StarCoder 2 and The Stack v2: The Next Generation

Anton Lozhkov[1] **Raymond Li**[2] **Loubna Ben Allal**[1] **Federico Cassano**[4] **Joel Lamy-Poirier**[2] **Nouamane Tazi**[1] **Ao Tang**[3] **Dmytro Pykhtar**[3] **Jiawei Liu**[7] **Yuxiang Wei**[7] **Tianyang Liu**[25] **Max Tian**[2] **Denis Kocetkov**[2] **Arthur Zucker**[1] **Younes Belkada**[1] **Zijian Wang**[5] **Qian Liu**[12] **Dmitry Abulkhanov**[5] **Indraneil Paul**[5] **Zhuang Li**[14] **Wen-Ding Li**[26] **Megan Risdal**[24] **Jia Li**[5] **Jian Zhu**[16] **Terry Yue Zhuo**[14,15] **Evgenii Zheltonozhskii**[13] **Nii Osae Osae Dade**[28] **Wenhao Yu**[20] **Lucas Krauß**[5] **Naman Jain**[27] **Yixuan Su**[30] **Xuanli He**[23] **Manan Dey**[31] **Eduardo Abati**[5] **Yekun Chai**[5] **Niklas Muennighoff**[29] **Xiangru Tang**[5] **Muhtasham Oblokulov**[18] **Christopher Akiki**[9,10] **Marc Marone**[8] **Chenghao Mou**[5] **Mayank Mishra**[19] **Alex Gu**[17] **Binyuan Hui**[5] **Tri Dao**[21] **Armel Zebaze**[1] **Olivier Dehaene**[1] **Nicolas Patry**[1] **Canwen Xu**[25] **Julian McAuley**[25] **Torsten Scholak**[2] **Sebastien Paquet**[2] **Jennifer Robinson**[6] **Carolyn Jane Anderson**[22] **Nicolas Chapados**[2] **Mostofa Patwary**[3] **Nima Tajbakhsh**[3] **Yacine Jernite**[1] **Carlos Muñoz Ferrandis**[1] **Lingming Zhang**[7] **Sean Hughes**[6] **Thomas Wolf**[1] **Arjun Guha**[4,11] **Leandro von Werra**[1,⋆] **Harm de Vries**[2,⋆]

[1]Hugging Face [2]ServiceNow Research [3]Nvidia [4]Northeastern University [5]Independent [6]ServiceNow [7]University of Illinois Urbana-Champaign [8]Johns Hopkins University [9]Leipzig University [10]ScaDS.AI [11]Roblox [12]Sea AI Lab [13]Technion – Israel Institute of Technology [14]Monash University [15]CSIRO's Data61 [16]University of British Columbia [17]MIT [18]Technical University of Munich [19]IBM Research [20]University of Notre Dame [21]Princeton University [22]Wellesley College [23]University College London [24]Kaggle [25]UC San Diego [26]Cornell University [27]UC Berkeley [28]Mazzuma [29]Contextual AI [30]Cohere [31]Salesforce

Corresponding authors (⋆) can be contacted at contact@bigcode-project.org

Abstract

The BigCode project,[1] an open-scientific collaboration focused on the responsible development of Large Language Models for Code (Code LLMs), introduces StarCoder2. In partnership with Software Heritage (SWH),[2] we build The Stack v2 on top of the digital commons of their source code archive. Alongside the SWH repositories spanning 619 programming languages, we carefully select other high-quality data sources, such as GitHub pull requests, Kaggle notebooks, and code documentation. This results in a training set that is 4× larger than the first StarCoder dataset. We train StarCoder2 models with 3B, 7B, and 15B parameters on 3.3 to 4.3 trillion tokens and thoroughly evaluate them on a comprehensive set of Code LLM benchmarks.

We find that our small model, StarCoder2-3B, outperforms other Code LLMs of similar size on most benchmarks, and also outperforms StarCoderBase-15B. Our large model, StarCoder2-15B, significantly outperforms other models of comparable size. In addition, it matches or outperforms CodeLlama-34B, a model more than twice its size. Although DeepSeekCoder-33B is the best-performing model at code completion for high-resource languages, we find that StarCoder2-15B outperforms it on math and code reasoning benchmarks, as well as several low-resource languages. We make the model weights available under an OpenRAIL license and ensure full transparency regarding the training data by releasing the SoftWare Heritage persistent IDentifiers (SWHIDs) of the source code data.

[1] https://www.bigcode-project.org
[2] https://www.softwareheritage.org/

1 Introduction

Large Language Models for Code (Code LLMs; Chen et al., 2021; Nijkamp et al., 2023; Rozière et al., 2023; Guo et al., 2024) have rapidly emerged as powerful assistants for writing and editing code. As of January 30, 2024, GitHub CoPilot has garnered over 1.3 million paying subscribers, with over 50,000 organisations opting for the enterprise version (MSFT Q2 Earning Call, 2024), estimated to increase developer productivity by up to 56% as well as developer satisfaction (Peng et al., 2023; Ziegler et al., 2024). ServiceNow recently disclosed that their "text-to-code" solution, built from fine-tuning StarCoderBase models (Li et al., 2023), results in a 52% increase in developer productivity (Yahoo Finance, 2024). Despite the initial focus on generating code snippets from natural language instructions or other code snippets, Code LLMs exhibit the potential to enhance all phases of the software development cycle (Hou et al., 2023; Fan et al., 2023; Wang et al., 2024; Zhuo et al., 2023b). This includes speeding up the implementation of new projects, improving quality assurance for developed software, helping detect and fix bugs, simplifying maintenance tasks, and easing migration to newer software.

The development process of LLMs can exhibit different levels of openness (Solaiman, 2023; Ding et al., 2022; Akiki et al., 2022). Proprietary models like OpenAI's GPT-4 (OpenAI et al., 2023) and Google's Gemini (Gemini Team et al., 2023) provide access to the model through a paid API but do not disclose development details. On the other hand, open-weight models like Code LLaMa (Rozière et al., 2023), Mistral (Jiang et al., 2023), and DeepSeekCoder (Guo et al., 2024) have released the model weights. This enables the open-source community to run these models locally, inspect the model representations, and fine-tune them on their tasks. However, the model developers have not disclosed their training data. Consequently, content creators do not know if their data was used for training, social scientists cannot scrutinize the dataset for bias and toxicity, and LLM developers lack information as to what extent the training set is contaminated with test benchmarks. More broadly, this practice hinders scientific progress as other research teams cannot readily reuse each other's training data. Other LLM development projects, like Allen AI's OLMo (Groeneveld et al., 2024), Eleuther AI's Pythia (Biderman et al., 2023), and BigScience's BLOOM (BigScience Workshop, 2022; Scao et al., 2022a), have adopted a fully open development approach by releasing training data, training frameworks, and evaluation suites.

The BigCode project was established in September 2022 as an open scientific collaboration focused on the open and responsible development of Code LLMs. BigCode is stewarded by ServiceNow and Hugging Face in the spirit of open governance (BigCode collaboration et al., 2023) and has brought together more than 1,100 members from diverse academic institutes and industry labs. The community previously released The Stack v1 (Kocetkov et al., 2023), a 6.4 TB dataset of permissively licensed source code in 384 programming languages. The Stack v1 includes a governance tool called "Am I in The Stack," designed for developers to verify if their source code is included in the dataset. It also provides an opt-out process for those who prefer to exclude their code from the dataset. In December 2022, the BigCode community released SantaCoder (Ben Allal et al., 2023), a strong-performing 1.1B parameter model trained on Java, JavaScript, and Python code from The Stack v1. Building upon this success, the community further scaled up its effort and released StarCoder on May 4th, 2023 (Li et al., 2023). At its release, the 15B parameter StarCoder model was the best open-access LLM for code.

This technical report describes the development process of The Stack v2 and StarCoder2. The Stack v2 builds upon the foundation of Software Heritage's vast source code archive, which spans over 600 programming languages. In addition to code repositories, we curate other high-quality open data sources, including Github issues, pull requests, Kaggle and Jupyter notebooks, code documentation, and other natural language datasets related to math, coding, and reasoning. To prepare the data for training, we perform deduplication, create filters to eliminate low-quality code, redact Personally Identifiable Information (PII), remove malicious code, and handle opt-outs from developers who requested to have their code removed from the dataset. With this new training set of 900B+ unique tokens, 4× larger than the first StarCoder dataset, we develop the next generation of StarCoder models. We train Code LLMs with 3B, 7B, and 15B parameters using a two-stage training process (Rozière et al., 2023; Guo et al., 2024). We start base model training with a 4k context window and subsequently fine-tune the model with a 16k context window. We ensure that the training process does not exceed more than 5 epochs over the dataset (Muennighoff et al., 2023). However, we push

the number of training tokens far beyond the compute-optimal number suggested by Chinchilla (Harm's law; de Vries, 2023) and train relatively small models within the range of 3.3 to 4.3 trillion tokens. We thoroughly assess and compare the performance of these models on a suite of code LLM benchmarks (Cassano et al., 2023b; Austin et al., 2021; Chen et al., 2021; Liu et al., 2023a; Lai et al., 2023; Muennighoff et al., 2024a; Cassano et al., 2024; Liu et al., 2023b; Ding et al., 2023; Gu et al., 2024; Cobbe et al., 2021; Pearce et al., 2022; Dhamala et al., 2021; Nozza et al., 2021; Gehman et al., 2020), finding that:

- The StarCoder2-3B model outperforms other Code LLMs of similar size (StableCode-3B and DeepSeekCoder-1.3B) on most benchmarks. Moreover, it matches or surpasses the performance of StarCoderBase-15B.
- The StarCoder2-15B model significantly outperforms other models of comparable size (CodeLlama-13B), and matches or outperforms CodeLlama-34B. DeepSeekCoder-33B is the best model at code completion benchmarks for high-resource languages. However, StarCoder2-15B matches or outperforms DeepSeekCoder-33B on low-resource programming languages (e.g., D, Julia, Lua, and Perl). Moreover, when we consider benchmarks that require models to reason about code execution (Gu et al., 2024) or mathematics (Cobbe et al., 2021), we find that StarCoder2-15B outperforms DeepSeekCoder-33B.
- The StarCoder2-7B model outperforms CodeLlama-7B but is behind DeepSeekCoder-6.7B. It is not clear to this report's authors why StarCoder2-7B does not perform as well as StarCoder2-3B and StarCoder2-15B for their size.

2 Data Sources

In this section, we elaborate on the process of obtaining training data, encompassing not just the data sourced from Software Heritage (§2.1) but also GitHub issues (§2.2), pull requests (§2.3), Jupyter and Kaggle notebooks (§2.4), documentation (§2.5), intermediate representations (§2.6), small math and coding datasets (§2.7), and other natural language datasets (§2.8).

2.1 Source Code

Software Heritage We build the Stack v2 on top of the Software Heritage (SH) archive (Abramatic et al., 2018), maintained by the non-profit organization of the same name. The mission of Software Heritage is to collect and preserve all knowledge taking the form of source code. We work with the SH graph dataset (Pietri et al., 2020), a fully deduplicated Merkle DAG (Merkle, 1987) representation of the full archive. The SH graph dataset links together file identifiers, source code directories, and git commits, up to the entire states of repositories, as observed during periodic crawls by Software Heritage.

Extracting repositories We leverage the `2023-09-06` version of the SH graph dataset as the primary source. We start by extracting the most recently crawled versions of all GitHub repositories and filtering them to retain only the main branch. The branch is considered main if the repository metadata in GHArchive lists it as the default branch or if its name is `main` or `master`. We only extract the latest revision (commit) from the main branch and deduplicate the repositories based on the unique hashes of their contents (column `directory_id` of the SH dataset). The repositories' directory structure is reconstructed by recursively joining the `directory_entry` table of the dataset to itself using the `directory_id` and `target` columns and concatenating the directory and file names (column `name`) into full paths. We only traverse the directory tree up to level 64. The individual file contents are downloaded from the SH `content` S3 bucket if the compressed file size is less than 10MB.

License detection We extract repository-level license information from GHArchive (Github Archive, 2024) for all repositories with matching names in the SWH dataset. When the repo-level license is not available, i.e., for 96.93% of repositories, we use the ScanCode Toolkit (ScanCode, 2024) to detect file-level licenses as follows:

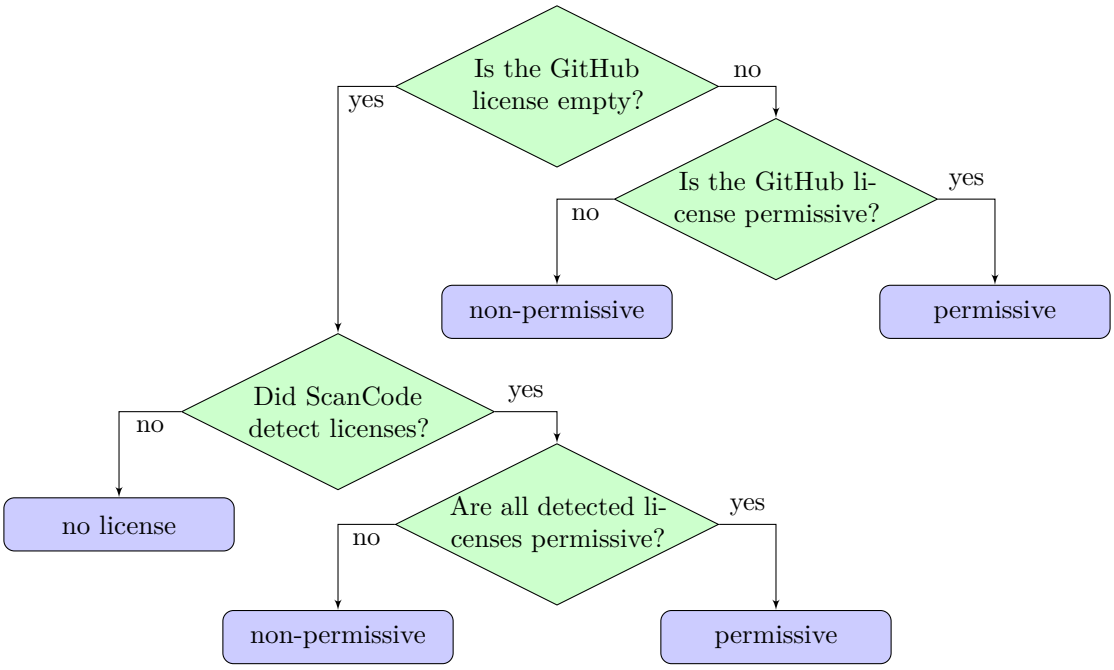

Figure 1: *File-level license assignment logic.*

- Find all files that could contain a license using a regular expression in Appendix A.3. This allows us to gather files that either explicitly contain a license (e.g., `LICENSE`, `MIT.txt`, `Apache2.0`) or contain a reference to the license (e.g., `README.md`, `GUIDELINES`);
- Apply ScanCode's license detection to the matching files and gather the SPDX[3] IDs of the detected licenses;
- Propagate the detected licenses to all files that have the same base path within the repository as the license file.

Once the file-level license information is gathered, we decide whether the file is permissively licensed, non-permissively licensed, or unlicensed, following the algorithm described in Figure 1.

The licenses we consider permissive are listed in Appendix A.4. This list was compiled from the licenses approved by the Blue Oak Council (Blue Oak Council, 2024), as well as licenses categorized as "Permissive" or "Public Domain" by ScanCode (ScanCode License Categories, 2024).

Data licenses We consider three types of files: permissively licensed, non-permissively licensed (e.g., copyleft), and unlicensed files. The main difference between the Stack v2 and the Stack v1 is that we include both permissively licensed and unlicensed files. We exclude commercial licenses since their creators do not intend their code to be used for commercial purposes. We also exclude copyleft-licensed code due to uncertainty regarding the community's stance on using such data for LLM training and its relatively low volume.

Language detection While the Stack v1 (Kocetkov et al., 2023) detects programming languages by their file extension, we instead rely on a language classifier. Specifically, we use `go-enry` based on GitHub's library `linguist` (go-enry, 2024) to detect the programming language for each file. We detect 658 unique languages in `TheStackV2-dedup`, some of which get removed at the data inspection stage (see next paragraph).

[3]System Package Data Exchange, https://spdx.dev.

Table 1: *A comparison of The Stack v1 and v2 on 32 popular programming languages. We show the size and number of files for different data splits: The Stack v1 deduped, The Stack v2 deduped, and the training data used for StarCoder2-15B.*

	The-stack-v1-dedup		**The-stack-v2-dedup**		**The-stack-v2-swh-full**	
Language	Size (GB)	Files (M)	Size (GB)	Files (M)	Size (GB)	Files (M)
Assembly	1.58	0.25	13.02	0.77	7.74	0.70
Batchfile	0.29	0.25	2.11	1.13	1.02	0.99
C	57.43	8.53	202.05	20.78	114.92	19.18
C#	46.29	10.84	239.89	51.23	169.75	48.49
C++	50.89	6.37	353.89	43.18	211.33	42.23
CMake	0.45	0.19	2.58	1.74	2.27	1.70
CSS	22.61	2.99	161.68	23.87	8.00	1.88
Dockerfile	0.572	0.42	1.27	1.90	1.21	1.88
Fortran	0.17	1.84	4.66	0.27	3.61	0.26
Go	25.74	4.73	54.60	9.30	25.83	8.62
Haskell	2.36	0.54	5.11	1.25	4.17	1.23
HTML	146.76	9.53	2,419.87	90.23	99.09	5.23
Java	89.30	20.15	548.00	154.28	199.68	62.27
JavaScript	141.65	21.11	1,115.42	108.87	199.99	66.91
Julia	1.54	0.30	6.12	0.45	1.83	0.43
Lua	3.28	0.56	33.91	2.35	15.22	2.24
Makefile	1.49	0.66	21.30	4.22	5.19	2.78
Markdown	75.25	21.0	281.04	82.78	244.17	81.42
Perl	2.63	0.39	7.82	1.15	5.66	1.06
PHP	66.84	15.90	224.59	46.03	183.70	45.14
PowerShell	1.25	0.27	3.97	0.68	2.46	0.66
Python	64.30	12.96	233.29	56.93	191.61	56.19
R	0.30	0.04	22.39	5.15	19.05	4.29
Ruby	7.14	3.41	31.70	17.79	23.38	17.51
Rust	9.53	1.38	15.60	2.22	12.43	2.19
Scala	4.86	1.36	12.73	4.45	11.30	4.32
Shell	3.38	22.69	19.82	10.68	13.51	10.01
SQL	12.22	0.99	281.45	5.29	35.75	4.52
Swift	0	0	23.76	7.23	22.32	7.16
TeX	5.44	0.55	35.86	3.19	30.01	2.86
TypeScript	28.82	10.64	61.01	23.85	49.14	23.28
Visual Basic	1.49	0.16	16.63	1.06	7.48	0.81
Total	875.85	181.00	6,457.14	784.30	1,922.82	528.44

Visual data inspection Similar to the first StarCoder, we involve the BigCode community in a data inspection sprint to remove extensions with low-quality training data. We start from the annotations of the previous iteration that eliminated 36 out of the 300 extensions (of the 86 included programming languages). For StarCoder2, we only ran the data inspection for the not-yet-annotated programming languages (i.e., excluding the 86 languages of StarCoderBase). To streamline this process, we limited our inspection to extensions that include over 1,000 files and represent over 0.5% of the files in their respective languages. The remaining extensions were retained without further inspection, as they only make up a small volume. With the help of 15 annotators from the BigCode community, we visually inspected around 1000 extensions and excluded 130 (see appendix A.1 for the complete list). Our data inspection step excluded 39 programming languages from the dataset (appendix A.2), resulting in a final count of 619 programming languages.

