The Stack:
3 TB of permissively licensed source code

Denis Kocetkov∗
ServiceNow Research

Raymond Li∗
ServiceNow Research

Loubna Ben Allal∗
Hugging Face

Jia Li
Independent Researcher

Chenghao Mou
Independent Researcher

Carlos Muñoz Ferrandis
Hugging Face

Yacine Jernite
Hugging Face

Margaret Mitchell
Hugging Face

Sean Hughes
ServiceNow

Thomas Wolf
Hugging Face

Dzmitry Bahdanau
ServiceNow Research

Leandro von Werra‡
Hugging Face

Harm de Vries‡∗
ServiceNow Research

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 dif-

*Corresponding authors (denoted by ‡) can be contacted at contact@bigcode-project.org
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 https://hf.co/BigCode, provide a tool called 'Am I in The Stack' (https://hf.co/spaces/bigcode/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 https://www.bigcode-project.org/docs/about/the-stack/.

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. 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.

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

---

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

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 [https://hf.co/BigCode](https://hf.co/BigCode).

• 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 https://www.bigcode-project.org/docs/about/the-stack/.

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. CodeNet (Puri et al., 2021) is another large code dataset collected from online programming contest websites. It consists of 1M code snippets and is therefore significantly smaller than The Stack (317M permissively licensed files).

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

Evaluation of code LLMs Code LLMs are frequently evaluated on text-to-code benchmarks, in which models are tasked with completing code snippets from natural language descriptions (Austin et al., 2021; Hendrycks et al., 2021; Chen et al., 2021; Lai et al., 2022). These benchmarks usually measure the execution accuracy of generated code against a number of unit tests. Two popular text-to-code benchmarks are HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021). Recently, these benchmarks have been extended from Python to other programming languages (Athiwaratkun et al., 2022; Cassano et al., 2022). Code LLMs have also been used to solve a variety of other tasks. CodeXGLUE (Lu et al., 2021) is a set of 14 datasets for evaluating code generation models. The tasks include code-to-code tasks like clone detection, code repair, and code translation; text-to-code tasks like code search and code generation; and code-to-text tasks like generating documentation. CoSet (Wang & Christodorescu, 2019) introduced a dataset of 84,165 programs for semantic evaluation of neural program embeddings.

2https://cloud.google.com/blog/topics/public-datasets/github-on-bigquery-analyze-all-the-open-source-code
3https://huggingface.co/datasets/codeparrot/github-code
4https://github.com/tree-sitter/tree-sitter
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, 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. This detector scans the repository and returns a list of predictions with the associated file, the SPDX license identifier 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 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).

https://gharchive.org
https://github.com/src-d/go-license-detector
https://spdx.org/licenses/
Table 2: The top 20 detected licenses in the collected repositories. We took the highest confidence prediction per repository.

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

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 [^3] 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 [^4]). 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 [https://www.bigcode-project.org/docs/about/the-stack/](https://www.bigcode-project.org/docs/about/the-stack/).

Near-deduplication We apply near-deduplication[^9] in our pre-processing pipeline on top of exact deduplication. We largely follow the implementation[^10] of Allamanis (2019) and rely on a MinHash approach[^11].

