jian Zhu, Daniel Van Strien, Zaid Alyafeai, Khalid Almubarak, Vu Minh Chien, Itziar Gonzalez-Dios, Aitor Soroa, Kyle Lo, Manan Dey, Pedro Ortiz Suarez, Aaron Gokaslan, Shamik Bose, David Ifeoluwa Adelani, Long Phan, Hieu Tran, Ian Yu, Suhas Pai, Jenny Chim, Violette Lepercq, Suzana Ilic, Margaret Mitchell, Sasha Luccioni, and Yacine Jernite. The bigscience ROOTS corpus: A 1.6TB composite multilingual dataset. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL https://openreview.net/forum?id=UoEw6KigkUn.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. Deduplicating training data makes language models better. *arXiv preprint arXiv:2107.06499*, 2021.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with alphacode. *arXiv preprint arXiv:2203.07814*, 2022.

Cristina V Lopes, Petr Maj, Pedro Martins, Vaibhav Saini, Di Yang, Jakub Zitny, Hitesh Sajnani, and Jan Vitek. Déjàvu: a map of code duplicates on github. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):1–28, 2017.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. A conversational paradigm for program synthesis. *arXiv preprint*, 2022.

John A. Rothchild and Daniel Rothchild. Copyright Implications of the Use of Code Repositories to Train a Machine Learning Model. `https://www.fsf.org/licensing/copilot/copyright-implications-of-the-use-of-code-repositories-to-train-a-machine-learning-model`, 2022.

Baptiste Roziere, Marie-Anne Lachaux, Marc Szafraniec, and Guillaume Lample. Dobf: A deobfuscation pre-training objective for programming languages. *arXiv preprint arXiv:2102.07492*, 2021.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 8696–8708, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.685. URL `https://aclanthology.org/2021.emnlp-main.685`.

Frank F. Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, MAPS 2022, pp. 1–10, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392730. doi: 10.1145/3520312.3534862. URL `https://doi.org/10.1145/3520312.3534862`.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022.