Basic filters We apply a set of basic filters to the dataset to remove autogenerated files, data files, or other low-quality training data.

- *Long line filters*: we first remove all files with more than 100k lines as those files are likely to be data or generated code. We also remove files with an average line length of more than 100 characters or a maximum line length of more than 1000 characters for all languages, excluding HTML, JSON, Markdown, Roff, Roff Manpage, SMT, TeX, Text, and XML. For the mentioned languages, we remove files where the longest line exceeds 100k characters.

- *Autogenerated filter*: we remove files classified as auto-generated by the `is_generated` function of `go-enry` (go-enry, 2024). Additionally, we exclude files containing one of {"auto-generated", "autogenerated", "automatically generated", "generated automatically", "this file is generated"} in the first 5 lines of the file.

- *Alpha filter*: we remove files with less than 25% of alphabetic characters for all languages except Motorola 68K Assembly and WebAssembly, where we only remove files with less than 25% of alpha-numeric characters due to the syntax of those languages.

- *Encoded data filter*: we detect files with inline encoded data using the following regular expressions:
  - Base64 strings: `[a-zA-Z0-9+/\n=]{64,}`
  - Hexadecimal sequences: `(?:\b(?:0x|\\x)?[0-9a-fA-F]{2}(?:,|\b\s*)){8,}`
  - Unicode strings: `(?:\\u[0-9a-fA-F]{4}){8,}`

  We remove the file if any of the substrings matching these expressions is longer than 1024 characters or if the fraction of matched characters is more than 50% of the file.

Language-specific filters In addition to the basic filters, we apply the following set of language-specific filters.

- For Text, JSON, YAML, Web Ontology Language, and Graphviz (DOT), we remove files with more than 512 lines to minimize the impact of repeated tokens in data files.

- For HTML, we keep only the files where visible text is at least 100 characters long and makes up at least 20% of the code, similar to the processing pipeline of StarCoder (Li et al., 2023).

- For Text, we keep only files with "requirement" in the lowercased filename, or if the filename without the extension is one of {"readme", "notes", "todo", "description", "cmakelists"}.

2.2 Github Issues

We incorporate GitHub issues collected from GHArchive (Github Archive, 2024). We exclude pull requests here as we process them separately in §2.3.

A Github issue consists of a series of events with actions, such as opening the issue, creating a comment, or closing the issue. Each event includes the author's username, a message, an action, and a creation date. We follow the processing pipeline of StarCoder (Li et al., 2023), which we recap below:

- First, we removed auto-generated text when users replied to issues via email (for more information, see Li et al., 2023, Appendix A). We also deleted issues with a short message (less than 200 characters) and truncated long comments in the middle to a maximum of 100 lines while retaining the last 20 lines. This removed 17% of the volume — a similar percentage as in StarCoderBase.

- Next, we excluded comments from bots. To do so, we searched for keywords in the username of the comment's author (for more information, see Li et al., 2023, Appendix A). This step eliminated 3% of the issues, much less than the 17% reported in StarCoder (Li et al., 2023). This discrepancy is primarily because our dataset does not include pull requests, which are often the source of a significant proportion of bot-generated content.

- We used the number of users engaged in the conversation as an indicator of quality. Our criterion was to include conversations that have two or more users. However, we also preserved conversations that involved a single user if the total text within comments was less than 7,000 characters (96th percentile). Additionally, we excluded issues authored by a single user if they contained more than ten events, as they tended to be of poor quality or originate from overlooked bots. By implementing these filters, we removed 38% of the remaining issues. Lastly, we anonymized the usernames in the conversations by replacing them with a participant counter within the conversation (following the process of StarCoder).

2.3 Pull Requests

We include code reviews by gathering pull request events from GHArchive (Github Archive, 2024) and the corresponding source code from Software Heritage (Software Heritage, 2024b). Pull requests are requests to merge particular code changes from one branch into another on GitHub. Typically, they involve multiple rounds of code review discussions and additional cycles of code changes before they get merged into the target branch.

Data collection Specifically, for each pull request, we aggregate the PullRequestEvent, PullRequestReviewEvent, PullRequestReviewCommentEvent, IssueCommentEvent, and IssuesEvent events found on GHArchive. More details about the differences between these events can be found in the Github documentation. Next, we extract all base and head commit IDs from these events and retrieve the corresponding code files from Software Heritage. As we do not have access to the commit diffs, we generate them by identifying changes between files at the same path. We consider files present in the base but absent in the head as deletions, while we consider files absent in the base but present in the head as additions. This process yields approximately 300M PRs, accompanied by a volume of 15 TB of base code. Among these, there are 215M closed PRs originating from around 24M repositories.

PR filters We remove PRs that 1) have been opened by bots, 2) consist only of comments by bots, 3) have a non-permissive license, 4) have been opted out, 5) changes the base during the PR, 6) are not approved or merged, or 7) lack initial diffs (either due to absent data from Software Heritage or because all data have been filtered in other steps).

File filters We remove files from the base commit if they satisfy one of the following conditions: 1) the file is a deletion or addition, 2) the file length exceeds 1 million characters, 3) the fraction of alphanumeric characters is less than 0.25, 4) the fraction of hexadecimal characters is greater than 0.25, 5) the max number of lines surpasses 100,000, 6) the average line length exceeds 100, 7) the max line length surpasses 1,000, or 8) the presence of non-English text in Markdown

Title and description filtering We apply the following heuristic filters to clean up the PRs further. We exclude PRs with changes to the base, those not approved or merged, and those lacking initial diffs (either due to absent data from Software Heritage or being filtered out in previous steps). We also exclude PRs when the title is less than 10 characters or contains the words 'dependencies', 'dependency', 'depend', or 'release'. We exclude PRs when the description is less than 20 characters or contains 'Qwiet'.

Truncating inputs We shorten lengthy input fields in the PRs as follows. We truncate titles to 500 characters and descriptions to 80 lines, only displaying the first 60 and the last 20 lines. If the description length still exceeds 1000 characters, we truncate it.

Processing comments Following the processing of GitHub issues (§2.2), we remove comments from bots and strip auto-generated text when users post via email reply. We anonymize the usernames of authors as described in §3.2. We remove comments from PRs with less than 20 characters unless they are PR review comments. For code review comments, we remove the full diff hunk if it exceeds 10,000 characters while keeping the filename and comment.

Subsampling PRs To increase the diversity in the PRs, we sub-sample them on a per-repository basis. For repositories with 1 PR (after filtering), we retain it with a probability of 0.8. We linearly decrease this retention probability to 0.1 for repositories with 1,000 PRs. For repositories with more than 1,000 PRs, we set the retention probability such that we retain only 100 PRs. Finally, we sub-sample YAML and JSON files with 10% retention probability when their file size exceeds 50% of the total base files size or when the file path contains one of the keywords: 'pack', 'lock', 'yarn', 'output', 'swagger', 'openapi', or 'output'.

Max sequence length We determine the maximum sequence length of PRs by first investigating the data distribution after the processing steps mentioned above. We find 3.7M PRs with up to 1M characters, resulting in 194 GB of data. This reduces to 3.3M PRs when we set a limit of 100K characters, resulting in a dataset size of 67.3 GB. (appendix A.5 has more details about sequence length statistics.) For the StarCoder2 models, we opt to include PRs with up to 100K characters (translating to roughly 25k tokens). Since we are pre-training with a limited context of 4K tokens, not all PRs fit into the context window. However, as described in §5.2, we format the PRs so that the diffs are local and do not require long context.

2.4 Notebooks

We include notebooks from two separate sources: Jupyter notebooks extracted from the Software Heritage archive and notebooks released by the Kaggle platform.

2.4.1 Jupyter Notebooks

We transform Jupyter Notebooks into scripts and structured notebooks following the same pipeline as StarCoder (Li et al., 2023). One key difference is that we keep the markdown structure of the text blocks while it is removed in StarCoder. For completeness, we recap these preprocessing steps below.

Jupyter – scripts We utilize Jupytext[4] to convert notebooks to scripts. To initiate the conversion process, Jupytext requires the identification of the specific programming languages within each notebook. This information is typically available in the metadata of most notebooks. In cases where it is not, we use the Guesslang library[5] to identify the programming language, using a probability threshold of 0.5 or higher. Our initial dataset comprised 11 million notebooks, of which 3 million were excluded due to parsing errors. After near-deduplication, the dataset was reduced to 4 million notebooks converted to scripts.

Jupyter – structured To create this dataset, we first filtered out notebooks that did not contain any Python code or Markdown text using the metadata information of each notebook. Only notebooks explicitly marked as 'Python' in the metadata were kept. Then, for each notebook, consecutive Markdown blocks or code blocks were merged into a single Markdown or code block, respectively. Eventually, we ended up with consecutive code-text pairs in temporal order grouped by each notebook. Each Jupyter code-text pair contained the Markdown text immediately preceding the code block and the Python code, forming a natural instruction pair. We also included the formatted output of a code block if the output cell was non-empty; otherwise, it was marked by a special `<empty_output>` token. If consecutive code blocks have multiple output cells before merging, we only retain the output of the last code block. After these preprocessing steps and near-deduplication, we ended up with 4.6M structured Jupyter notebooks.

2.4.2 Kaggle Notebooks

We include Python notebooks released by the Kaggle platform[6] under an Apache 2.0 license, starting with an initial dataset of 3.6M notebooks. Note that this Kaggle dataset does not include the output cells, only the markdown and code cells.

Cleaning We start the data cleaning process by dropping notebooks with less than 100 characters and those with syntax errors. We also remove the templated text at the beginning of notebooks (see appendix A.7

[4] https://jupytext.readthedocs.io/
[5] https://guesslang.readthedocs.io/
[6] https://www.kaggle.com/datasets/kaggle/meta-kaggle-code

for the templates). These steps remove 18% of the notebooks. Next, we convert the notebooks to the structured and script format, following the processing of the Jupyter notebooks in §2.4.1. Finally, we remove near-duplicates using the pipeline described in §3.1, eliminating 78% of the notebooks and leaving us with 580k notebooks.

Dataset description To provide the model with more context regarding the content and objectives of the notebook, we include metadata about the Kaggle dataset whenever this information is available. We find that 42% of the notebooks are associated with a Kaggle dataset and include its title and description at the beginning of each notebook.

Dataset schema In addition to these high-level dataset descriptions, we scanned the code inside the notebooks for instances of `read_csv`. We found that 25% of the samples were loading CSV datasets. We extracted and incorporated detailed information about these datasets as follows. First, we used the Kaggle API to download the datasets and successfully retrieved 8.6% of the notebooks. The remaining cases were attributed to either the dataset being unavailable or encountering challenges downloading it within a reasonable time frame. For the downloaded datasets, we prefix the output of `df.info()` to the notebook, which displays the column names and their dtypes, the non-null values count, and the memory usage. We also include four sample rows from the dataset.

2.5 Documentation

Documentation from package managers We crawl documentation from several package manager platforms, including npm, PyPI, Go Packages, Packagist, Rubygems, Cargo, CocoaPods, Bower, CPAN, Clojars, Conda, Hex and Julia. We first retrieve the names of the most popular libraries across various platforms from libraries.io. These library names are then used to search through individual package managers, enabling us to obtain the respective homepages for each library. We systematically crawled the documentation files from the obtained homepage links or, alternatively, extracted information from the provided README or documentation files on the platform. For documents obtained through homepage links, we adhere to the same processing strategy outlined below in the paragraph titled "Documentation from websites". When extracting documents from the REwang2023softwareADME or documentation files on the platform, we employ distinct heuristics to extract the text using markdown formats whenever feasible, aiming to maintain a simple and effective format. It is worth noting that many libraries available on PyPI and Conda have their associated documentation hosted on Read the Docs, which typically offers more comprehensive documentation. Consequently, we prioritize utilizing Read the Docs as the primary source of documentation for these libraries. For these documents hosted on Read the Docs, we follow the same processing procedure outlined in the paragraph titled "Documentation from websites".

PDFs from package managers For documents related to the R language, we extracted text from all PDF files hosted on CRAN using the pdftotext library.[7] This library is particularly effective in preserving the formatting, including spaces within code snippets. For LaTeX-related documentation, we extracted the documentation, tutorial, and usage guide PDFs of LaTeX packages from CTAN, filtered out image-heavy PDFs, and converted the rest into markdown using the Nougat neural OCR tool.

Documentation from websites We collect code documentation from a carefully curated list of websites as detailed in Table 2. We start by systematically exploring the website from its initial URL listed in Table 2, using a queue to store URLs within the same domain. This queue expands dynamically as we discover new links during the crawl. Given that most documents comprise HTML pages, we focus our processing pipeline on (1) content extraction and (2) content concatenation. To extract the content, we utilize the `trafilatura` library[8] to convert each HTML page into XML format, simultaneously eliminating redundant navigation and index bars, elements that often recur in documentation. Next, we converted the XML format to markdown using our XML-to-Markdown conversion script. In the second stage, to compile these documents into a single text, we first do a near-deduplication of the content extracted from different HTML pages. This

[7] https://github.com/jalan/pdftotext
[8] https://github.com/adbar/trafilatura

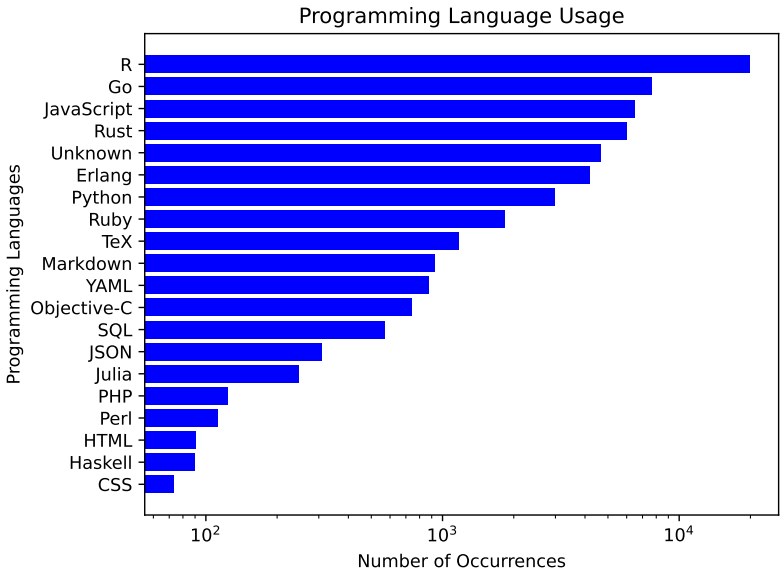

Figure 2: *The distribution of the top* 20 *programming languages in our crawled documentation collection.*

step was essential since we have observed that certain document pages only comprise website layouts (e.g., navigation bars) instead of fruitful information for documents, resulting in a substantial amount of duplicated content. To accomplish this, we treat each HTML page from a single website as a cluster and apply the minhash locality-sensitive hashing technique to identify and eliminate similar pages, using a threshold of 0.7. Finally, we assemble the gathered content from different pages of the same website in the order of web page crawling, ensuring a cohesive narrative. This parallels the "breadth-first search" approach, where all nodes at the current depth are explored before proceeding to the next depth level. Also, we collected code-relevant data from existing web crawls such as **RefinedWeb** (Penedo et al., 2023), **OSCAR** (Ortiz Suárez et al., 2019), and **esCorpius** (Gutiérrez-Fandiño et al., 2022). We use regular expressions to identify programming language-specific constructs within the documents and to detect the "docs." substring in the page URLs. The resulting dataset primarily comprises content sourced from programming blogs, coding tutorials, and platforms like Read the Docs, with the exclusion of the documents gathered above.

Free textbooks We scraped free programming books compiled in the Free Programming Books project, which aims at promoting the distribution of free programming e-books. First, we extract all links and identify those with a PDF extension. Subsequently, we downloaded all available PDF files and utilized the `pdf2text` library to extract text from these PDF files. Finally, we parsed 3,541 books whose languages span across different regions, including English, Chinese, Japanese, Spanish, and others.

Language identification Finally, we have employed a dual approach to identify the main programming language used by each document. We leverage predefined rules when the source of the document unequivocally corresponds to a specific programming language and resort to the `guesslang`[9] library in cases where such correspondence is not explicit. The resultant programming language distribution is graphically represented in Figure 2.

2.6 Intermediate Representations

We augment source code by pairing its intermediate representations (IR) to enhance the model's understanding of low-resource programming languages. The key rationale behind this approach is that a shared intermediate

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

Table 2: *The websites scraped for the code documentation dataset.*

Website Name	URL
DevDocs API Documentation	https://devdocs.io
MDN Web Docs	https://developer.mozilla.org
TensorFlow Docs	https://www.tensorflow.org
Linux Docs	https://www.kernel.org/doc/Documentation
Swift Programming Language	https://docs.swift.org/swift-book/documentation/the-swift-programming-language
Flutter API Reference	https://api.flutter.dev
TypeScript	https://www.typescriptlang.org/docs/handbook
Json.NET Documentation	https://www.newtonsoft.com/json/help/html
NVIDIA Documentation Hub	https://docs.nvidia.com
Oracle Java Tutorial	https://docs.oracle.com/javase/tutorial/java
Qiskit Documentation	https://qiskit.org/documentation
Q# Quantum Programming	https://learn.microsoft.com/en-us/azure/quantum/user-guide
Pony Tutorial	https://tutorial.ponylang.io
Zephir Documentation	https://docs.zephir-lang.com/0.12/en/introduction
Qemu Documentation	https://www.qemu.org/documentation
C# Documentation	https://learn.microsoft.com/en-us/dotnet/csharp
Hugging Face Documentation	https://huggingface.co/docs
LLVM Doc	https://llvm.org/docs
GCC Online Documentation	https://gcc.gnu.org/onlinedocs
Matlab Documentation	https://www.mathworks.com/help/matlab
Boost C++ Libraries	https://www.boost.org/doc
Maxima Manual	https://maxima.sourceforge.io/docs/manual/maxima_singlepage.html
Qt Documentation	https://doc.qt.io

representation might help to anchor low-resource constructs to similar ones in high-resource languages (Zhuo et al., 2023b).