[^8]: https://blueoakcouncil.org/list
[^9]: See https://github.com/bigcode-project/bigcode-analysis/tree/main/data_analysis
[^10]: https://gist.github.com/mallamanis/ce1a3626b5d1a9ec9b6966e6b7191dcd
[^11]: https://gist.github.com/mallamanis/ce1a3626b5d1a9ec9b6966e6b7191dcd
<table>
<thead>
<tr>
<th>Language</th>
<th>All-licenses</th>
<th>Permissive</th>
<th>Perm. + near-dedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size (GB)</td>
<td>Files (M)</td>
<td>Size (GB)</td>
</tr>
<tr>
<td>Assembly</td>
<td>36.04</td>
<td>1.34</td>
<td>2.36</td>
</tr>
<tr>
<td>Batchfile</td>
<td>31.05</td>
<td>2.82</td>
<td>1.00</td>
</tr>
<tr>
<td>C</td>
<td>1461.23</td>
<td>95.57</td>
<td>222.88</td>
</tr>
<tr>
<td>C#</td>
<td>644.28</td>
<td>105.96</td>
<td>128.37</td>
</tr>
<tr>
<td>C++</td>
<td>1106.54</td>
<td>62.72</td>
<td>192.84</td>
</tr>
<tr>
<td>CMake</td>
<td>11.25</td>
<td>3.59</td>
<td>1.96</td>
</tr>
<tr>
<td>CSS</td>
<td>1040.53</td>
<td>50.47</td>
<td>145.33</td>
</tr>
<tr>
<td>Dockerfile</td>
<td>3.89</td>
<td>3.74</td>
<td>1.95</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>26.67</td>
<td>1.21</td>
<td>3.10</td>
</tr>
<tr>
<td>GO</td>
<td>271.92</td>
<td>23.34</td>
<td>118.37</td>
</tr>
<tr>
<td>Haskell</td>
<td>15.79</td>
<td>2.06</td>
<td>6.95</td>
</tr>
<tr>
<td>HTML</td>
<td>9491.23</td>
<td>267.81</td>
<td>746.33</td>
</tr>
<tr>
<td>Java</td>
<td>1311.99</td>
<td>279.16</td>
<td>271.43</td>
</tr>
<tr>
<td>JavaScript</td>
<td>5820.23</td>
<td>209.51</td>
<td>486.20</td>
</tr>
<tr>
<td>Julia</td>
<td>21.75</td>
<td>0.88</td>
<td>3.09</td>
</tr>
<tr>
<td>Lua</td>
<td>88.39</td>
<td>5.18</td>
<td>6.58</td>
</tr>
<tr>
<td>Makefile</td>
<td>39.36</td>
<td>6.34</td>
<td>5.09</td>
</tr>
<tr>
<td>Markdown</td>
<td>706.8</td>
<td>135.69</td>
<td>164.61</td>
</tr>
<tr>
<td>Perl</td>
<td>49.21</td>
<td>2.74</td>
<td>5.50</td>
</tr>
<tr>
<td>PHP</td>
<td>779.66</td>
<td>115.53</td>
<td>183.19</td>
</tr>
<tr>
<td>PowerShell</td>
<td>13.26</td>
<td>1.39</td>
<td>3.37</td>
</tr>
<tr>
<td>Python</td>
<td>737.89</td>
<td>106.91</td>
<td>190.73</td>
</tr>
<tr>
<td>Ruby</td>
<td>78.63</td>
<td>30.74</td>
<td>23.82</td>
</tr>
<tr>
<td>Rust</td>
<td>78.97</td>
<td>6.3</td>
<td>40.35</td>
</tr>
<tr>
<td>Scala</td>
<td>28.37</td>
<td>6.06</td>
<td>14.87</td>
</tr>
<tr>
<td>Shell</td>
<td>71.56</td>
<td>14.01</td>
<td>8.69</td>
</tr>
<tr>
<td>SQL</td>
<td>1438.73</td>
<td>10.2</td>
<td>18.15</td>
</tr>
<tr>
<td>TeX</td>
<td>69.4</td>
<td>4.01</td>
<td>4.65</td>
</tr>
<tr>
<td>TypeScript</td>
<td>4145.01</td>
<td>75.8</td>
<td>131.46</td>
</tr>
<tr>
<td>Visual Basic</td>
<td>28.57</td>
<td>1.97</td>
<td>2.73</td>
</tr>
</tbody>
</table>

| Total          | 29648.2      | 1633.05    | 3135.95   | 317.41    | 19.4%  | 1450.75      | 194.79    | 61.4% |

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. The “Perc.” columns show the percentage of files kept after the permissive license selection and the removal of near-duplicates, respectively.