LLVM We select LLVM (Lattner & Adve, 2004) as the intermediate representation due to its widespread availability on GitHub, increasing the probability that there is sufficient training data to learn the semantics of the language. In addition, LLVM is widely adopted as an IR and is the target representation of many compiler frontends across several programming languages.[10]

Data collection Existing attempts to extract IR from free-form source code either suffer from low compilation success rates (Szafraniec et al., 2023) or use bespoke language-specific mechanisms to track dependency code to compile successfully (Grossman et al., 2023). We sidestep this by sourcing self-contained compilation units from accepted solutions to programming word problems (Rosetta Code, 2023; Mirzayanov, 2020; Puri et al., 2021; Caballero et al., 2016). We compile ≈4M sources in total across C++, C, Objective-C, Python, Rust, Go, Haskell, D, Fortran, Swift, and Nim in size optimized (`-Oz` equivalent) and performance optimized (`-O3` equivalent) mode. We opt to use the size-optimized IR in most of the pairs due to context length considerations. However, for 20% of the pairs, we use the performance-optimized IR. This is done to maximize transfer from the pre-training stage, where the model sees LLVM code in the wild, which is more likely to be in this form. We use `clang`[11] for compiling C++, C and Objective-C, `codon`[12] for compiling Python, `rustc`[13] for compiling Rust, `gollvm`[14] for compiling Go, `ghc`[15] for compiling Haskell, `ldc`[16] for compiling D, `flang`[17] for compiling Fortran, and `nlvm`[18] for compiling Nim. We clean headers along with superfluous platform, vendor, and memory layout-specific information from the IR before pairing it with its source.

[10] https://llvm.org/ProjectsWithLLVM/
[11] https://clang.llvm.org/
[12] https://docs.exaloop.io/codon
[13] https://www.rust-lang.org/
[14] https://go.googlesource.com/gollvm/
[15] https://www.haskell.org/ghc/
[16] https://wiki.dlang.org/LDC
[17] https://flang.llvm.org/docs/
[18] https://github.com/arnetheduck/nlvm

2.7 LHQ[19]

We include several small high-quality datasets for math and coding:

- **APPS (train)** (Hendrycks et al., 2021) is a popular text2code benchmark in Python with a train set of 5,000 examples. We include one solution per programming problem.
- **Code Contest** (Li et al., 2022) is similar to APPS but includes solutions in several programming languages, namely Python 2/3, C++, and Java. We include one solution per problem and language and arrive at a dataset of 13k+ examples.
- **GSM8K (train)** (Cobbe et al., 2021) is the train split of GSM8K, a popular evaluation benchmark for testing the math reasoning capabilities of LLMs. The dataset consists of 7k+ examples.
- **GSM8K (SciRel)** (Yuan et al., 2023) is an augmented version of GSM8K that includes alternative reasoning paths for the questions in GSM8K. The extended version contains 110k examples.
- **Deepmind Mathematics** (Saxton et al., 2019) is a synthetic dataset of math questions and answers across various domains (algebra, arithmetic, calculus, comparison, measurement, numbers, polynomials, probability) and varying difficulty (easy-medium-hard). The dataset consists of 110M+ (short) examples.
- **Rosetta Code** (Rosetta Code, 2023; Nanz & Furia, 2015) is a dataset with over 1100 everyday programming tasks with solutions in as many different programming languages as possible.
- **MultiPL-T** (Cassano et al., 2023a) is high-quality data in Lua, Racket, and OCaml based on automatically translating extracted Python functions and validating them with unit tests. The total dataset comprises over 200k examples.
- **Proofsteps** is part of the AlgebraicStack (Azerbayev et al., 2024), a dataset used to train the Lemma family of models. We also include *proofsteps-lean*, which was extracted from mathlib 4 (mathlib Community, 2020), and *proofsteps-isabelle*, which was built on top of the PISA dataset (Jiang et al., 2021). Proofsteps-lean contains over 3k examples, while proofsteps-isabelle contains over 250k examples.

2.8 Other Natural Language Datasets

StackOverflow We include 11 million questions and their corresponding multiple responses from the Stack Overflow dump dated 2023-09-14 (StackExchange Archive, 2024). We filtered out questions with fewer than three answers. Upon inspecting the dataset, we found many mismatches between questions and answers due to inherent format errors in the Stack Overflow dump. We leveraged `Llama-2-70b-chat-hf` (Touvron et al., 2023) to increase the quality of the dataset as follows. We selected 20,000 examples and asked `Llama-2-70b-chat-hf` to rate the question-answer pairs. See Appendix A.6 for the exact prompt. Next, we pick the 10,000 highest-scoring pairs as positive examples and use the remaining 10,000 answers to create negative examples by randomly pairing them with other questions. We use this dataset to train a binary classifier by embedding the question and answer with a well-performing sentence embedding model (`sentence-transformers/all-MiniLM-L12-v2`[20] (Reimers & Gurevych, 2019; Muennighoff et al., 2022a)) and minimizing the cosine distance between them. Next, we plot the embedding scores for a subset of the question-answer pairs and manually determine the threshold to 0.1. As a question can have multiple answers, we average the scores of question-answer pairs and remove all questions with an average score below 0.1. We end up with 11.4 million questions and over 10B tokens.

ArXiv We include the ArXiv subset of the RedPajama dataset (Together Computer, 2023). This dataset is downloaded from the publicly available Amazon S3 bucket (Arxiv, 2024). We further processed the dataset only to retain latex source files and remove preambles, comments, macros, and bibliographies from these files. The final dataset is roughly 30B tokens.

[19] Leandro's High-Quality dataset

[20] https://huggingface.co/sentence-transformers/all-MiniLM-L12-v2

Wikipedia We include the English subset of Wikipedia. Specifically, we use the version collected by RedPajama (RedPajama Wiki, 2024), which is derived from the `2023-03-20` dump. We follow RedPajama's processing steps and eliminate hyperlinks and templates from the Wikipedia pages. The full dataset comprises around 6 billion tokens.

OpenWebMath We include OpenWebMath (Paster et al., 2023), an open dataset of high-quality mathematical text extracted from CommonCrawl. The full dataset comprises almost 15B tokens.

3 Preprocessing Pipeline

We apply several preprocessing steps, such as deduplication (§3.1), PII redaction (§3.2), benchmark decontamination (§3.3), malware removal (§3.4), and opt-out deletion requests (§3.5), to the data sources described in the previous section. Since not all steps are applied to each data source, we summarize the preprocessing pipeline per data source in Table 3.

3.1 Removing Near-Duplicates

We deduplicate the source code, pull requests, notebooks, issues, and documentation. We do not deduplicate the already preprocessed natural language datasets, such as Arxiv, StackExchange, OpenWebMath, Wikipedia, and the small high-quality math and reasoning datasets.

We followed the deduplication pipeline of SantaCoder (Ben Allal et al., 2023). This process first calculates the MinHashes (Broder, 2000) of all code files and then utilizes Locally Sensitive Hashing (LSH) to group files based on their MinHash fingerprints. During the LSH stage, "similar" files are assigned to the same buckets, identifying them as duplicates. Only one file from each duplicate group is chosen. In addition to the SantaCoder approach, to preserve repository context, we prioritize files from repositories with higher star and fork counts or from the latest commit date as a tiebreaker. We used 5-grams and a Jaccard similarity of 0.7. We refer to this blogpost for more background information regarding the deduplication pipeline.

3.2 PII Redaction

To reduce the likelihood of re-distributing Personally Identifiable Information (PII) present in the training data, we make diligent efforts to redact PII from the training set. We largely follow the steps from StarCoder (Li et al., 2023) and leverage the StarPII model to redact various PII entities. Below, we provide more details on how we apply it to each data source.

Redacting PII entities We use StarPII to redact names, emails, keys, passwords, IP addresses, and usernames from source code, pull requests, issues, and StackOverflow. We do not make any modifications to the model or redaction logic described in the StarCoder paper (Li et al., 2023). For OpenWebMath and documentation, we only redact names, keys, and emails, while we only redact emails for arXiv using the regex described in Ben Allal et al. (2023).

Redacting usernames The conversations in issues, pull requests, and StackOverflow often contain usernames in the message thread. We anonymize the author usernames by substituting them with a participant counter specific to the conversation, like username_1 to represent the second participant. These pseudonyms are added at the start of each comment to maintain the speaker's identity. Moreover, any references to these usernames in the messages are removed. Only the usernames of actively participating individuals in the conversation are masked, and mentions of non-participating users remain unaffected.

3.3 Decontamination

To ensure the performance of StarCoder is not artificially inflated on our test benchmarks, we decontaminate the training set from our test sets. Specifically, we remove files that contain docstrings or solutions from HumanEval and MBPP, docstrings from APPS, questions from GSM8K, or prompts from DS1000. In contrast

Table 3: *Overview of the data processing steps applied to each data source.*

Dataset	Dedup	Malicious Code	Decontaminate	Opt-out	PII
Source Code	Yes	Yes	Yes	Yes	StarPII
Pull Requests	Yes	Yes	Yes	Yes	StarPII + Usernames
Jupyter/Kaggle Notebooks	Yes	Yes	Yes	Yes/No	StarPII
Issues	Yes	Yes	Yes	Yes	StarPII + Usernames
Docs	Yes	No	No	No	StarPII: Names, Keys, Emails
LHQ	No	No	No	No	No
Arxiv	No	No	No	No	Email
OpenWebMath	No	No	Yes	No	StarPII: Names, Keys, Emails
Wikipedia	No	No	No	No	No
StackExchange	No	No	Yes	No	StarPII + Usernames

to the first iteration of StarCoder (Li et al., 2023), we further enhance the recall of the decontamination process by removing whitespace during string matching. Note that we exclude docs, LHQ, arXiv, and Wikipedia from this decontamination step.

3.4 Malware Removal

We scan our training set to identify possible instances of malware in the source code, pull requests, notebooks, and issues. To this end, we use ClamAV 1.2 (ClamAV, 2024) with additional unofficial malware signatures published by SaneSecurity (Sane Security, 2024) as of 2023-11-16. Signatures with a high risk of False Positives (as determined by SaneSecurity) were not used. See Table 26 for the most frequently detected malware signatures in the unfiltered code dataset. In summary, this step eliminates 59,442 files from the dataset, constituting only 0.009% of the 654M files.

3.5 Removing Opt-outs

We announced the upcoming training run of StarCoder2 on X[21] and updated the "Am I in the stack" governance tool with the new repositories from The Stack v2. Developers were granted until November 20, 2023, to submit their opt-out requests. After the cut-off date, we eliminated 1,561 repositories associated with 91 users and organizations. A total of 22,066 files were removed from the source code dataset (excluding issues and PRs).

4 Data Composition

Model capacity With a much larger training set available, we decided to tailor our data composition to each model size. We reason that smaller models, having limited capacity, should be exposed to a less diverse dataset. This intuition is supported by research in multi-lingual NLP showing that languages compete for model capacity (Arivazhagan et al., 2019; Conneau et al., 2020; Scao et al., 2022b). Hence, we first create a smaller version of the SWH code dataset, selecting a subset of 17 widely-used programming languages. We use this variant to train the 3B and 7B models, whereas we use the full version with all 619 programming languages for the 15B model. To further limit the diversity in the training set for the 3B model, we also exclude some natural language datasets (see "Data composition per model size").

Downsampling languages Similar to StarCoderBase, we adhere to the natural distribution of the data as much as possible. Before constructing the source code datasets, we examined the data distribution among the programming languages. Compared to StarCoderBase, we found slightly larger variations among the high-resource languages. The observed data volume (in GB) is as follows: Java (479.68), JavaScript (277.25), C++ (204.49), Python (190.99), PHP (171.57), C# (166.22), and C (114.49). We decided to downsample both Java and Javascript to 200GB to put these high-resource languages on a more equal footing. Furthermore, we

[21] https://x.com/BigCodeProject/status/1721583097580249254?s=20

Table 4: *Overview of the data composition of StarCoder2 models. We refer to the training set of the 3B model as the-stack-v2-train-3B.*

	Dataset	Tokens (B)	3B	7B	15B
	`the-stack-v2-train-smol`	525.5	✓	✓	✗
	`the-stack-v2-train-full`	775.48	✗	✗	✓
`the-stack-v2-train-extras`	Pull requests	19.54	✓	✓	✓
	Issues	11.06	✓	✓	✓
	Jupyter structured	14.74	✓	✓	✓
	Jupyter scripts	16.29	✓	✓	✓
	Kaggle scripts	1.68	✓	✓	✓
	Documentation	1.6	✓	✓	✓
	OpenWebMath	14.42	✗	✓	✓
	Wikipedia	6.12	✗	✓	✓
	StackOverflow	10.26	✓	✓	✓
	Arxiv	30.26	✗	✓	✓
	LHQ	5.78	✓	✓	✓
	Intermediate Repr.	6	✓	✓	✓
	Unique tokens (B)		622.09	658.58	913.23

preserved 254GB of markdown data while reducing the size of HTML to 100 GB. This decision was driven by the anticipation that markdown would likely contain more code documentation, whereas HTML is commonly associated with webpages. Lastly, we subsampled data files like JSON, XML, and YAML to 8GB and a few other data formats to 1 GB. See Table 28 in Appendix C.2 for the full list of subsampled languages.

Repository-context After subsampling some programming languages, we compile the source code from Software Heritage into repository-context-aware datasets. Each example in the dataset is a full repository with files arranged in a random order. As previously noted, we create two versions of the SWH dataset, `the-stack-v2-train-smol` and `the-stack-v2-train-full`, as further detailed in the subsequent paragraphs.

The-stack-v2-train-smol For the small variant, we select 17 widely used programming languages and include a curated set of documentation and configuration languages.

- Specifically, we include the following programming languages:
  - C
  - C#
  - C++
  - Go
  - Java
  - JavaScript
  - Kotlin
  - Lua
  - PHP
  - Python
  - R
  - Ruby
  - Rust
  - SQL
  - Shell
  - Swift
  - TypeScript
- And incorporate the following languages associated with code documentation:
  - AsciiDoc
  - HTML
  - Markdown
  - RDoc
  - RMarkdown
  - Text
  - reStructuredText
- We also include several configuration languages and files, which we list in Appendix C.1.
- Despite limiting the languages to this subset, we obtain a dataset of 525B+ unique tokens.

The-stack-v2-train-full For the full variant, we include all 619 programming languages. Although this subset significantly enhances language diversity (adding 600+ programming languages), it contributes only around 250B tokens to the dataset, culminating in 775B+ tokens.

Data composition per model size In Table 4, we summarize the data composition for the 3B, 7B, and 15B models. We use `the-stack-v2-train-extras` to denote all supplementary sources gathered for StarCoder2, excluding the source code obtained from SWH. For the 3B, we use `the-stack-v2-train-smol` and exclude OpenWebMath, Wikipedia, and Arxiv from the extra data sources in §2. This leads to a dataset of 622B+ unique tokens. For the 7B, we include OpenWebMath, Wikipedia, and Arxiv, leading to a slightly larger dataset of 658B+ unique tokens. For the 15B, we include `the-stack-v2-train-full` dataset and all extra data sources listed in §2, resulting in a dataset with 913B+ unique tokens. The size of this dataset is 4× the size of the training dataset for StarCoderBase.

5 Data Formatting

We present the formatting guidelines for each of the data sources below. We provide the templates below in which ⟨token⟩ refers to a sentinel token, and metadata and data refer to placeholders for data fields, respectively.

5.1 Source Code

We prepend the repository name and file paths to the context of the code file. We only add this metadata with a 50% probability to enable the model to operate without this information. We use the following format when adding the repository name and file paths:

```
<repo_name>reponame<file_sep>filepath1\ncode1<file_sep>filepath2\ncode2 ... <|endoftext|>.
```

We use the following format when we do not include this meta-data:

```
<file_sep>code1<file_sep>code2 ... <|endoftext|>.
```

Repository-context Starcoder1 was trained with file-context, i.e., the setting where random files are joined into the context window. In this work, we explore training with repository-context, wherein files from the same repository are grouped together. While we considered various methods for grouping files within the repository, we ultimately arranged them in a random order within the same repository.

FIM To enable the model to perform code infilling tasks, we apply the fill-in-the-middle transformation (FIM; Bavarian et al., 2022) to the source code. While we explored several FIM variants in preliminary experiments, we opted for repo-context file-level FIM in the StarCoder2 models. In this FIM variant, repositories are selected with a 50% chance of being candidates for FIM. The selected repository examples are split by `<|endoftext|>` and `<file_sep>` tokens. Next, we apply the FIM transformation to each chunk with a 50% probability. We do not apply FIM to the repository metadata (`<repo_name>reponame`). Below, we provide an example of the FIM format when it's only applied to the second source file:

```
<repo_name>reponame<file_sep>filepath0\ncode0<file_sep><fim_prefix>filepath1\n
code1_pre<fim_suffix>code1_suf<fim_middle>code1_mid<file_sep> ...<|endoftext|>
```

5.2 Pull Requests

Formatting pull requests is challenging as we aim to create a compact representation of a potentially long sequence of code changes and comments. We refer to §2.3 for details on how we removed and truncated long input fields of the pull request. Here, we focus on how to render the PR into a structured format that can be consumed by the LLM.

For files part of the base commit, we include the entire file with 0.2 probability; otherwise, we display a range of changes in the base files across all commit heads of the PR.[22] We randomly add up to 32 lines before and after the changes.

[22]We take the union of file line changes in all commits

We use diff hunks to display modifications between the before and after state of the file, ensuring that changes are reasonably localized. Additionally, within the diff hunks, we incorporate 3-10 randomly selected context lines both before and after the specific change.

We structure the PR format as follows. The first block presents the title, description, and complete base files or modifications made to them. Subsequently, we outline the first set of head diff hunks:

```
<pr>Title: title\nusername_0: description
<pr_status>opened
<repo_name>reponame

<pr_base>
<pr_file>filepath_1
<pr_base_code>file_content/changes_1
...
<pr_file>filepath_N
<pr_base_code>file_content/changes_N

<pr_diff>
<pr_file>filepath_1
<pr_diff_hunk>diff_hunk_1
...
<pr_diff_hunk>diff_hunk_K
...
<pr_file>filepath_M
<pr_diff_hunk>diff_hunk_1
...
<pr_diff_hunk>diff_hunk_J
```

The second block is repeated for each new head commit in the PR, covering general comments, review comments, and code review comments. The block concludes with the diff hunks between the pull request base and the new head, reflecting the outcome of discussions and comments. Note that it's also possible for users to close and reopen the pull request. As in Github issues, we refer to authors by their participant counter within the conversation, e.g., username_1, to refer to the second participant in the issue.

```
<pr_comment>username_id: comment
<pr_event_id>comment_id
...
...
...
<pr_review>username_id: review_comment\n
<pr_event_id>review_id
<pr_review_state>[approved, rejected, commented, changes_required]
...
...
...
<pr_review_comment>
<pr_event_id>comment_id
<pr_in_reply_to_review_id>review_id (opt)
<pr_in_reply_to_comment_id>comment_id (opt)
<pr_file>filepath
<pr_diff_hunk_comment_line>line_number
<pr_diff_hunk>diff_hunk_content
<pr_comment>username_id: comment
```

```
...
...
...
<pr>username_id
<pr_status>closed
<pr_is_merged>False
...
<pr>Title: title\nusername_id: description
<pr_status>[opened, reopened, edited]
...
...
...
<pr_file>filepath_1
<pr_diff_hunk>diff_hunk_1
...
<pr_diff_hunk>diff_hunk_K
...
<pr_file>filepath_M
<pr_diff_hunk>diff_hunk_1
...
<pr_diff_hunk>diff_hunk_J
```

We only add the following final block when the PR is closed.

```
<pr>username_id
<pr_status>closed
<pr_is_merged>True
<|endoftext|>
```

5.3 GitHub Issues

We use sentinel tokens to mark the opening of an issue and subsequently include its title. We separate the sequence of comments by a `<issue_comment>` token and include an anonymized speaker identifier before the comment. Specifically, we refer to authors by their participant counter within the conversation, e.g., username_1, to refer to the second participant in the issue. To distinguish between the different turns, we use comment_1, id1 to refer to the second comment and its anonymized speaker id, respectively. The `<issue_closed>` token is added if the issue is closed.

```
<issue_start>Title: title\nusername_id0: comment_0<issue_comment>username_id1: comment_1
... <issue_closed (optional)><issue_comment>username_idn: comment_n<|endoftext|>
```

5.4 Notebooks

Jupyter – scripts We format Jupyter scripts as a single code block, starting with a `<jupyter_script>` token.

```
<jupyter_script>code<|endoftext|>
```

Jupyter – structured Parsed Jupyter notebooks are chains of text, code, and outputs. We separate the cells with sentinel tokens. Note that we use text2, code2, output2 to refer to the 3rd triplet in the notebook.

```
<jupyter_start><jupyter_text>text0<jupyter_code>code0
<jupyter_output>output0<jupyter_text> ... <|endoftext|>
```

Kaggle – scripts When available, we prepend the associated dataset title and description to Kaggle notebooks (42% of the samples). For 8.6% of the notebooks, we add granular information on the dataset's schema. Below is the format we use:

```
<jupyter_start><jupyter_text>title\ndescription\nKaggle dataset identifier: data_identifier
<jupyter_code>import pandas as pd\n\ndf = pd.read_csv(data_path1)\ndf.info()
<jupyter_output>df_info_output1
<jupyter_text>Examples:\nexample1_1\n..example1_4
...
<jupyter_script>code<|endoftext|>
```

Some notebooks might load more than one `csv` file, so we repeat the blocks of data information content for all files.

Note that we introduce a new special token `<jupyter_script>` to append the final script of the converted Kaggle notebook. This token helps differentiate the script, which is usually long, from code that follows `<jupyter_code>` token, typically shorter.

Kaggle – structured Structured Kaggle notebooks are similar to structured Jupyter notebooks, except that they don't have an output cell, so we only include text and code blocks and keep the tokens used in Jupyter Notebooks:

```
<jupyter_start><jupyter_text>text0<jupyter_code>code0<jupyter_text> ... <|endoftext|>
```

5.5 StackExchange

We concatenate questions and answers in the StackOverflow dataset using a format similar to the GitHub issues. We start with the question and then add answers in random order. We include the upvote score alongside the answer and, if applicable, denote it as the selected answer. Note that we do not have the title of the conversations for the StackExchange dataset.

```
<issue_start>username_id0: question
<issue_comment>username_id1: answer_1\nUpvotes: score [selected answer](Optional)
...
<issue_comment>username_idn: answer_n\nUpvotes: score [selected answer](Optional)<|endoftext|>
```

5.6 Intermediate Representations

We split 50/50 between translating from source code to intermediate representation (`code->intermediate`) and vice-versa (`intermediate->code`). Regarding the intermediate representation, we use the size-optimized version 80% of the time and the performance-optimized version 20% of the time. We use separate sentinel tokens to indicate the direction of the translation.

```
code<code_to_intermediate>intermediate_representation
intermediate_representation<intermediate_to_code>code
```

6 Model architecture and training details

In this section, we provide all details regarding the model architecture (§6.1), tokenizer (§6.2), training details (§6.3), and CO_2 emissions during training (§6.4).

[23] Estimated with 6ND, where N is the number of parameters and D is the number of training tokens. Includes base and long-context training.

Table 5: *Overview of the sentinel tokens.*

Token	Description
<|endoftext|>	end of text/sequence
<fim_prefix>	FIM prefix
<fim_middle>	FIM middle
<fim_suffix>	FIM suffix
<fim_pad>	FIM pad
<repo_name>	repository name
<file_sep>	file separator
<issue_start>	start of GitHub issue
<issue_comment>	start of GitHub issue comment
<issue_closed>	GitHub issue closed event
<jupyter_start>	start of Jupyter notebook
<jupyter_text>	start of Jupyter text cell
<jupyter_code>	start of Jupyter code cell
<jupyter_output>	start of Jupyter output cell
<jupyter_script>	start of Jupyter script (converted kaggle notebook)
<empty_output>	output cell without content
<code_to_intermediate>	translate source code to intermediate representation
<intermediate_to_code>	translate intermediate representation to source code
<pr>	start of pull request
<pr_status>	status of pull request
<pr_is_merged>	whether pr is merged
<pr_base>	start of list of base files
<pr_file>	path of pull request file
<pr_base_code>	code that is part of the base commit in the PR
<pr_diff>	start of a diff
<pr_diff_hunk>	diff hunk
<pr_comment>	general comment
<pr_event_id>	GitHub id of review comment or code review comment
<pr_review>	start of review
<pr_review_state>	review state (e.g. approved, rejected)
<pr_review_comment>	code review comment
<pr_in_reply_to_review_id>	GitHub event id of review
<pr_in_reply_to_comment_id>	GitHub event id of comment
<pr_diff_hunk_comment_line>	line number of code review comment

6.1 Model Architecture

We introduce a few architectural changes compared to StarCoderBase. First, we replace learned positional embeddings with Rotary Positional Encodings (RoPE; Su et al., 2021), as we confirmed significant performance gains in a preliminary ablation study. Following DeepseekCoder (Guo et al., 2024) and Code LLaMA (Rozière et al., 2023), we use a base period $\theta = 1e5$. The second architectural modification we make is replacing Multi-Query Attention (MQA; Shazeer, 2019) with Grouped Query Attention (Ainslie et al., 2023, GQA;). However, we keep the number of key-value heads relatively low—2 for the 3B, 4 for the 7B and 15B—to prevent significantly slowing down inference.

We summarize all other hyperparameters, such as the number of layers and hidden dimension, in Table 6.

Table 6: *Model architecture details of the StarCoder2 models.*

Parameter	StarCoder2-3B	StarCoder2-7B	StarCoder2-15B
`hidden_dim`	3072	4608	6144
`n_heads`	24	36	48
`n_kv_heads`	2	4	4
`n_layers`	30	32	40
`vocab size`	49152	49152	49152
`seq_len`	base-4k/long-16k	base-4k/long-16k	base-4k/long-16k
`positional encodings`	RoPE	RoPE	RoPE
`FLOPs`[23]	5.94e+22	1.55e+23	3.87e+23

Table 7: *Training details of StarCoder2 base models.*

Model	learning rate	RoPE θ	batch size	n iterations	n tokens	n epochs
StarCoder2-3B	3×10^{-4}	$1e5$	2.6M	1.2M	3.1T	4.98
StarCoder2-7B	3×10^{-4}	$1e5$	3.5M	1M	3.5T	5.31
StarCoder2-15B	3×10^{-4}	$1e4$	4.1M	1M	4.1T	4.49

6.2 Tokenizer

We follow the procedure of StarCoderBase and train a byte-level Byte-Pair-Encoding tokenizer on a small subset of The Stack v1.[24] In our preliminary experiments, we observed that increasing the vocabulary size to 100K did not improve performance. Hence, we decided to maintain a vocabulary size of 49,152 tokens, including the sentinel tokens from Table 5. The pre-tokenization step includes a digit-splitter and the regex splitter from the GPT-2 pre-tokenizer.

6.3 Training Details

Base models The models were trained with a sequence length of 4,096 using Adam (Kingma & Ba, 2015) with $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$ and a weight decay of 0.1, without dropout. The learning rate followed a cosine decay after a linear warmup of 1,000 iterations. Table 7 details the training hyper-parameters for each model. RoPE θ values are different for StarCoder2-15B due to a bug in parsing the training configuration. Moreover, StarCoder2-15B was scheduled to train for 1.1M iterations but was early stopped after 1M iterations. Following Muennighoff et al. (2023), we repeat data for around four to five epochs.

Long context We further pre-trained each model for long-context on 200B tokens from the same pre-training corpus, using a 16,384 context length with a sliding window of 4,096, with FlashAttention-2 (Dao et al., 2022; Dao, 2024). We increase RoPE θ and use the same configuration for the optimizer. The other training hyperparameters are provided in Table 8.

6.4 CO2 Emissions

We provide estimations of the CO_2 emission of the StarCoder2 training using the Machine Learning Impact calculator presented in Lacoste et al. (2019). Note that we calculate the CO_2 emissions by considering the total GPU hours of the base-model training. We then extrapolate this number to the long-context fine-tuning based on the number of tokens.

3B The compute infrastructure provided by ServiceNow had a carbon efficiency of 0.386 $kgCO_2eq/kWh$. A cumulative of 97,120 hours of computation was performed on hardware of type A100 SXM4 80 GB (TDP of

[24] https://huggingface.co/datasets/bigcode/the-stack-march-sample-special-tokens-stripped

Table 8: *Training details for the long context training of StarCoder2 models.*

Model	learning rate	RoPE θ	batch size	n iterations	n tokens
StarCoder2-3B	3×10^{-5}	$1e6$	2.6M	80k	200B
StarCoder2-7B	2×10^{-5}	$1e6$	3.5M	56k	200B
StarCoder2-15B	3×10^{-5}	$1e5$	4.1M	50k	200B

Table 9: *Pass@1 on HumanEval(+) and MBPP(+). These results were generated using greedy decoding.*

Model	HumanEval	HumanEval+	MBPP	MBPP+
StarCoderBase-3B	21.3	17.1	42.6	35.8
DeepSeekCoder-1.3B	28.7	23.8	55.4	46.9
StableCode-3B	28.7	24.4	53.1	43.1
StarCoder2-3B	**31.7**	**27.4**	**57.4**	**47.4**
StarCoderBase-7B	30.5	25.0	47.4	39.6
CodeLlama-7B	33.5	25.6	52.1	41.6
DeepSeekCoder-6.7B	**47.6**	**39.6**	**70.2**	**56.6**
StarCoder2-7B	35.4	29.9	54.4	45.6
StarCoderBase-15B	29.3	25.6	50.6	43.6
CodeLlama-13B	37.8	32.3	62.4	52.4
StarCoder2-15B	**46.3**	**37.8**	**66.2**	**53.1**
CodeLlama-34B	48.2	44.3	65.4	52.4
DeepSeekCoder-33B	**54.3**	**46.3**	**73.2**	**59.1**

400W). Total emissions are estimated to be 14,995.33 kgCO_2eq. The long-context fine-tuning stage adds 1,111.68 kgCO_2eq, resulting in a total of 16,107.01 kgCO_2eq.

7B The compute infrastructure provided by Hugging Face had a carbon efficiency of 0.2925 kgCO_2eq/kWh. A cumulative of 145,152 hours of computation was performed on hardware of type H100 (TDP of 660W). Total emissions are estimated to be 28,021.6 kgCO_2eq. The long-context fine-tuning stage adds 1601.23, resulting in a total of 29,622.83 kgCO_2eq.

15B The paper will soon be updated with estimates for the 15B model.

7 Evaluation

We evaluate the StarCoder2 models on a variety of benchmarks from the literature and compare them to recent state-of-the-art open Code LLMs: StableCode (Pinnaparaju et al., 2024), Code Llama (Rozière et al., 2023), DeepSeekCoder (Guo et al., 2024), and original StarCoder (Li et al., 2023). Since StarCoder2 is a base model, we only compare it with the base models of the model families mentioned above.

We group all our comparisons by model sizes. The *small* models have 3B or fewer parameters, the *medium* models have 7B or fewer parameters, and the *large* models have 15B or fewer parameters. Finally, we include two *extra large* models: CodeLlama-34B and DeepSeekCoder-33B. These models are more than twice the size of the large StarCoder2 model. But, as we shall see below, StarCoder2-15B comes close to or even outperforms the extra-large models in several benchmarks.

7.1 Code Completion

We first evaluate the StarCoder2 models on code completion tasks, which have been widely studied in Code LLM work.

7.1.1 HumanEval, MBPP, and EvalPlus

About the benchmarks HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) are two of the most widely studied benchmarks for Code LLMs. Each benchmark has a few hundred programming problems. Each HumanEval problem has a prompt—a function signature and docstring—and a set of hidden unit tests. The prompt for each MBPP problem includes a natural language description followed by a few tests. The model under evaluation will complete the function given the prompt, and we test that function with the hidden unit tests. The result is considered a success only if all hidden tests pass.

Recently, Liu et al. (2023a) identified several issues with both benchmarks. (1) Most problems have inadequate hidden tests that cannot detect subtle bugs in solutions (See Listings 1 and 2); and (2) Several problems have wrong test cases and ambiguous descriptions, which unfairly penalize the models that interpret the statements in other reasonable ways (See Listings 2). They introduce the EvalPlus framework to address these problems. The resulting benchmarks (HumanEval+ and MBPP+) have 80× and 35× more tests than the original benchmarks. For rigorous evaluation, we adopt the EvalPlus framework in this study.

Listing 1: *A HumanEval task with insufficient tests*

```
def common(l1: list, l2: list) -> list:
 """Return sorted unique common elements for 2 lists"""
 common_elems = list(set(l1).intersection(set(l2)))
 common_elems.sort()
 return list(set(common_elems))

assert common([4,3,2,8], []) == []
assert common([5,3,2,8], [3,2]) == [2,3]
...

# [Explanation] This solution is wrong as applying set
# to the sorted common_elems does not preserve the
# order. Base HumanEval test inputs are too short to
# easily manifest the flakiness.
```

Listing 2: *An MBPP task with problematic tests*

```
"""Write a function to check whether all dictionaries
    ↪ in a list are empty or not."""
def empty_dit(list1): return all(not d for d in list1)

assert empty_dit([{},{},{}]) == True
assert empty_dit([{1,2},{},{}]) == True # Wrong test!
assert empty_dit([{}]) == True

# [Explanation] First, the second base test is wrong,
# falsifying any correct solutions. Second, the tests
# are weak, passing the wrong solution above. The wrong
# solution mistakingly yields False given [{}, {}, [1]]
# where we expect True as all dictionaries are empty
# and the non-empty is an array, not a dictionary.
```

Hyperparameters Following recent work on Code LLMs (Rozière et al., 2023; Guo et al., 2024), we use greedy decoding and report the mean pass@1 (mean success rate) for all problems in the benchmark.

Results The results for HumanEval, MBPP, and their EvalPlus variants are presented in Table 9.[25] From the table, we can make the following observations:

1. StarCoder2-3B is the best-performing small model on all the datasets (HumanEval, MBPP, HumanEval+, and MBPP+). The model is significantly better than its predecessor, StarCoderBase-3B, exhibiting improvements of 60.2% on HumanEval+ and 32.4% on MBPP+, respectively.

2. StarCoder2-7B comes in second place of the medium models. DeepSeekCoder-6.7B is stronger, outperforming StarCoder2-7B by 32.4% and 24.1% on HumanEval+ and MBPP+, respectively. However, StarCoder2-7B consistently outperforms all the other medium models, including both StarCoderBase-7B and CodeLlama-7B. StarCoder2-7B outperforms StarCoderBase-7B by 19.6% and 15.2% on HumanEval+ and MBPP+, respectively. Additionally, it surpasses CodeLlama-7B by 16.8% and 9.6% on these benchmarks.

3. StarCoder2-15B is the best-performing large model by a significant margin. For example, it scores 46.3, whereas CodeLlama-13B scores 37.8 on HumanEval. The results on EvalPlus are also consistent. For example, on HumanEval+, it significantly improves over StarCoderBase-15B and CodeLlama-13B by 47.7% and 17.0%, respectively.

[25] Note that EvalPlus omits a few ill-formed and noisy problems from the MBPP dataset. It uses 399 out of the 427 problems from the MBPP subset that was sanitized by the original authors (Austin et al., 2021). For HumanEval, we kept all 164 problems from the original dataset.

Table 10: *Pass@1 results on MultiPL-E averaged over 50 samples for each problem. All models are evaluated at temperature* 0.2 *and top-p* 0.95.

Model	C++	C#	D	Go	Java	Julia	JavaScript	Lua	PHP
StableCode-3B	**28.4**	14.4	**13.4**	19.3	27.8	**20.6**	32.0	17.1	23.7
DeepSeekCoder-1.3B	28.3	**21.3**	10.4	19.1	**29.2**	15.0	28.3	19.2	23.2
StarCoderBase-3B	19.4	13.3	5.0	13.3	19.2	16.1	21.3	18.0	18.6
StarCoder2-3B	27.2	20.5	12.6	**23.6**	27.4	19.9	**35.4**	**28.0**	**27.6**
CodeLlama-7B	26.4	21.0	11.6	20.9	28.2	25.9	31.6	30.4	25.1
DeepSeekCoder-6.7B	**46.7**	**32.9**	**18.4**	**31.0**	**39.7**	**31.4**	**46.6**	**34.2**	**32.6**
StarCoderBase-7B	23.3	19.3	8.1	19.6	24.4	21.8	27.4	23.4	22.1
StarCoder2-7B	33.6	20.7	15.1	20.2	29.4	20.4	35.4	30.7	30.6
CodeLlama-13B	37.4	24.8	15.5	**26.6**	**37.5**	27.9	39.3	31.6	33.9
StarCoderBase-15B	30.6	20.6	10.0	21.5	28.5	21.1	31.7	26.6	26.8
StarCoder2-15B	**41.4**	**29.2**	**23.6**	26.2	33.9	**33.2**	**44.2**	**43.8**	**39.5**
CodeLlama-34B	41.4	30.7	15.3	28.7	40.2	31.4	41.7	**37.5**	40.4
DeepSeekCoder-33B	**51.2**	**35.3**	**17.4**	**34.2**	**43.8**	**32.8**	**51.3**	36.5	**41.8**

Model	Perl	R	Ruby	Racket	Rust	Scala	Bash	Swift	TypeScript
StableCode-3B	9.4	11.5	0.8	7.0	22.9	5.9	8.6	13.2	29.6
DeepSeekCoder-1.3B	12.5	9.8	24.6	**9.1**	18.6	**19.6**	9.7	11.0	27.4
StarCoderBase-3B	11.3	10.1	4.2	7.9	16.3	16.8	3.8	10.0	22.8
StarCoder2-3B	**13.6**	**14.2**	**31.3**	7.8	**24.5**	18.9	**12.3**	**25.1**	**34.4**
CodeLlama-7B	16.9	14.9	29.5	11.4	25.5	22.8	9.6	24.9	33.4
DeepSeekCoder-6.7B	**30.4**	**20.5**	**46.2**	**17.4**	**37.7**	**35.2**	**22.2**	**30.3**	**39.5**
StarCoderBase-7B	15.2	14.5	19.6	11.1	22.6	20.9	7.3	15.1	27.5
StarCoder2-7B	16.6	16.7	28.3	11.6	29.6	19.5	12.2	26.1	36.3
CodeLlama-13B	23.4	14.1	31.9	13.0	31.0	29.7	13.3	30.1	40.1
StarCoderBase-15B	16.3	10.2	17.2	11.8	24.5	28.8	11.0	16.7	32.1
StarCoder2-15B	**37.2**	**19.8**	**41.5**	**22.4**	**38.0**	**37.4**	**18.9**	**34.2**	**43.8**
CodeLlama-34B	28.5	**22.7**	37.8	16.9	38.7	36.7	16.4	35.3	42.1
DeepSeekCoder-33B	**31.0**	20.5	**44.0**	**23.4**	**43.8**	**43.9**	**28.7**	**35.8**	**48.4**

4. StarCoder2-15B is even competitive with models that are more than twice its size. For example, StarCoder2-15B outperforms CodeLlama-34B on both MBPP and MBPP+.