\cite{2000}: Lee et al. (2021) to find near-duplicates. We first split the files into words/unigrams based on non-alphanumeric characters and remove files with fewer than 10 tokens. Next, we compute the MinHash with 256 permutations of all documents, and use Locality Sensitive Hashing (Har-Peled et al., 2012) to find clusters of duplicates. We include another false positive check by ensuring that each file in the original cluster is similar to (Jaccard similarity exceeding 0.85) at least one other file. We find that in the permissive license dataset, 38.6% of the files are 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. We find that CMake, C, C++, and HTML subsets contain many near-duplicates (45-55%) whereas SQL, Tex, and Assembly contain relatively few near-duplicates (15-25%).

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 developed a tool to help users understand whether their data is in The Stack. See [https://hf.co/spaces/bigcode/in-the-stack](https://hf.co/spaces/bigcode/in-the-stack).

**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 the BigCode organization 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 1.6B files (29.6 TB data). Only selecting permissively licensed files reduces the dataset to 317M files (3.1 TB), i.e. only roughly 19.4% of the files are kept. For certain programming languages (e.g. CSS, Tex, Visual Basic), we find that less than 12% of the files have a permissive license. We might be able to increase this percentage by adding more licenses to the permissive license list (see Appendix A. If we further apply near-deduplication to the permissive license dataset, we end up with 194M files (1.4 TB), a reduction of almost 40%. For example, more than half of the files are removed for CMake and HTML. Another important observation is that the majority of the dataset are coming from a few programming languages, as can be seen in Figure [I]. 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 volume.
Comparison with other code datasets  We compare The Stack with CodeParrot\footnote{https://huggingface.co/datasets/codeparrot/github-code}, AlphaCode\cite{Li2022}, CodeGen\cite{Nijkamp2022}, PolyCoder\cite{Xu2022} 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 library12. 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
In this work, we investigate the quality of the Python subset of The Stack. Subsequent work by the BigCode community has investigated the quality of other subsets by training larger Code LLMs that support more programming languages (Allal et al., 2023; Li et al., 2023). Here, we train several LLMs on the Python subset of The Stack and evaluate the models on HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021). We 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) 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 version13.

12 See https://github.com/bigcode-project/bigcode-analysis/tree/main/data_analysis
13 https://hf.co/datasets/codeparrot/codeparrot-train-near-deduplication
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.

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 MBPP (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 (Chen et al., 2021): k model samples are generated for each problem, and a problem is considered solved if at least one of the samples passes all the unit tests. We report the average fraction of problems solved. In practice, we sample 200 programs for each problem, and estimate pass@k with the unbiased estimated 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 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 $3 \times 10^{-4}$ 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
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Filtering</th>
<th>Pass@1</th>
<th>Pass@10</th>
<th>Pass@100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Codex (300M)</td>
<td></td>
<td>13.17</td>
<td>20.17</td>
<td>36.27</td>
</tr>
<tr>
<td>CodeGen (350M)</td>
<td></td>
<td>12.76</td>
<td>23.11</td>
<td>35.19</td>
</tr>
<tr>
<td>Python all-license</td>
<td>None</td>
<td>13.11</td>
<td>21.77</td>
<td>36.67</td>
</tr>
<tr>
<td></td>
<td>Near-dedup</td>
<td>16.60</td>
<td>27.82</td>
<td>44.00</td>
</tr>
<tr>
<td></td>
<td>+ Decontamination</td>
<td>17.34</td>
<td>27.64</td>
<td>45.52</td>
</tr>
<tr>
<td>Python permissive-license</td>
<td>None</td>
<td>10.99</td>
<td>15.94</td>
<td>27.21</td>
</tr>
<tr>
<td></td>
<td>Near-dedup</td>
<td>13.94</td>
<td>22.36</td>
<td>37.00</td>
</tr>
<tr>
<td></td>
<td>+ Decontamination</td>
<td>12.89</td>
<td>22.26</td>
<td>36.01</td>
</tr>
<tr>
<td>CodeParrot</td>
<td>Near-dedup</td>
<td>11.23</td>
<td>18.16</td>
<td>30.37</td>
</tr>
</tbody>
</table>

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

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

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

...Pile (Gao et al., 2020) and Python files from The Stack. We use a fork of Megatron-LM 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% 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.