Although EvalPlus makes HumanEval and MBPP far more robust, the problems in these benchmarks only exercise basic Python built-ins. They do not test them on other programming languages and do not test models' knowledge of other Python libraries. We address these limitations in the rest of this subsection with more comprehensive evaluations on code completion.

7.1.2 MultiPL-E: Multilingual Code Completion

About the benchmark MultiPL-E (Cassano et al., 2023b) uses a suite of lightweight, rule-based compilers to translate HumanEval from Python to 18 other programming languages. Thus MultiPL-E is a multi-language benchmark with the same problems translated to different languages.[26]

Hyperparameters We sample 50 completions per prompt at temperature 0.2 with top-p 0.95. This is how MultiPL-E results are reported on the BigCode Models Leaderboard (Ben Allal, 2023).

Results The results on MultiPL-E appear in Table 10. We make the following observations:

[26]MultiPL-E makes some small changes to the HumanEval prompts, and a few prompts fail to translate to certain languages. We refer the reader to Cassano et al. (2023b) for more information.

1. Across all size classes, there is no single model that is best at every language. Nevertheless, the StarCoder2 models perform well as described below.

2. Of the small models, StarCoder2-3B performs the best on 11/18 programming languages.

3. Of the medium models, DeepSeekCoder-6.7B performs best. StarCoder2-7B does better than CodeLlama-7B on most languages.

4. Of the large models, StarCoder2-15B does the best on 16/18 programming languages. CodeLlama-13B outperforms StarCoder2-15B on Go and Java.

5. StarCoder2-15B meets or exceeds the performance of CodeLlama-34B on 10/18 programming languages and DeepSeekCoder-33B on four lower-resource languages (D, Julia, Lua, and Perl).

7.1.3 DS-1000: Data Science Tasks in Python

About the benchmark DS-1000 (Lai et al., 2023) is a widely studied benchmark with 1,000 data science tasks in Python. Unlike the HumanEval and MBPP problems that only use the Python standard library, DS-1000 exercises seven widely used libraries, from Matplotlib to TensorFlow. Therefore, here we further adopt DS-1000 to evaluate the performance of Code LLMs in completing data science tasks with popular libraries.

Hyperparameters Following Lai et al. (2023), we use temperature 0.2 and top-p 0.95 to generate 40 samples per problem, and report mean pass@1.

Results Table 11 reports the results on DS-1000. We make the following observations:

1. StarCoder2-3B overall is the best-performing small model on DS-1000. Except for PyTorch and TensorFlow (where it is slightly worse than StableCode-3B), StarCoder2-3B achieves the best performance on all the other popular libraries.

2. StarCoder2-7B comes in second place out of the medium models, with a performance similar to DeepSeekCoder-6.7B.

3. StarCoder2-15B is the best-performing large model on DS-1000. It substantially outperforms both StarCoderBase-15B and CodeLlama-13B by large margins, and approaches the overall performance of CodeLlama-34B.

7.2 Code Fixing and Editing

While the above subsection has studied various code completion tasks, Code LLMs can be used in various other ways. In this subsection, we focus on studying their capabilities for fixing bugs or editing existing code.

7.2.1 HumanEvalFix: Fixing Bugs in Six Programming Languages

About the benchmark HumanEvalFix (Muennighoff et al., 2024a) is a benchmark that tests a model's ability to identify and fix bugs in code. The benchmark supports six programming languages shown in Figure 12. Since it is not a code completion benchmark, most base models do poorly on HumanEvalFix whereas instruction-tuned (Wei et al., 2022; Sanh et al., 2022; Muennighoff et al., 2022b; 2024b) models perform better. Thus, we consider the instruction-tuned variants of DeepSeekCoder and CodeLlama in our comparison (Guo et al., 2024; Rozière et al., 2023). We also compare with OctoCoder, which is an instruction-tuned version of the initial StarCoder using the CommitPackFT dataset (Muennighoff et al., 2024a; Zhuo et al., 2024; Longpre et al., 2023). We benchmarked the default HumanEvalFixTests subvariant; hence, there were no docstrings present to guide the model.

Table 11: *Performance of open-access models on DS-1000. Benchmarks are as follows. All models were evaluated at temperature 0.2 and top-p 0.95. Scores reflect mean pass@1 accuracy averaged over 40 samples.*

Format	Model	Matplotlib	NumPy	Pandas	PyTorch	SciPy	Scikit-Learn	TensorFlow	Overall
	# problems:	155	220	291	68	106	115	45	1,000
Completion	StarCoderBase-3B	32.1	16.8	5.3	9.2	13.2	10.5	17.2	14.2
Completion	StableCode-3B	42.5	24.5	**16.2**	**15.4**	13.5	20.2	**27.7**	22.7
Completion	DeepSeekCoder-1.3B	36.2	18.8	9.1	10.7	7.9	13.9	13.3	16.2
Completion	StarCoder2-3B	**45.5**	**27.7**	**16.2**	12.9	**15.8**	**30.8**	22.8	**25.0**
Completion	StarCoderBase-7B	38.0	23.0	8.2	13.1	13.7	24.5	14.6	19.1
Completion	DeepSeekCoder-6.7B	52.4	33.0	**20.0**	13.9	19.8	**29.7**	27.4	**28.9**
Completion	CodeLlama-7B	46.3	21.6	13.9	12.2	17.5	16.7	20.6	21.5
Completion	StarCoder2-7B	**53.6**	**33.3**	16.9	**16.2**	**20.6**	22.2	**31.9**	27.8
Completion	StarCoderBase-15B	47.0	27.1	10.1	**19.5**	21.7	**27.0**	20.5	23.8
Completion	CodeLlama-13B	49.0	27.2	17.4	12.9	15.6	24.0	24.8	25.1
Completion	StarCoder2-15B	**60.3**	**43.3**	**23.2**	11.0	**26.4**	26.0	**36.0**	**33.8**
Completion	DeepSeekCoder-33B	**56.1**	**49.6**	**25.8**	**36.8**	**36.8**	**40.0**	**46.7**	**40.2**
Completion	CodeLlama-34B	50.3	42.7	23.0	25.0	28.3	33.9	40.0	34.3

Table 12: *Pass@1 performance on HumanEvalFix. StarCoder2 and StarCoderBase are not instruction-tuned thus they are at a disadvantage compared to the other models which are all instruction-tuned.*

Model	Prompt	Python	JavaScript	Java	Go	C++	Rust	Avg.
StarCoderBase-15B	Instruct	12.6	16.8	18.9	12.5	11.2	0.6	12.1
StarCoderBase-15B	Commit	25.6	29.4	28.8	28.7	28.2	19.7	26.7
CodeLlama-13B-Instruct	Instruct	19.4	18.9	24.1	21.6	10.1	0.4	15.8
CodeLlama-34B-Instruct	Instruct	36.5	28.1	36.4	25.7	25.2	18.5	28.4
DeepSeekCoder-6.7B-Instruct	Instruct	44.9	**55.3**	**52.2**	42.9	37.9	19.5	**42.1**
DeepSeekCoder-33B-Instruct	Instruct	47.5	47.6	46.5	**52.0**	**48.0**	10.2	**42.1**
OctoCoder-15B	Instruct	30.4	28.4	30.6	30.2	26.1	16.5	27.0
StarCoder2-15B	Instruct	9.7	20.7	24.1	36.3	25.6	15.4	22.0
StarCoder2-15B	Issue	**48.6**	41.6	48.4	48.5	20.7	**24.2**	38.7

StarCoder2 issues format Although StarCoder2 is a base model, it is pretrained on GitHub issues and StackOverflow discussions using a special format (§5.3). We experiment with prompting the model to fix code bugs in the style of a discussion as follows:

```
<issue_start>username_0: instruction\n\n```buggy function```\nUpvotes: 100<issue_comment>
username_1: Sure, here is the fixed code.\n\n```function start
```

In this template, "instruction" is the HumanEvalFix instruction telling the model to fix the bug in the code, "buggy function" is the function with a subtle bug, and "function start" is the function header including imports. The generation of the model is stopped as soon as ``` is generated. The evaluation code is available via Ben Allal et al. (2022), and we denote this as the "Issue" prompt. We also benchmark StarCoder2 with the same basic "Instruct" prompt used in Muennighoff et al. (2024a).

Hyperparameters: Following (Muennighoff et al., 2024a), we use a temperature of 0.2 to estimate pass@1 with 20 samples.

Results Unlike the previous sections, we only evaluate StarCoder2-15B and primarily compare it to instruction-tuned models. The results are in Table 12 (with best-performing models highlighted in bold and second-best underscored), and we make the following conclusions:

1. The base models (StarCoder2-15B and StarCoderBase-15B) perform very poorly when given an instruction prompt, which motivates using a different prompt format.

2. Using the Issue prompt described above, StarCoder2-15B performs remarkable well as a base model. It outperforms the instruction-tuned CodeLlama models by a significant margin and nearly reaches the performance of the instruction-tuned DeepSeekCoder models.

3. Using the Issue prompt for StarCoder2-15B leads to a larger increase in performance than using the Commit prompt for StarCoderBase-15B. This indicates that pre-training on pull requests (StarCoder2) is a viable alternative to pre-training on commits (StarCoderBase).

4. Using the Issue prompt, StarCoder2-15B also outperforms all other open models presented in Muennighoff et al. (2024a).

5. StarCoder2-15B underperforms on C++ when using the Issue prompt, which hurts its overall performance. Our investigation shows that this is mainly because one-third of the code generated is incomplete, e.g., having an unexpected break immediately after the beginning of a `for` loop. Additional prompt engineering may be necessary to fix this. Thus, we still see value in instruction tuning StarCoder2 to further improve its usability in handling similar scenarios more effectively without prompt engineering. We leave the instruction tuning or even preference alignment (Christiano et al., 2017; Ethayarajh et al., 2024) of StarCoder2 to future work.

7.2.2 Code Editing

About the benchmark CanItEdit (Cassano et al., 2024) is a hand-crafted benchmark designed to evaluate model performance in Python code editing tasks. Each problem consists of a code snippet accompanied by an instruction of two types: *descriptive* or *lazy*. Descriptive instructions are systematic and provide detailed information, whereas lazy instructions are brief, direct, and mimic the typical instructions humans provide to code completion models. The goal is to modify the code according to the instruction; both lazy and descriptive instructions should lead to the same edit. The accuracy of each modification is assessed using a hidden test suite, and pass@1 is reported. The benchmark encompasses a variety of problems, from simple single-function, single-line edits to intricate multi-class problems requiring multiple-line edits in separate locations. Some tasks demand domain-specific knowledge like mathematics, and successful completion of a problem often requires the model to understand the connections between the components of the program. Listing 3 shows an abbreviated[27] sample problem from CanItEdit with its lazy instruction.

Listing 3: *Abbreviated sample problem from CanItEdit*

```
-class C4(nn.Module):
+class C8(nn.Module):
-    """Represents the C4 class of group theory,
+    """Represents the C8 class of group theory,
        where each element represents a discrete rotation."""

    def __init__(self):
        super().__init__()

    def elements(self):
        """Returns all the elements of this group"""
-       return torch.tensor([0., np.pi/2, np.pi, 3*np.pi/2])
+       d = np.pi / 4
+       return torch.tensor([0., d, d*2, d*3, d*4, d*5, d*6, d*7])
```

Code Editing Instruction: Edit the C4 class and its methods to represent the C8 group.

[27]The original problem includes additional methods to edit in the C4 class and a descriptive instruction.

Table 13: *Performance of instructional code editing on the CanItEdit benchmark (Cassano et al., 2024). The results for non-StarCoder2 models are from the benchmark paper.*

Model	Format	Descriptive Instructions	Lazy Instructions
		Pass@1	
StarCoderBase-3B	Commit	19.62	12.78
StarCoder2-3B	Issue	21.68	15.91
DeepSeekCoder-Instruct-1.3B	Instruct	**25.83**	**18.33**
StarCoder2-7B	Issue	35.23	18.55
CodeLlama-Instruct-7B	Instruct	33.89	27.04
StarCoderBase-7B	Commit	**40.64**	25.83
DeepSeekCoder-Instruct-6.7B	Instruct	33.89	**33.61**
CodeLlama-Instruct-13B	Instruct	28.33	20.19
OctoCoder-15B	Instruct	31.46	25.69
StarCoderBase-15B	Commit	38.24	26.38
StarCoder2-15B	Issue	**43.08**	**38.45**
CodeLlama-Instruct-34B	Instruct	35.0	26.76
DeepSeekCoder-Instruct-33B	Instruct	**53.06**	**43.89**

Hyperparameters We evaluate all sizes of StarCoder2 on the CanItEdit benchmark using the Issue prompt format (introduced in §7.2.1) and compare its performance with other models previously assessed on this benchmark. Following Cassano et al. (2024), we employ random sampling with a temperature of 0.2 and a top-p of 0.95, with 100 completions per problem.

Results The results appear in Table 13. As described in §7.2.1, we use an "Issue" prompt and "Commit" prompt for the StarCoder2 and StarCoderBase models since they are not instruction-tuned. For all the other models, we use instruction-tuned versions. From the table, we make the following observations:

1. Of the small models, StarCoder2-3B comes in second place behind DeepSeekCoder-Instruct-1.3B.
2. Of the medium models, StarCoder2-7B and DeepSeekCoder-Instruct-6.7B each performs best at descriptive and lazy instructions respectively.
3. StarCoder2-15B is the best-performing large model by a significant margin.
4. StarCoder2-15B outperforms CodeLlama-Instruct-34B as well.

These results give further evidence that the StarCoder2 "Issue" format is a viable alternative to the StarCoderBase "Commit" format.

7.3 Math Reasoning

About the benchmark We use the widely studied GSM8K benchmark (Cobbe et al., 2021), a set of middle-school math problems, to evaluate the mathematical reasoning capabilities of the models. We use the PAL approach proposed by Gao et al. (2023): the model is prompted to generate a Python program, which is executed to produce the answer to the problem.

Hyperparameters We evaluate models with greedy decoding in an 8-shot setting following Chowdhery et al. (2023).

Results The results on GSM8K with PAL appear in Table 14 and we make the following observations:

1. StableCode-3B is the best-performing small model. StarCoder2-3B is in second place.

Table 14: *8-shot accuracy on the GSM8K math-reasoning benchmark.*

Model	GSM8K (PAL)
StarCoderBase-3B	8.0
DeepSeekCoder-1.3B	12.6
StableCode-3B	**39.7**
StarCoder2-3B	27.7
StarCoderBase-7B	14.9
DeepSeekCoder-6.7B	**41.9**
CodeLlama-7B	27.0
StarCoder2-7B	40.4
StarCoderBase-15B	21.5
CodeLlama-13B	38.1
StarCoder2-15B	**65.1**
CodeLlama-34B	54.2
DeepSeekCoder-33B	**58.7**

2. StarCoder2-7B comes second place. Its performance is very close to the first-place model, which is DeepSeekCoder-6.7B, while substantially outperforming both CodeLlama-7B and StarCoderBase-7B.
3. StarCoder2-15B significantly outperforms all large models, including both CodeLlama-13B and StarCoderBase-15B.
4. In fact, StarCoder2-15B even outperforms CodeLlama-34B and DeepSeekCoder-33B which are more than twice its size.

7.4 CRUXEval: Code Reasoning, Understanding, and Execution

About the benchmark CRUXEval (Gu et al., 2024) is a two-part benchmark consisting of 800 samples designed to evaluate code reasoning, understanding, and execution. In the first task, CRUXEval-I, the model is asked to predict any input such that executing a given Python function on that input produces a given output. In the second task, CRUXEval-O, the model is asked to simulate the execution of a given function on an input and predict an output. Two samples are shown below in Listings 4 and 5. The functions and inputs of the benchmark were generated by CodeLlama-34B and then filtered to remove complicated functions such as those requiring complex arithmetic or a large number of execution steps.

Listing 4: *Sample CRUXEval Problem 1*

```
def f(string):
    string_x = string.rstrip("a")
    string = string_x.rstrip("e")
    return string

# output prediction, CRUXEval-O
assert f("xxxxaaee") == ??

# input prediction, CRUXEval-I
assert f(??) == "xxxxaa"
```

Listing 5: *Sample CRUXEval Problem 2*

```
def f(nums):
    count = len(nums)
    for i in range(-count+1, 0):
        nums.append(nums[i])
    return nums
# output prediction, CRUXEval-O
assert f([2, 6, 1, 3, 1]) == ??

# input prediction, CRUXEval-I
assert f(??) == [2, 6, 1, 3, 1, 6, 3, 6, 6]
```

Hyperparameters Following (Gu et al., 2024), we use temperature 0.2 to report pass@1 and temperature 0.8 to report pass@5, both using 10 samples.

Results We show the pass@1 and pass@5 scores for both tasks in our benchmark in Table 15. In terms of error and standard deviation, the original paper reports two sources of noise. First, the noise due to sampling from the language model for the given set of 800 candidates is around 0.2% for 10 samples. Second, the

Table 15: *Accuracy on the CRUXEval benchmark.*

Model	CRUXEval-I		CRUXEval-O	
	Pass@1	Pass@5	Pass@1	Pass@5
StarCoderBase-3B	27.1	43.7	27.4	40.9
DeepSeekCoder-1.3B	27.8	44.7	31.0	43.4
StableCode-3B	**33.5**	**53.3**	26.7	43.5
StarCoder2-3B	32.7	50.1	**34.2**	**48.4**
StarCoderBase-7B	29.7	47.3	32.2	44.9
CodeLlama-7B	35.9	52.9	34.2	48.4
DeepSeekCoder-6.7B	**41.9**	**62.7**	**43.5**	**54.8**
StarCoder2-7B	34.6	53.5	36.0	52.0
StarCoderBase-15B	31.3	49.2	34.2	47.1
CodeLlama-13B	42.5	62.0	39.7	53.9
StarCoder2-15B	**48.1**	**66.9**	**47.1**	**59.5**
CodeLlama-34B	**47.2**	**66.6**	42.4	55.9
DeepSeekCoder-33B	46.5	64.9	**48.6**	**61.6**

precise samples in the benchmark were chosen from a larger set of samples, and the noise from choosing which samples to include in the benchmark when using 800 samples is about 1.5%. We make the following observations:

1. StarCoder2-3B performs competitively with other small models. It slightly underperforms StableCode-3B on CRUXEval-I (but within the noise margin of error) but beats all other small models on CRUXEval-O.

2. For both tasks, StarCoder2-7B performs on par with CodeLlama-7B but lags significantly behind DeepSeekCoder-6.7B.

3. StarCoder2-15B is the best-performing large model. It surpasses CodeLlama-13B and drastically improves upon StarCoderBase-15B on both CRUXEval-I and CRUXEval-O.

4. StarCoder2-15B performs on par with the extra-large models. On CRUXEval-I, it outperforms both CodeLlama-34B and DeepSeekCoder-33B but within standard deviation. On CRUXEval-O, it significantly outperforms CodeLlama-34B and slightly underperforms DeepSeekCoder-33B.

7.5 Fill-in-the-Middle

About the benchmark StarCoder2 supports fill-in-the-middle (FIM), which is the ability to complete an arbitrary span of code conditioned on both text before and after the insertion point. We use the benchmark from Ben Allal et al. (2023), which tests the ability of models to fill in a single line of code in Python, JavaScript, and Java solutions to HumanEval.

Hyperparameters Following Ben Allal et al. (2023), we sample 20 completions per example at temperature 0.2 and top-p 0.95 and report the mean exact match, as done

Results The results appear in Table 16. We observe that StarCoder2-3B performs as well as StarCoderBase-15B on this FIM benchmark. Unfortunately, StarCoder2-15B underperforms on FIM. Due to an implementation bug, the FIM-rate was smaller than intended for most of the training.

Table 16: *Exact-match on FIM-task (Ben Allal et al., 2023). Due to an implementation bug, FIM was incorrect for most of the training of StarCoder2-15B. CodeLlama results are from Rozière et al. (2023).*

Model	Java	JavaScript	Python
StableCode-3B	63.7	73.3	59.1
StarCoder2-3B	75.0	73.0	59.1
StarCoder2-7B	81.1	77.5	61.1
CodeLlama-13B	80.0	85.0	74.5
StarCoderBase-15B	73	74	62
StarCoder2-15B	60.5	54.7	48.4

7.6 Repository-Level Code Completion Evaluation

Code completion in practice often occurs within the context of a repository rather than in isolated files. Leveraging repository-level context for code completion is thus essential for models to perform well in real-world scenarios. We evaluate models on repository-level code completion with two benchmarks: RepoBench (Liu et al., 2023b) and CrossCodeEval (Ding et al., 2023).

7.6.1 RepoBench

About the benchmark RepoBench (Liu et al., 2023b) is a live benchmark designed for evaluating code completion at the repository level, with a focus on next-line prediction. In this work, we use the latest version (v1.1) of RepoBench[28,29], which sources its data from GitHub repositories created from October 6th to December 31st, 2023, and takes steps to avoid data leakage by removing duplicates against The Stack v2. Our evaluation includes five levels—2k, 4k, 8k, 12k, and 16k—across three settings: `cross-file-first`, `cross-file-random`, and `in-file`, with each setting comprising 5,000 data points (1,000 per level). We report the average edit similarity, exact match, and CodeBLEU (Ren et al., 2020) scores for the three settings.

Hyperparameters Following prior work on Code LLMs (Chen et al., 2021), we set the generation temperature to 0.2 and the top-p sampling parameter to 0.95 for all models under evaluation. We constrained the models to generate a maximum of 128 new tokens per prompt, and the first non-comment line of the output was selected as the prediction. While StarCoder2 uses special tokens for repository-level training, we ensured uniformity in prompt construction across all models by following the official implementation in line with Liu et al. (2023b). The maximum token count for prompts was set to 15,800 by truncating excess cross-file context, except for StarCoderBase, which was constrained to 7,800 tokens due to its maximum context length limit of 8,192.

Results Table 17 showcases the performance of open-access models on RepoBench v1.1. We observe that:

1. StarCoder2, with repository-level training, consistently outperforms StarCoderBase, across all evaluated model sizes.
2. StarCoder2-3B demonstrates notable performance among the smaller models, ranking as the second-best one following StableCode-3B.
3. StarCoder2-7B achieves competitive performance closely matching that of CodeLlama-7B among the medium models, with DeepSeekCoder-6.7B achieving the leading performance metrics.
4. StarCoder2-15B not only outpaces CodeLlama-13B but also showcases comparable, and in some aspects superior, performance against the significantly larger CodeLlama-34B model.

[28] https://huggingface.co/datasets/tianyang/repobench_python_v1.1
[29] https://huggingface.co/datasets/tianyang/repobench_java_v1.1

Table 17: *Average exact match (EM), edit similarity (ES), and CodeBLEU (CB) scores for open-access base models on RepoBench v1.1 (Liu et al., 2023b).*

Model	Python			Java		
	EM	ES	CB	EM	ES	CB
StarCoderBase-3B	29.99	69.37	36.77	36.01	74.18	45.30
DeepSeekCoder-1.3B	31.02	70.07	37.88	37.75	75.66	46.69
StableCode-3B	**34.48**	**71.79**	**40.43**	**40.13**	**76.56**	**49.00**
StarCoder2-3B	32.47	71.19	39.25	38.46	76.53	47.96
StarCoderBase-7B	32.70	71.08	39.48	37.97	75.66	47.47
CodeLlama-7B	33.85	71.79	40.47	39.61	76.71	48.92
DeepSeekCoder-6.7B	**36.79**	**73.85**	**42.65**	**42.87**	**78.93**	**51.69**
StarCoder2-7B	33.72	72.07	40.34	39.84	77.23	48.96
StarCoderBase-15B	33.51	71.64	40.39	39.34	76.24	48.36
CodeLlama-13B	35.50	72.98	42.02	41.27	77.57	50.26
StarCoder2-15B	**36.99**	**74.08**	**43.25**	**42.57**	**79.05**	**51.45**
CodeLlama-34B	37.22	73.77	43.38	42.35	78.22	50.99
DeepSeekCoder-33B	**39.25**	**75.20**	**45.21**	**44.59**	**79.92**	**52.70**

7.6.2 CrossCodeEval

About the benchmark CrossCodeEval (Ding et al., 2023) is a diverse and multilingual benchmark designed for repository-level code completion. It was constructed from a wide range of real-world, open-sourced, permissively licensed repositories in four popular programming languages: Python, Java, TypeScript, and C#. Through careful static analysis methods, CrossCodeEval **strictly** requires cross-file context for accurate code completion. We report results in both Code Match (Edit Similarity) and Identifier Match (F1 Score) following the definitions in Ding et al. (2023) in all four languages.

Hyperparameters We use a max sequence length of 16k for all models except for StarCoderBase, which only supports 8k. In line with Ding et al. (2023), we use the retrieve-and-generate (RG) method with OpenAI's ada embedding, which was found to perform well in their study. To optimize the usage of the extended 16k context, we retrieve a maximum of 100 code segments, each comprising its file path and 10 lines of code. The maximum cross-file context was set to 12,800 tokens and the max generation token is 50 tokens following. Consistent with Ding et al. (2023), we use the uniform prompt formatting in the original implementation, with a temperature of 0.2 and top-p of 0.95 for all model generations.

Results Table 18 presents the evaluation results. We found that:

1. Across almost all dimensions, including model sizes, programming languages, and metrics, StarCoder2 consistently outperforms StarCoderBase. This enhancement could likely be attributed to better pre-training with increased context length and repository-level objectives (Section 5.1).

2. StarCoder2-15B achieves the state-of-the-art performance compared to models of similar sizes. For certain languages like Java and C#, the performance is better even than models with 2x capacity.

3. The analysis also reveals significant performance variances in different languages for the same model, similar to the findings in MultiPL-E (§7.1.2). While a model can be strong overall, achieving uniformly high performance across all programming languages remains challenging, e.g., StarCoder2-15B is behind on TypeScript while StableCode-3B in C# and DeepSeekCoder-34B in Java. The disparity calls for future research on building models that can achieve high performance across diverse range of languages in different settings.

Table 18: *CrossCodeEval (Ding et al., 2023) evaluation results. We report Code Match (Edit Similarity) and Identifier Match (F1) results for four languages.*

Model	Python		Java		TypeScript		C#	
	Code ES	ID F1	Code ES	ID F1	Code ES	ID F1	Code ES	ID F1
StarCoderBase-3B	69.47	62.56	66.43	59.77	41.42	35.26	70.11	53.15
DeepSeekCoder-1.3B	72.41	66.76	65.92	59.93	63.59	56.41	**70.98**	54.84
StableCode-3B	**76.00**	**70.75**	**73.19**	**67.93**	**65.61**	**59.61**	61.70	48.98
StarCoder2-3B	73.01	67.85	66.31	61.06	38.79	35.17	70.86	**55.42**
StarCoderBase-7B	72.24	65.40	69.91	64.12	44.21	39.77	71.93	55.98
DeepSeekCoder-6.7B	**77.43**	**73.16**	70.60	**66.28**	**69.08**	**63.61**	**74.84**	**62.29**
CodeLlama-7B	74.52	69.11	**71.49**	65.99	65.96	59.46	71.41	56.66
StarCoder2-7B	74.52	68.81	70.75	65.27	43.19	38.84	72.73	57.69
StarCoderBase-15B	73.43	66.74	70.58	64.66	45.24	40.47	71.77	55.71
CodeLlama-13B	75.88	70.97	73.08	68.29	**67.88**	**61.46**	72.73	59.62
StarCoder2-15B	**78.72**	**74.27**	**74.92**	**70.45**	48.63	43.78	**75.38**	**62.14**
CodeLlama-34B	76.34	71.36	**74.30**	**69.45**	68.98	63.19	73.96	60.07
DeepSeekCoder-33B	**78.78**	**74.51**	73.41	69.02	**70.31**	**65.14**	**75.04**	**63.03**

Table 19: *Performance on the "Asleep at the Keyboard" benchmark.*

Model	Valid (↑)	Insecure (↓)
StarCoderBase-3B	910/1000 (91.0%)	224/910 (24.6%)
DeepSeekCoder-1.3B	893/1000 (89.3%)	290/893 (32.5%)
StarCoder2-3B	**925/1000 (92.5%)**	**113/900 (12.2%)**
StarCoderBase-7B	916/1000 (91.6%)	243/916 (26.5%)
CodeLlama-7B	900/1000 (90.0%)	**195/900 (21.7%)**
DeepSeekCoder-6.7B	**921/1000 (92.1%)**	315/921 (34.2%)
StarCoder2-7B	912/1000 (91.2%)	363/926 (39.8%)
StarCoderBase-15B	**933/1000 (93.3%)**	332/933 (35.6%)
CodeLlama-13B	903/1000 (90.3%)	**273/903 (30.2%)**
StarCoder2-15B	898/1000 (89.8%)	352/898 (39.2%)

7.7 "Asleep at the Keyboard" Security Benchmark

About the benchmark "Asleep at the Keyboard" is a benchmark designed for assessing security vulnerabilities in code generation (Pearce et al., 2022). Similar to Li et al. (2023), we focus on the subset of tasks amenable to automated evaluation, which is the *Diversity of Weakness* problems. These cover 18 diverse vulnerability classes from the MITRE Common Weakness Enumeration (CWE) taxonomy, with scenarios drawn from the 2021 CWE Top 25 Most Dangerous Software Weaknesses list published by MITRE. The problems have 23 scenarios in C and 17 scenarios in Python.

Hyperparameters Following Li et al. (2023), we set the temperature to 0.2 and top-p to 0.95. Each model generates 25 samples per scenario, resulting in a total of 1,000 completions.

Results We report results of selected models in Table 19. Column **Valid** gives the percentage of solutions that were syntactically valid, and Column **Insecure** shows the percentage of valid solutions that include the vulnerability the scenario tests for. From the table, we draw the following conclusions:

1. StarCoder2 generates comparable numbers of valid programs to StarCoderBase, CodeLlama, and DeepSeekCoder. Both StarCoderBase and StarCoder2 models achieve around 90% valid program

rate. However, after some manual inspection, we notice that StarCoder2 tends to generate more functionally correct code than StarCoderBase. The observation is aligned with the evaluation in previous sections.

2. Except for StarCoder2-3B, StarCoder2-7B and StarCoder2-15B have the highest insecure program rate among the models having similar parameters. The high insecure rate might be a side-effect of its higher rate of valid and functionally correct completions. These generated programs are more likely to be exposed to potential vulnerabilities, as suggested by Bhatt et al. (2023). Notably, Li et al. (2023) find that `code-cushman-001`, the initial model used in commercialized Copilot, has an insecure rate beyond 40%.

7.8 Measuring Harmful Generations

About the benchmarks Bias in Open-ended Language Generation Dataset (BOLD) (Dhamala et al., 2021) is a dataset of 23,679 prompts that measure fairness across Religious Ideology, Procession, Race, Gender, and Political Ideology. We use the Regard classifier by Sheng et al. (2019) and average the classifier scores across each category.

WinoBias (Zhao et al., 2018) measures bias towards gender stereotypes across professions. When given a sentence regarding a profession, the gender is masked, and a language model is used to generate the replaced masked token and the perplexity is used to calculate the bias towards male and female pronouns.

Hurtful Sentence Completion in English Language Models (HONEST) (Nozza et al., 2021) is a template-based corpus to assess the hurtfulness of sentence completions generated by the language models. To calculate the HONEST score, we check whether each generated completion has any terms contained in each of the categories within `Hurtlex`[30].

RealToxicityPrompts (Gehman et al., 2020) consists of 100,000 naturally occurring, sentence-level prompts, which are extracted from the large web corpus of English text. They can be used to evaluate the risk of neural toxic degeneration in the language models. We use a 10,000 subset to perform the evaluation. We use the classifier by Vidgen et al. (2021) to detect toxicity and report the average probability of the detected toxic output as our toxicity score.

Hyperparameters For each prompt in BOLD and RealToxicityPrompts, we generate one completion with up to 50 additional tokens. On HONEST, we generate 5 completions for each sample with up to 50 additional tokens.

Results The results for BOLD, WinoBias, HONEST, and RealToxicityPrompts are presented in Tables 20, 21, 22, and 23, respectively. The tables suggest that our models LLMs that we consider produce roughly the same amount of harmful content, and based on Li et al. (2023), LLMs trained primarily on code produce less harmful content than LLMs trained on general web text.

8 Search Index and Attribution Tools

Following the standard set by Li et al. (2023) we build another suite of data inspection, attribution, and search tools. The NLP community has recognized the need for data inspection and has begun producing computational documentation artifacts to complement static data descriptions (Piktus et al., 2023b; Marone & Van Durme, 2023; Piktus et al., 2023a; Akiki et al., 2023, among others). Open science and open data go beyond releasing dumps of datasets.

Membership checking tools This work collects and constructs a dataset 4 times larger than that used in StarCoderBase. Compared to the initial version of The Stack, the version here contains many additional non-code sources (see Table 4). As data sizes increase, it becomes even more important to construct tools that allow for accessible and efficient data inspection. We update the "Am I in the Stack" tool with repositories in

[30] https://github.com/valeriobasile/hurtlex

Table 20: *BOLD evaluations of open source code models.*

Model	Category	Negative Score	Neutral Score	Other Score	Positive Score
StarCoder2-3B	Religious Ideology	0.16	0.33	0.13	0.38
	Profession	0.07	0.6	0.06	0.27
	Race	0.05	0.5	0.05	0.5
	Gender	0.05	0.48	0.05	0.43
	Political Ideology	0.3	0.29	0.18	0.23
StarCoderBase-3B	Religious Ideology	0.12	0.32	0.12	0.45
	Profession	0.07	0.58	0.06	0.3
	Race	0.04	0.44	0.05	0.47
	Gender	0.04	0.35	0.05	0.55
	Political Ideology	0.3	0.27	0.18	0.25
StableCode-3B	Religious Ideology	0.18	0.25	0.16	0.41
	Profession	0.08	0.57	0.06	0.28
	Race	0.07	0.4	0.06	0.46
	Gender	0.05	0.36	0.06	0.53
	Political Ideology	0.32	0.27	0.18	0.25
StarCoder2-7B	Religious Ideology	0.19	0.81	0.03	0.13
	Profession	0.08	0.52	0.07	0.33
	Race	0.06	0.4	0.07	0.47
	Gender	0.06	0.37	0.07	0.5
	Political Ideology	0.33	0.22	0.21	0.24
StarCoderBase-7B	Religious Ideology	0.16	0.28	0.13	0.43
	Profession	0.07	0.56	0.06	0.31
	Race	0.05	0.41	0.06	0.48
	Gender	0.04	0.33	0.06	0.57
	Political Ideology	0.33	0.23	0.19	0.25
CodeLlama-7B	Religious Ideology	0.16	0.27	0.14	0.43
	Profession	0.07	0.58	0.06	0.3
	Race	0.06	0.42	0.06	0.46
	Gender	0.05	0.38	0.06	0.5
	Political Ideology	0.3	0.28	0.19	0.24
DeepSeekCoder-6.7B	Religious Ideology	0.15	0.33	0.13	0.39
	Profession	0.07	0.61	0.06	0.27
	Race	0.05	0.46	0.05	0.44
	Gender	0.04	0.34	0.06	0.56
	Political Ideology	0.3	0.28	0.19	0.23
StarCoder2-15B	Religious Ideology	0.21	0.22	0.16	0.42
	Profession	0.09	0.51	0.07	0.33
	Race	0.07	0.39	0.07	0.47
	Gender	0.05	0.36	0.07	0.53
	Political Ideology	0.25	0.02	0.1	0.09
StarCoderBase-15B	Religious Ideology	0.16	0.31	0.13	0.41
	Profession	0.07	0.61	0.06	0.26
	Race	0.06	0.46	0.06	0.43
	Gender	0.04	0.38	0.06	0.53
	Political Ideology	0.32	0.28	0.19	0.22
CodeLlama-13B	Religious Ideology	0.17	0.24	0.14	0.45
	Profession	0.07	0.54	0.06	0.33
	Race	0.07	0.36	0.07	0.5
	Gender	0.05	0.35	0.06	0.53
	Political Ideology	0.3	0.23	0.19	0.28

new dataset.[31] This tool allows for data inspection at the username and repository level. Marone & Van Durme (2023) recommend releasing a documentation artifact called a *Data Portrait* to support lightweight membership inspection. We implement one using Bloom filters to enable matching on file contents, crucially including the non-code sources like documentation, textbooks, and papers.[32] These prose data sources may

[31] https://huggingface.co/spaces/bigcode/in-the-stack
[32] https://stack-v2.dataportraits.org

Table 21: *WinoBias evaluations of open source code models.*

Model	Male	Female	Average
StarCoder2-3B	0.33	-0.33	0.27
StarCoderBase-3B	0.42	-0.42	0.28
StableCode-3B	0.44	-0.44	0.39
StarCoder2-7B	0.45	-0.45	0.34
StarCoderBase-7B	0.51	-0.51	0.31
CodeLlama-7B	0.37	-0.37	0.38
DeepSeekCoder-6.7B	0.41	-0.41	0.34
StarCoder2-15B	0.36	-0.36	0.38
StarCoderBase-15B	0.55	-0.55	0.35
CodeLlama-13B	0.36	-0.36	0.37

Table 22: *HONEST evaluations.*

Model	Score
StarCoder2-3B	0.11
StarCoderBase-3B	0.11
StableCode-3B	0.09
StarCoder2-7B	0.1
StarCoderBase-7B	0.11
CodeLlama-7B	0.11
DeepSeekCoder-6.7B	0.1
StarCoder2-15B	0.11
StarCoderBase-15B	0.1
CodeLlama-13B	0.1

Table 23: *Toxicity score evaluation of open source code models.*

Model	Toxicity Score
StarCoder2-3B	0.05
StarCoderBase-3B	0.04
StableCode-3B	0.05
StarCoder2-7B	0.08
StarCoderBase-7B	0.04
CodeLlama-7B	0.04
DeepSeekCoder-6.7B	0.05
StarCoder2-15B	0.05
StarCoderBase-15B	0.04
CodeLlama-13B	0.04

describe algorithms or solutions not present elsewhere. Content creators can use our system as a simple "no code" inspection tool to check if their material occurs verbatim in our data. It also enables a rapid first-pass attribution check for coding tools built on our models.[33] This system takes about 70GB, substantially smaller than the data, but provides only exact matches for long strings. If necessary, users can use the full search index for additional analysis.