**Acknowledgement** We thank Christopher Akiki, Evgenii Zholtonozhskii, Sebastien Paquet, Torsten Scholak, and Luis Villa for their feedback on an early draft of this paper. We also thank Fanny Rancourt and Masoud Hashemi for help with the data cards. Lastly, we are grateful to ServiceNow and Hugging Face for the provided compute resources.
6 Limitations

Social Impact of the Dataset  The Stack is an output of the BigCode Project\cite{bigcode_project}. BigCode 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 various BigCode working groups. The Legal, Ethics, and Governance working 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 \cite{chen2021impact} for a broader impact analysis of Code LLMs, as well as \cite{khalaf2021exposing} 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, Batch-file, 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\cite{trendmicro} 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.

\begin{thebibliography}{10}
\bibitem{bigcode_project} https://www.bigcode-project.org
\bibitem{trendmicro} See this article https://www.trendmicro.com/en_us/research/22/a/codex-exposed-helping-hackers-in-training.html
\end{thebibliography}
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 potential for malicious code bases in The Stack. Please report your findings of malicious code to contact@bigcode-project.org 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 contact@bigcode-project.org 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


A Permissive Licenses

For the experiments presented in the paper, we use the following SPDX identifiers for our permissive license dataset:

- MIT-0
- MIT
- MIT-feh
- Apache-2.0
- BSD-3-Clause
After running all 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 and will release the updated version of The Stack. The weak copyleft-licensed data is only a small part of the overall dataset (below 0.5% for the Python subset), hence we expect the experimental findings of the paper to remain unchanged. In the next section, we describe how we updated The Stack.

## B The Stack v1.1

For the Stack v1.1, we rely on the Blue Oak Council\[17\] to classify the licenses. The new classification process results in 193 permissive licenses (which we list, for completeness, at the end of this section). The Stack v1.1 also includes more programming languages. We used the following list of programming language extensions https://gist.github.com/ppisarczyk/43962d06686722d26d176fad46879d41 and obtained data for 370 programming languages. We show the updated statistics for the 30 popular programming languages in Table 7. In general, we find that the amount of data increased for most programming languages. Specifically, we see a large increase in data for C# (128.37 GB vs 215.07 GB), Tex (4.65 GB vs 8.19 GB), and Markdown (164.61 GB vs 245.26 GB). On the other hand, we see a minor decrease in data for Go (118.37 GB vs 112.86 GB) and Rust (40.35 GB vs 39.85 GB). We release this updated version to the research community.