Search index The preceding tools provide lightweight data inspection. However, it may be necessary to perform full-text searches that support fuzzy matching and retrieval. Following StarCoder1 (Li et al., 2023), we build an Elasticsearch index on the source code subset of The Stack v2 and make it available at https://huggingface.co/spaces/bigcode/search-v2.

9 Social Impact and Limitations

Social impact and limitations have already been documented in the BigCode project (Kocetkov et al., 2023; Ben Allal et al., 2023; Li et al., 2023; BigCode collaboration et al., 2023). In the following sections, we cover our project approach towards the responsible development of large language models for code and highlight some more recent advances.

[33] https://github.com/huggingface/llm-vscode

9.1 Project Approach

Open-science StarCoder2 is the output of a community research project. The project is conducted in the spirit of Open Science (Woelfle et al., 2011; Mendez et al., 2020), focused on the responsible development and use of Code LLMs. Through open-governance practices, priority in decision-making has always yielded to the more responsible option, even if this meant introducing limitations that might impact adoption or future research (BigCode collaboration et al., 2023).

Ethical data sourcing Significant efforts from the BigCode community went into the careful curation, validation, decontamination, malware removal, license filtering, opt-out process, PII removal, structuring, packaging, hosting, licensing, and the publishing of a Dataset Card (Project, 2024) for the data used to train StarCoder2. Full transparency has been provided about the data used for training StarCoder2. A significant portion of the training dataset was sourced under license from Software Heritage (Software Heritage, 2024a).

Accelerating research BigCode's open approach to scientific collaboration (BigCode collaboration et al., 2023), open access model distribution and licensing (BigCode Project, 2023a; Malfa et al., 2023), and openness and disclosures of training data, architectures, and development are essential for the research community to have access to powerful, truly open LLMs, helping to accelerate future research (Groeneveld et al., 2024; Xu et al., 2024; Soldaini et al., 2024; Singh et al., 2024; Üstün et al., 2024; Luukkonen et al., 2023; Woelfle et al., 2011).

Open, but responsible The BigCode Open RAIL-M license (BigCode Project, 2023a) contains important use restrictions and is accompanied by an FAQ to help guide the responsible deployment and use of the model by downstream users (BigCode Project, 2023b).

Community of practice BigCode is very much a community of practice, with over 1,200 multi-disciplinary members from more than 60 countries working towards the responsible development of large language models for code (Sholler et al., 2019; Kocetkov et al., 2023; Ben Allal et al., 2023; Li et al., 2023; Muennighoff et al., 2024a; Zhuo et al., 2024). Of these members, 417 were active in the BigCode community collaboration tools within the period 27 October 2023 through 24 February 2024, the period aligning with StarCoder2 development. There has also been considerable downstream adoption of BigCode outputs, with millions of downloads collectively reported via the Hugging Face API (BigCode, 2024).

Auditable The StarCoder2 model, pre-training dataset, and supporting artifacts are easily accessible and available to anyone who wishes to conduct an independent audit (Solaiman, 2023; Mökander et al., 2023; BigCode collaboration et al., 2023).

9.2 Advancements in Code LLMs

Governance Card The BigCode Governance Card (BigCode collaboration et al., 2023) serves as an overview of the different mechanisms and areas of governance in the BigCode project. It aims to support transparency by providing relevant information about choices that were made during the project to the broader public and to serve as an example of intentional governance (Sholler et al., 2019) of an open research project that future endeavors can leverage to shape their own approach. The first section, Project Structure, covers the project organization, its stated goals and values, its internal decision processes, and its funding and resources. The second section, Data and Model Governance, covers decisions relating to the questions of data subject consent, privacy, and model release.

Archival of software metadata: Software metadata is vital for the classification, curation, and sharing of free and open-source software (FOSS). The source code landscape is very diverse. By generating linked data and referencing source code contributions within the Software Heritage archive from the global community of developers and scientists (Heritage, 2024), there is potential to enable a more ethical data supply chain for training LLMs (Cosmo & Zacchiroli, 2017; Abramatic et al., 2018).

Acceptable ML use: On October 19, 2023, Software Heritage published a statement that defines the acceptable machine learning use of the Software Heritage archive. This is a significant milestone that opens the door for more responsible data sourcing and licensing of AI training data (Software Heritage, 2023).

SoftWare Hash IDentifiers (SWHID): Software Heritage provides the SWHID unique identifiers, intrinsically bound to the software components, and that need no central registry, to ensure that a resilient web of knowledge can be built on top of the Software Heritage archive (The SWHID Specification Project, 2024). This can also be used by downstream developers to support efforts for those companies that prioritize a "software bill of materials" (SBOM) as a key building block in software security and software supply chain transparency and risk management (Cybersecurity & Infrastructure Security Agency, 2024; Mirakhorli et al., 2024), for example by including the SWHIDs in the SBOM, alongside other relevant information such as component names, versions, licenses, and source locations.

9.3 Challenges and Risks

Openness and safety risks Solaiman (2023) explains how the degree of openness in the LLM development process is connected to the potential risks associated with a model release. When systems are developed in a fully closed manner, it is more likely for power to become concentrated among high-resourced organizations, and the small development team may not fully comprehend the impact and long-term consequences of the model being deployed. In addition, closed-development systems are often less auditable by external experts and can impede scientific progress since researchers cannot build upon each other's work. On the other hand, fully open development allows for community research, democratizes access to the models, and enables audits throughout the whole development process. However, without appropriate guardrails, open LLM development poses a higher risk of misuse, as increased model access also increases the likelihood of harm caused by the model. Even though a released API can be shut down, once the model weights are released, it is nearly impossible to retract them. Discussing and implementing responsible AI practices has, therefore, been front and center during the development of our project's LLMs.

Privacy compliant generated code It is difficult to correctly identify and classify the different types of PII so that personal data processing, transformations, and flows through code can be evaluated (Tang et al., 2023). Where privacy-relevant methods are invoked in generated code, checking for PII leaks to the internet, use of encrypted data and anonymous IDs, will be necessary (Tang & Østvold, 2024). Downstream users are advised to implement additional PII scanning, filtering, cleansing, and mitigation to ensure compliance with their intended use cases (Yang et al., 2023; Albalak et al., 2024).

Security As with any open scientific research that provides open access to model weights, hyper-parameters, data processing code, training code, training data, and documentation, any actor can run or fine-tune the optimized model with very low computing costs (Governance AI, 2024). Even with the use restrictions set forth within the BigCode Open RAIL-M license, this will not prevent bad actors with malicious intent from attempting to cause harm (Mozes et al., 2023). For example, code LLMs with API access could be used to create sophisticated polymorphic malware (CrowdStrike, 2024) that would be highly evasive to security products that rely on signature-based detection and will be able to bypass measures such as Anti-Malware Scanning Interface (AMSI) as it eventually executes and runs code (CyberArk, 2024; Gupta et al., 2023).

Societal bias As has been previously established in evaluations of coding models, code LLMs can generate code with a structure that reflects stereotypes about gender, race, emotion, class, the structure of names, and other characteristics (Chen et al., 2021; Zhuo et al., 2023a). Further evaluation and guardrail mitigations are required in the context of downstream use cases (Huang et al., 2023; Dong et al., 2024).

Representation bias As discussed in previous sections, there is a lot more data in the training dataset for popular programming languages like Python and Java than for niche languages like Haskell and Fortran. As such, the model performs better on such high-resource languages, which may reinforce the preference of developers towards using such languages. Fortunately, there's much ongoing research on how to improve the performance of Code LLMs on low-resource languages (Cassano et al., 2023a; Zhuo et al., 2023b). Furthermore,

the predominant natural language in source code and other datasets used is English although other languages are also present. As such, the model can generate code snippets provided some non-English context, but the generated code is not guaranteed to work as intended or equally as well for all languages. This could limit the model's fairness and effectiveness across different coding tasks and environments (Alyafeai et al., 2024).

Traceability Using the SWHID to trace software components is not an easy task and will challenge most if not all, downstream developers. Future development and advancement of tools that make it easier to trace software components will be necessary to enable more transparent and responsible data supply chains (Cosmo et al., 2020).

Job augmentation vs. automation Code LLMs serve as powerful foundation models that can be fine-tuned to generate high-quality code, documentation, unit tests, text summaries, automation workflows, and more. Chen et al. (2023) find a positive correlation between occupation exposure and wage levels/experience premiums, suggesting higher-paying and experience-intensive jobs may face greater displacement risks from LLM-powered software. Goldman Sachs (2024) suggest that AI has the potential to automate 25% of labor tasks in advanced economies and 10 – 20% in emerging economies, however, they also state that "those fears should be counterbalanced, since AI has the potential to create new job tasks or categories requiring specialized human expertise". Autor et al. (2022) reports that "Roughly 60% of employment in 2018 is found in job titles that did not exist in 1940." and that "augmentation innovations boost occupational labor demand, while automation innovations erode it". Results from the task-based analysis in (World Economic Forum, 2024) reveal that jobs with the highest potential for automation of tasks by LLMs emphasize routine and repetitive procedures and do not require a high degree of interpersonal communication. Jobs with the highest potential for augmentation by LLMs emphasize critical thinking and complex problem-solving skills, especially those in science, technology, engineering, and mathematics (STEM) fields. Ziegler et al. (2024) reports the benefits of receiving AI suggestions while coding span the full range of typically investigated aspects of productivity, such as task time, product quality, cognitive load, enjoyment, and learning. In (Peng et al., 2023), a two-year collaboration between Google Core and Google Research (Brain Team), they find that of the 10k+ Google-internal developers using the code completion setup in their IDE, they measured user's code acceptance rate of 25-34%. Yahoo Finance (2024) announced ServiceNow, Inc. (NYSE: NOW) 2024 Q4 Earnings with coverage that the ServiceNow platform Now Assist skills using text-to-code (ServiceNow, 2024b) and text-to-workflow (ServiceNow, 2024a) LLMs (based on StarCoder), augment and increased developer productivity and speed of innovation by 52%.

10 Conclusion

We introduced StarCoder2, a family of LLMs designed for code generation, along with The Stack v2, the largest pre-training corpus for Code LLMs built on the foundations of the Software Heritage archive. The Stack v2 is ten times larger than its predecessor, yielding a raw dataset of 67.5 TB. Through extensive cleaning, filtering, and subsampling of the source code, along with the incorporation of other high-quality code-related datasets, we created a training set of approximately 3TB (900B+ tokens). Leveraging this new dataset, we trained StarCoder2 models with 3B, 7B, and 15B parameters. Our extensive Code LLM evaluations, assessing code completion, editing, and reasoning capabilities, revealed that StarCoder2-3B and StarCoder2-15B are state-of-the-art models within their respective size classes. By not only releasing the model weights but also ensuring complete transparency regarding the training data, we hope to increase trust in the developed models and empower other engineering teams and scientists to build upon our efforts.

11 Acknowledgements

This work was made possible by Software Heritage, the great library of source code: https://www.softwareheritage.org, and all the developers and scientists that contribute to the open source archives.

We thank Joydeep Biswas (UT Austin), Northeastern Research Computing, and NCSA Delta for providing computing resources used for evaluation. Carolyn Jane Anderson and Arjun Guha were partially sponsored by the U.S. National Science Foundation awards SES-2326173 and SES-2326174. Jiawei Liu, Yuxiang Wei,

and Lingming Zhang were partially sponsored by the U.S. National Science Foundation award CCF-2131943. Federico Cassano was partly sponsored by Roblox.

We thank Jenny Hui, ServiceNow, for her leadership in executing the StarCoder2 Research Collaboration Agreement between ServiceNow, Hugging Face, and NVIDIA to enable the training of all 3 models.

We thank the extended members of the BigCode community for the ongoing support and for their downstream contributions back to the community.

We also thank Hessie Jones and the Privacy Protection Collab that shared insights and lessons learned from their work in Defining Personal Information and the Remediation Framework during early exploration and consideration of PII redaction.

Evgenii Zheltonozhskii is supported by the Adams Fellowships Program of the Israel Academy of Sciences and Humanities.

References

Jean-François Abramatic, Roberto Di Cosmo, and Stefano Zacchiroli. Building the universal archive of source code. *Communications of the ACM*, 61(10):29–31, 2018. doi: 10.1145/3183558. URL https://cacm.acm.org/magazines/2018/10/231366-building-the-universal-archive-of-source-code/fulltext. (cited on pp. 3 and 37)

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.298. URL https://aclanthology.org/2023.emnlp-main.298. (cited on p. 20)

Christopher Akiki, Giada Pistilli, Margot Mieskes, Matthias Gallé, Thomas Wolf, Suzana Ilic, and Yacine Jernite. BigScience: a case study in the social construction of a multilingual large language model. In *Workshop on Broadening Research Collaborations 2022*, 2022. URL https://openreview.net/forum?id=2e346l2PPOm. (cited on p. 2)

Christopher Akiki, Odunayo Ogundepo, Aleksandra Piktus, Xinyu Zhang, Akintunde Oladipo, Jimmy Lin, and Martin Potthast. Spacerini: Plug-and-play search engines with pyserini and Hugging Face. In Yansong Feng and Els Lefever (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 140–148, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-demo.12. URL https://aclanthology.org/2023.emnlp-demo.12. (cited on p. 34)

Alon Albalak, Yanai Elazar, Sang Michael Xie, Shayne Longpre, Nathan Lambert, Xinyi Wang, Niklas Muennighoff, Bairu Hou, Liangming Pan, Haewon Jeong, Colin Raffel, Shiyu Chang, Tatsunori Hashimoto, and William Yang Wang. A survey on data selection for language models. *arXiv preprint*, February 2024. URL https://arxiv.org/abs/2402.16827. (cited on p. 38)

Zaid Alyafeai, Khalid Almubarak, Ahmed Ashraf, Deema Alnuhait, Saied Alshahrani, Gubran A. Q. Abdulrahman, Gamil Ahmed, Qais Gawah, Zead Saleh, Mustafa Ghaleb, Yousef Ali, and Maged S. Al-Shaibani. CIDAR: culturally relevant instruction dataset for Arabic. *arXiv preprint*, February 2024. URL https://arxiv.org/abs/2402.03177. (cited on p. 39)

Naveen Arivazhagan, Ankur Bapna, Orhan Firat, Dmitry Lepikhin, Melvin Johnson, Maxim Krikun, Mia Xu Chen, Yuan Cao, George Foster, Colin Cherry, Wolfgang Macherey, Zhifeng Chen, and Yonghui Wu. Massively multilingual neural machine translation in the wild: Findings and challenges. *arXiv preprint*, July 2019. URL https://arxiv.org/abs/1907.05019. (cited on p. 14)

Arxiv, 2024. URL https://info.arxiv.org/help/bulk_data_s3.html. (cited on p. 12)

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint*, August 2021. URL https://arxiv.org/abs/2108.07732. (cited on pp. 3 and 23)

David Autor, Caroline Chin, Anna M Salomons, and Bryan Seegmiller. New frontiers: The origins and content of new work, 1940–2018. Technical Report 30389, National Bureau of Economic Research, August 2022. URL http://www.nber.org/papers/w30389. (cited on p. 39)

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen Marcus McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=4WnqRR915j. (cited on p. 12)

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. *arXiv preprint*, July 2022. URL https://arxiv.org/abs/2207.14255. (cited on p. 16)

Loubna Ben Allal. Big code models leaderboard, 2023. URL https://huggingface.co/spaces/bigcode/bigcode-models-leaderboard. (cited on p. 24)

Loubna Ben Allal, Niklas Muennighoff, Logesh Kumar Umapathi, Ben Lipkin, and Leandro von Werra. A framework for the evaluation of code generation models. https://github.com/bigcode-project/bigcode-evaluation-harness, 2022. (cited on p. 26)

Loubna Ben Allal, Raymond Li, Denis Kocetkov, Chenghao Mou, Christopher Akiki, Carlos Munoz Ferrandis, Niklas Muennighoff, Mayank Mishra, Alex Gu, Manan Dey, Logesh Kumar Umapathi, Carolyn Jane Anderson, Yangtian Zi, Joel Lamy Poirier, Hailey Schoelkopf, Sergey Troshin, Dmitry Abulkhanov, Manuel Romero, Michael Lappert, Francesco De Toni, Bernardo García del Río, Qian Liu, Shamik Bose, Urvashi Bhattacharyya, Terry Yue Zhuo, Ian Yu, Paulo Villegas, Marco Zocca, Sourab Mangrulkar, David Lansky, Huu Nguyen, Danish Contractor, Luis Villa, Jia Li, Dzmitry Bahdanau, Yacine Jernite, Sean Hughes, Daniel Fried, Arjun Guha, Harm de Vries, and Leandro von Werra. SantaCoder: don't reach for the stars! *arXiv preprint*, August 2023. URL https://arxiv.org/abs/2301.03988. (cited on pp. 2, 13, 30, 31, 36, and 37)

Manish Bhatt, Sahana Chennabasappa, Cyrus Nikolaidis, Shengye Wan, Ivan Evtimov, Dominik Gabi, Daniel Song, Faizan Ahmad, Cornelius Aschermann, Lorenzo Fontana, Sasha Frolov, Ravi Prakash Giri, Dhaval Kapil, Yiannis Kozyrakis, David LeBlanc, James Milazzo, Aleksandar Straumann, Gabriel Synnaeve, Varun Vontimitta, Spencer Whitman, and Joshua Saxe. Purple llama CyberSecEval: A secure coding benchmark for language models. *arXiv preprint*, December 2023. URL https://arxiv.org/abs/2312.04724. (cited on p. 34)

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, Usvsn Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. Pythia: A suite for analyzing large language models across training and scaling. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 2397–2430. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/biderman23a.html. (cited on p. 2)

BigCode. Models by BigCode on Hugging Face, 2024. URL https://huggingface.co/api/models?author=bigcode&expand[]=downloadsAllTime. Accessed: 2024. (cited on p. 37)

BigCode collaboration, Sean Hughes, Harm de Vries, Jennifer Robinson, Carlos Muñoz Ferrandis, Loubna Ben Allal, Leandro von Werra, Jennifer Ding, Sebastien Paquet, and Yacine Jernite. The BigCode project governance card. *arXiv preprint*, December 2023. URL https://arxiv.org/abs/2312.03872. (cited on pp. 2, 36, and 37)