- MIT
- Apache-2.0
- BSD-3-Clause
- Unlicense
- CC0-1.0
- BSD-2-Clause

---

17 https://blueoakcouncil.org/list
<table>
<thead>
<tr>
<th>Language</th>
<th>All-licenses</th>
<th>Permissive</th>
<th>Perm. + near-dedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size (GB)</td>
<td>Files (M)</td>
<td>Size (GB)</td>
</tr>
<tr>
<td>Assembly</td>
<td>36.04</td>
<td>1.34</td>
<td>2.57</td>
</tr>
<tr>
<td>Batchfile</td>
<td>31.05</td>
<td>2.82</td>
<td>1.06</td>
</tr>
<tr>
<td>C</td>
<td>1461.23</td>
<td>95.57</td>
<td>255.29</td>
</tr>
<tr>
<td>C++</td>
<td>644.28</td>
<td>105.96</td>
<td>215.07</td>
</tr>
<tr>
<td>C#</td>
<td>1106.54</td>
<td>62.72</td>
<td>133.55</td>
</tr>
<tr>
<td>CMake</td>
<td>11.25</td>
<td>3.59</td>
<td>2.1</td>
</tr>
<tr>
<td>CSS</td>
<td>1040.53</td>
<td>50.47</td>
<td>150.2</td>
</tr>
<tr>
<td>Dockerfile</td>
<td>3.89</td>
<td>3.74</td>
<td>1.9</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>26.67</td>
<td>1.21</td>
<td>3.81</td>
</tr>
<tr>
<td>GO</td>
<td>271.92</td>
<td>23.34</td>
<td>112.86</td>
</tr>
<tr>
<td>Haskell</td>
<td>15.79</td>
<td>2.06</td>
<td>5.85</td>
</tr>
<tr>
<td>HTML</td>
<td>9491.23</td>
<td>267.81</td>
<td>812.73</td>
</tr>
<tr>
<td>Java</td>
<td>1311.99</td>
<td>279.16</td>
<td>266.41</td>
</tr>
<tr>
<td>Javascript</td>
<td>5820.23</td>
<td>209.51</td>
<td>496.22</td>
</tr>
<tr>
<td>Julia</td>
<td>21.75</td>
<td>0.88</td>
<td>3.4</td>
</tr>
<tr>
<td>Lua</td>
<td>88.39</td>
<td>5.18</td>
<td>7.09</td>
</tr>
<tr>
<td>Makefile</td>
<td>39.36</td>
<td>6.34</td>
<td>6.88</td>
</tr>
<tr>
<td>Markdown</td>
<td>706.8</td>
<td>135.69</td>
<td>245.26</td>
</tr>
<tr>
<td>Perl</td>
<td>49.21</td>
<td>2.74</td>
<td>7.08</td>
</tr>
<tr>
<td>PHP</td>
<td>779.66</td>
<td>115.53</td>
<td>155.79</td>
</tr>
<tr>
<td>PowerShell</td>
<td>13.26</td>
<td>1.39</td>
<td>3.42</td>
</tr>
<tr>
<td>Python</td>
<td>737.89</td>
<td>106.91</td>
<td>200.93</td>
</tr>
<tr>
<td>Ruby</td>
<td>78.63</td>
<td>30.74</td>
<td>25.95</td>
</tr>
<tr>
<td>Rust</td>
<td>78.97</td>
<td>6.3</td>
<td>39.85</td>
</tr>
<tr>
<td>Scala</td>
<td>28.37</td>
<td>6.06</td>
<td>15.56</td>
</tr>
<tr>
<td>Shell</td>
<td>71.56</td>
<td>14.01</td>
<td>9.07</td>
</tr>
<tr>
<td>SQL</td>
<td>1438.73</td>
<td>10.2</td>
<td>19.94</td>
</tr>
<tr>
<td>TeX</td>
<td>69.4</td>
<td>4.01</td>
<td>8.19</td>
</tr>
<tr>
<td>Typescript</td>
<td>4145.01</td>
<td>75.8</td>
<td>131.01</td>
</tr>
<tr>
<td>Visual Basic</td>
<td>28.57</td>
<td>1.97</td>
<td>3.81</td>
</tr>
</tbody>
</table>

| Total      | 29648.2      | 1633.05    | 3372.85           | 340.31     | 1212.7    | 201.90    |

Table 7: An overview of the amount of data we collected for The Stack v1.1. 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.

- CC-BY-4.0
- CC-BY-3.0
- 0BSD
- RSA-MD
- WTFPL
- MIT-0
- ISC
- ADSL
- BSL-1.0
• Zlib
• Artistic-2.0
• FTL
• MS-PL
• BSD-2-Clause-FreeBSD
• FSFAP
• BSD-Source-Code
• Apache-1.1
• BSD-4-Clause
• Ruby
• Artistic-1.0
• MulanPSL-1.0
• BSD-1-Clause
• X11
• CNRI-Python
• Beerware
• Condor-1.1
• PostgreSQL
• CECILL-B
• Intel
• Vim
• Naumen
• OML
• BSD-3-Clause-Clear
• AML
• PHP-3.01
• OpenSSL
• PSF-2.0
• Xnet
• Linux-OpenIB
• BSD-3-Clause-LBNL
• UPL-1.0
• AFL-3.0
- BlueOak-1.0.0
- Info-ZIP
- BSD-4-Clause-UC
- AAL
- LPPL-1.3c
- bzip2-1.0.6
- W3C
- W3C-20150513
- AFL-1.1
- DOC
- ICU
- CC-BY-2.0
- curl
- MTLL
- OLDAP-2.2.1
- ECL-2.0
- Adobe-Glyph
- CNRI-Python-GPL-Compatible
- BSD-2-Clause-Patent
- IJG
- PHP-3.0
- ZPL-2.1
- MIT-advertising
- NCSA
- Fair
- BSD-3-Clause-Attribution
- OLDAP-2.3
- NLPL
- BSD-3-Clause-Open-MPI
- CIArtistic
- Python-2.0
- NASA-1.3
- TCL
• Artistic-1.0-Perl
• blessing
• BSD-3-Clause-No-Nuclear-Warranty
• ImageMagick
• Net-SNMP
• Artistic-1.0-cl8
• OLDAP-2.5
• MIT-feh
• OLDAP-2.4
• MITNFA
• AFL-2.1
• libpng-2.0
• EFL-2.0
• OLDAP-2.7
• IBM-pibs
• libtiff
• OLDAP-2.8
• Cube
• Adobe-2006
• BSD-2-Clause-NetBSD
• zlib-acknowledgement
• OLDAP-2.6
• BSD-3-Clause-No-Nuclear-License-2014
• OLDAP-1.4
• Libpng
• MIT-CMU
• AFL-2.0
• JasPer-2.0
• LPL-1.02
• Zend-2.0
• TCP-wrappers
• XFree86-1.1
• FSFUL
• OLDAP-1.3
• SGI-B-2.0
• NetCDF
• CNRI-Jython
• Zed
• ZPL-2.0
• AFL-1.2
• Apache-1.0
• CC-BY-1.0
• OLDAP-2.1
• OLDAP-1.2
• OLDAP-2.0
• NTP
• LPL-1.0
• AMPAS
• Barr
• mpich2
• ANTLR-PD
• Xerox
• Spencer-94
• AMDPLPA
• BSD-3-Clause-No-Nuclear-License
• HPND
• ECL-1.0
• MirOS
• Qhull
• ZPL-1.1
• TU-Berlin-2.0
• Spencer-86
• SMLNJ
• xinetd
• OLDAP-2.2.2
• OGTSL
• MIT-enna
• Font-exception-2.0
• FSFULLR
• TU-Berlin-1.0
• xpp
• NRL
• W3C-19980720
• EFL-1.0
• eGenix
• Unicode-DFS-2016
• SWL
• Spencer-99
• Plexus
• VSL-1.0
• Leptonica
• Unicode-DFS-2015
• Mup
• Giftware
• OLDAP-2.2
• APAFML
• NBPL-1.0
• OLDAP-1.1
• Entessa
• Multics
• Newsletr
• psutils
• bzip2-1.0.5
• Afmparse
• diffmark
• BSD-2-Clause-Views
• DSDP
• MIT-Modern-Variant
• ANTLR-PD-fallback
• Bahyph
• BSD-3-Clause-Modification
• BSD-4-Clause-Shortened
• HTM LTIDY
• MIT-open-group
• MulanPSL-2.0
• OLDAP-2.0.1
• Saxpath
• Borceux
• Crossword
• CrystalStacker
• Rdisc
• Wsuipa

C Excluded file extensions

Part of this list was taken from https://github.com/EleutherAI/github-downloader/blob/345e7c4cb9e0dc8a0615fd995a08bf9d73b3fe6/download_repo_text.py


D Included programming language extensions

This list of programming language extensions is taken from https://gist.github.com/ppisarczyk/43962d0668722d6176fadd41.