BigCode Project. Bigcode model license agreement, 2023a. URL https://huggingface.co/spaces/bigcode/bigcode-model-license-agreement. Accessed: 2023. (cited on p. 37)

BigCode Project. BigCode open RAIL: Responsible AI licensing framework, 2023b. URL https://www.bigcode-project.org/docs/pages/bigcode-openrail/. Accessed: 2023. (cited on p. 37)

BigScience Workshop. BLOOM (revision 4ab0472), 2022. URL https://huggingface.co/bigscience/bloom. (cited on p. 2)

Blue Oak Council, 2024. URL https://blueoakcouncil.org/list. (cited on p. 4)

Andrei Z. Broder. Identifying and filtering near-duplicate documents. In *Annual symposium on combinatorial pattern matching*, pp. 1–10. Springer, 2000. URL https://link.springer.com/chapter/10.1007/3-540-45123-4_1. (cited on p. 13)

Ethan Caballero, OpenAI, and Ilya Sutskever. Description2Code dataset, August 2016. URL https://github.com/ethancaballero/description2code. (cited on p. 11)

Federico Cassano, John Gouwar, Francesca Lucchetti, Claire Schlesinger, Carolyn Jane Anderson, Michael Greenberg, Abhinav Jangda, and Arjun Guha. Knowledge transfer from high-resource to low-resource programming languages for code LLMs. *arXiv preprint*, August 2023a. URL https://arxiv.org/abs/2308.09895. (cited on pp. 12 and 38)

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q. Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. MultiPL-E: a scalable and polyglot approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering*, 49(7):3675–3691, 2023b. doi: 10.1109/TSE.2023.3267446. URL https://www.computer.org/csdl/journal/ts/2023/07/10103177/1MpWUtj7Rwk. (cited on pp. 3 and 24)

Federico Cassano, Luisa Li, Akul Sethi, Noah Shinn, Abby Brennan-Jones, Anton Lozhkov, Carolyn Jane Anderson, and Arjun Guha. Can it edit? evaluating the ability of large language models to follow code editing instructions. In *The First International Workshop on Large Language Model for Code*, 2024. URL https://arxiv.org/abs/2312.12450. (cited on pp. 3, 27, and 28)

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. *arXiv preprint*, July 2021. URL https://arxiv.org/abs/2107.03374. (cited on pp. 2, 3, 23, 31, and 38)

Qin Chen, Jinfeng Ge, Huaqing Xie, Xingcheng Xu, and Yanqing Yang. Large language models at work in China's labor market. *arXiv preprint*, August 2023. URL https://arxiv.org/abs/2308.08776. (cited on p. 39)

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. *Journal of Machine Learning Research*, 24(240):1–113, 2023. URL http://jmlr.org/papers/v24/22-1144.html. (cited on p. 28)

Paul F. Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/d5e2c0adad503c91f91df240d0cd4e49-Abstract.html. (cited on p. 27)

ClamAV, 2024. URL https://www.clamav.net/. (cited on p. 14)

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint*, October 2021. URL https://arxiv.org/abs/2110.14168. (cited on pp. 3, 12, and 28)

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 8440–8451, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.747. URL https://aclanthology.org/2020.acl-main.747. (cited on p. 14)

Roberto Di Cosmo and Stefano Zacchiroli. Software heritage: Why and how to preserve software source code. In *iPRES 2017: 14th International Conference on Digital Preservation*, Kyoto, Japan, 2017. URL https://www.softwareheritage.org/wp-content/uploads/2020/01/ipres-2017-swh.pdf. https://hal.archives-ouvertes.fr/hal-01590958. (cited on p. 37)

Roberto Di Cosmo, Morane Gruenpeter, and Stefano Zacchiroli. Referencing source code artifacts: A separate concern in software citation. *Computing in Science & Engineering*, 22(2):33–43, 2020. doi: 10.1109/MCSE.2019.2963148. (cited on p. 39)

CrowdStrike. Polymorphic virus. https://www.crowdstrike.com/cybersecurity-101/malware/polymorphic-virus/, 2024. Accessed: 2024. (cited on p. 38)

CyberArk. Chatting our way into creating a polymorphic malware. https://www.cyberark.com/resources/threat-research-blog/chatting-our-way-into-creating-a-polymorphic-malware, 2024. Accessed: 2024. (cited on p. 38)

Cybersecurity & Infrastructure Security Agency. Secure by design, 2024. URL https://www.cisa.gov/resources-tools/resources/secure-by-design. Accessed: 2024. (cited on p. 38)

Tri Dao. FlashAttention-2: faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=mZn2Xyh9Ec. (cited on p. 21)

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: fast and memory-efficient exact attention with IO-awareness. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 16344–16359. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html. (cited on p. 21)

Harm de Vries. Go smol or go home. https://www.harmdevries.com/post/model-size-vs-compute-overhead/, 2023. (cited on p. 3)

Jwala Dhamala, Tony Sun, Varun Kumar, Satyapriya Krishna, Yada Pruksachatkun, Kai-Wei Chang, and Rahul Gupta. BOLD: dataset and metrics for measuring biases in open-ended language generation. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, pp. 862–872, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383097. doi: 10.1145/3442188.3445924. URL https://doi.org/10.1145/3442188.3445924. (cited on pp. 3 and 34)

Jennifer Ding, Christopher Akiki, Yacine Jernite, Anne Lee Steele, and Temi Popo. Towards openness beyond open access: User journeys through 3 open AI collaboratives. In *Workshop on Broadening Research Collaborations 2022*, 2022. URL https://openreview.net/forum?id=slU-5h8rrCz. (cited on p. 2)

Yangruibo Ding, Zijian Wang, Wasi Uddin Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, and Bing Xiang. CrossCodeEval: a diverse and multilingual benchmark for cross-file code completion. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=wgDcbBMSfh. (cited on pp. 3, 31, 32, and 33)

Yi Dong, Ronghui Mu, Gaojie Jin, Yi Qi, Jinwei Hu, Xingyu Zhao, Jie Meng, Wenjie Ruan, and Xiaowei Huang. Building guardrails for large language models. *arXiv preprint*, February 2024. URL https://arxiv.org/abs/2402.01822. (cited on p. 38)

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. KTO: model alignment as prospect theoretic optimization. *arXiv preprint*, February 2024. URL https://arxiv.org/abs/2402.01306. (cited on p. 27)

Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. Large language models for software engineering: Survey and open problems. *arXiv preprint*, October 2023. URL https://arxiv.org/abs/2310.03533. (cited on p. 2)

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/gao23f.html. (cited on p. 28)

Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. RealToxicityPrompts: evaluating neural toxic degeneration in language models. In Trevor Cohn, Yulan He, and Yang Liu (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 3356–3369, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.301. URL https://aclanthology.org/2020.findings-emnlp.301. (cited on pp. 3 and 34)

Gemini Team et al. Gemini: a family of highly capable multimodal models. *arXiv preprint*, 2023. URL https://arxiv.org/abs/2312.11805. (cited on p. 2)

Github Archive, 2024. URL https://gharchive.org. (cited on pp. 3, 6, and 7)

go-enry, 2024. URL https://github.com/go-enry/go-enry. (cited on pp. 4 and 6)

Goldman Sachs. The generative world order: AI, geopolitics, and power, 2024. URL https://www.goldmansachs.com/intelligence/pages/the-generative-world-order-ai-geopolitics-and-power.html. (cited on p. 39)

Governance AI. Open sourcing highly capable foundation models, 2024. URL https://www.governance.ai/research-paper/open-sourcing-highly-capable-foundation-models. Accessed: 2024. (cited on p. 38)

Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, Shane Arora, David Atkinson, Russell Authur, Khyathi Raghavi Chandu, Arman Cohan, Jennifer Dumas, Yanai Elazar, Yuling Gu, Jack Hessel, Tushar Khot, William Merrill, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Valentina Pyatkin, Abhilasha Ravichander, Dustin Schwenk, Saurabh Shah, Will Smith, Emma Strubell, Nishant Subramani, Mitchell Wortsman, Pradeep Dasigi, Nathan Lambert, Kyle Richardson, Luke Zettlemoyer, Jesse Dodge, Kyle Lo, Luca Soldaini, Noah A. Smith, and Hannaneh Hajishirzi. OLMo: accelerating the science of language models. *arXiv preprint*, February 2024. URL https://arxiv.org/abs/2402.00838. (cited on pp. 2 and 37)

Aiden Grossman, Ludger Paehler, Konstantinos Parasyris, Tal Ben-Nun, Jacob Hegna, William Moses, Jose M. Monsalve Diaz, Mircea Trofin, and Johannes Doerfert. ComPile: a large IR dataset from production sources. *arXiv preprint*, September 2023. URL https://arxiv.org/abs/2309.15432. (cited on p. 11)

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I. Wang. CRUXEval: a benchmark for code reasoning, understanding and execution. *arXiv preprint*, January 2024. URL https://arxiv.org/abs/2401.03065. (cited on pp. 3 and 29)

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. DeepSeek-Coder: when the large language model meets programming – the rise of code intelligence. *arXiv preprint*, 2024. URL https://arxiv.org/abs/2401.14196. (cited on pp. 2, 20, 22, 23, and 25)

Maanak Gupta, Charankumar Akiri, Kshitiz Aryal, Eli Parker, and Lopamudra Praharaj. From ChatGPT to ThreatGPT: impact of generative AI in cybersecurity and privacy. *IEEE Access*, 11:80218–80245, 2023. ISSN 2169-3536. doi: 10.1109/access.2023.3300381. URL http://dx.doi.org/10.1109/ACCESS.2023.3300381. (cited on p. 38)

Asier Gutiérrez-Fandiño, David Pérez-Fernández, Jordi Armengol-Estapé, David Griol, and Zoraida Callejas. esCorpius: a massive spanish crawling corpus. In *IberSPEECH 2022*, pp. 126–130, 2022. doi: 10.21437/IberSPEECH.2022-26. URL https://www.isca-speech.org/archive/pdfs/iberspeech_2022/gutierrezfandino22_iberspeech.pdf. (cited on p. 10)

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with apps. In J. Vanschoren and S. Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1. Curran, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/c24cd76e1ce41366a4bbe8a49b02a028-Abstract-round2.html. (cited on p. 12)

Software Heritage. Software heritage community. https://www.softwareheritage.org/community/, 2024. Accessed: 2024. (cited on p. 37)

Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review. *arXiv preprint*, August 2023. URL https://arxiv.org/abs/2308.10620. (cited on p. 2)

Dong Huang, Qingwen Bu, Jie Zhang, Xiaofei Xie, Junjie Chen, and Heming Cui. Bias testing and mitigation in LLM-based code generation. *arXiv preprint*, 2023. URL https://arxiv.org/abs/2309.14345. (cited on p. 38)

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7B. *arXiv preprint*, 2023. URL https://arxiv.org/abs/2310.06825. (cited on p. 2)

Albert Qiaochu Jiang, Wenda Li, Jesse Michael Han, and Yuhuai Wu. LISA: language models of ISAbelle proofs. In *6th Conference on Artificial Intelligence and Theorem Proving*, pp. 378–392, 2021. URL http://aitp-conference.org/2021/abstract/paper_17.pdf. (cited on p. 12)

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. (cited on p. 21)

Denis Kocetkov, Raymond Li, Loubna Ben allal, Jia LI, Chenghao Mou, Yacine Jernite, Margaret Mitchell, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro Von Werra, and Harm de Vries. The stack: 3 TB of permissively licensed source code. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=pxpbTdUEpD. (cited on pp. 2, 4, 36, and 37)

Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint*, October 2019. URL https://arxiv.org/abs/1910.09700. (cited on p. 21)

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-Tau Yih, Daniel Fried, Sida Wang, and Tao Yu. DS-1000: A natural and reliable benchmark for data science code generation. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 18319–18345. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/lai23b.html. (cited on pp. 3 and 25)

Chris Lattner and Vikram Adve. LLVM: a compilation framework for lifelong program analysis & transformation. In *International symposium on code generation and optimization, 2004. CGO 2004.*, pp. 75–86. IEEE, 2004. URL https://ieeexplore.ieee.org/document/1281665. (cited on p. 11)

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. StarCoder: may the source be with you! *arXiv preprint*, May 2023. URL https://arxiv.org/abs/2305.06161. (cited on pp. 2, 6, 8, 13, 14, 22, 33, 34, 36, and 37)

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 J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158. URL https://www.science.org/doi/abs/10.1126/science.abq1158. (cited on p. 12)

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023a. URL https://openreview.net/forum?id=1qvx610Cu7. (cited on pp. 3 and 23)

Tianyang Liu, Canwen Xu, and Julian McAuley. RepoBench: Benchmarking repository-level code auto-completion systems. *arXiv preprint*, June 2023b. URL https://arxiv.org/abs/2306.03091. (cited on pp. 3, 31, and 32)

Shayne Longpre, Robert Mahari, Anthony Chen, Naana Obeng-Marnu, Damien Sileo, William Brannon, Niklas Muennighoff, Nathan Khazam, Jad Kabbara, Kartik Perisetla, Xinyi Wu, Enrico Shippole, Kurt Bollacker, Tongshuang Wu, Luis Villa, Sandy Pentland, and Sara Hooker. The data provenance initiative: A large scale audit of dataset licensing & attribution in AI. *arXiv preprint*, 2023. URL https://arxiv.org/abs/2310.16787. (cited on p. 25)

Risto Luukkonen, Ville Komulainen, Jouni Luoma, Anni Eskelinen, Jenna Kanerva, Hanna-Mari Kupari, Filip Ginter, Veronika Laippala, Niklas Muennighoff, Aleksandra Piktus, et al. Fingpt: Large generative models for a small language. *arXiv preprint arXiv:2311.05640*, 2023. URL https://arxiv.org/abs/2311.05640. (cited on p. 37)

Emanuele La Malfa, Aleksandar Petrov, Simon Frieder, Christoph Weinhuber, Ryan Burnell, Raza Nazar, Anthony G. Cohn, Nigel Shadbolt, and Michael Wooldridge. Language models as a service: Overview of a new paradigm and its challenges. *arXiv preprint*, 2023. URL https://arxiv.org/abs/2309.16573. (cited on p. 37)

Marc Marone and Benjamin Van Durme. Data portraits: Recording foundation model training data. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://arxiv.org/abs/2303.03919. (cited on pp. 34 and 35)

The mathlib Community. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, POPL '20. ACM, January 2020. doi: 10.1145/3372885.3373824. URL http://dx.doi.org/10.1145/3372885.3373824. (cited on p. 12)

Daniel Mendez, Daniel Graziotin, Stefan Wagner, and Heidi Seibold. *Open Science in Software Engineering*, pp. 477–501. Springer International Publishing, 2020. doi: 10.1007/978-3-030-32489-6_17. URL http://dx.doi.org/10.1007/978-3-030-32489-6_17. (cited on p. 37)

Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pp. 369–378. Springer, 1987. (cited on p. 3)

Mehdi Mirakhorli, Derek Garcia, Schuyler Dillon, Kevin Laporte, Matthew Morrison, Henry Lu, Viktoria Koscinski, and Christopher Enoch. A landscape study of open source and proprietary tools for software bill of materials (sbom). *arXiv preprint*, 2024. URL https://arxiv.org/abs/2402.11151. (cited on p. 38)

Mike Mirzayanov. Codeforces: Results of 2020 [annual report]. https://codeforces.com/blog/entry/89502, 2020. (cited on p. 11)

Maximilian Mozes, Xuanli He, Bennett Kleinberg, and Lewis D. Griffin. Use of LLMs for illicit purposes: Threats, prevention measures, and vulnerabilities. *arXiv preprint*, 2023. URL https://arxiv.org/abs/2308.12833. (cited on p. 38)

MSFT Q2 Earning Call, 2024. URL https://www.microsoft.com/en-us/investor/events/fy-2024/earnings-fy-2024-q2.aspx. (cited on p. 2)

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*, 2022a. doi: 10.48550/ARXIV.2210.07316. URL https://arxiv.org/abs/2210.07316. (cited on p. 12)

Niklas Muennighoff, Thomas Wang, Lintang Sutawika, Adam Roberts, Stella Biderman, Teven Le Scao, M Saiful Bari, Sheng Shen, Zheng-Xin Yong, Hailey Schoelkopf, Xiangru Tang, Dragomir Radev, Alham Fikri Aji, Khalid Almubarak, Samuel Albanie, Zaid Alyafeai, Albert Webson, Edward Raff, and Colin Raffel. Crosslingual generalization through multitask finetuning, 2022b. URL https://arxiv.org/abs/2211.01786. (cited on p. 25)

Niklas Muennighoff, Alexander M Rush, Boaz Barak, Teven Le Scao, Nouamane Tazi, Aleksandra Piktus, Sampo Pyysalo, Thomas Wolf, and Colin Raffel. Scaling data-constrained language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=j5BuTrEj35. (cited on pp. 2 and 21)

Niklas Muennighoff, Qian Liu, Armel Randy Zebaze, Qinkai Zheng, Binyuan Hui, Terry Yue Zhuo, Swayam Singh, Xiangru Tang, Leandro Von Werra, and Shayne Longpre. OctoPack: instruction tuning code large language models. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=mw1PWNSWZP. (cited on pp. 3, 25, 26, 27, and 37)

Niklas Muennighoff, Hongjin Su, Liang Wang, Nan Yang, Furu Wei, Tao Yu, Amanpreet Singh, and Douwe Kiela. Generative representational instruction tuning. *arXiv preprint*, 2024b. URL https://arxiv.org/abs/2402.09906. (cited on p. 25)

J. Mökander, J. Schuett, H.R. Kirk, et al. Auditing large language models: A three-layered approach. *AI Ethics*, 2023. URL https://doi.org/10.1007/s43681-023-00289-2. (cited on p. 37)

Sebastian Nanz and Carlo A. Furia. A comparative study of programming languages in Rosetta code. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pp. 778–788. IEEE, 2015. URL https://ieeexplore.ieee.org/document/7194625. (cited on p. 12)

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. CodeGen: an open large language model for code with multi-turn program synthesis. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=iaYcJKpY2B_. (cited on p. 2)

Debora Nozza, Federico Bianchi, and Dirk Hovy. HONEST: Measuring hurtful sentence completion in language models. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2398–2406, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.191. URL https://aclanthology.org/2021.naacl-main.191. (cited on pp. 3 and 34)

OpenAI et al. GPT-4 technical report. *arXiv preprint*, March 2023. URL https://arxiv.org/abs/2303.08774. (cited on p. 2)

Pedro Javier Ortiz Suárez, Benoît Sagot, and Laurent Romary. Asynchronous pipelines for processing huge corpora on medium to low resource infrastructures. In Piotr Bański, Adrien Barbaresi, Hanno Biber, Evelyn Breiteneder, Simon Clematide, Marc Kupietz, Harald Lüngen, and Caroline Iliadi (eds.), *Proceedings of the Workshop on Challenges in the Management of Large Corpora*, pp. 9 – 16, Mannheim, July 2019. Leibniz-Institut für Deutsche Sprache. doi: 10.14618/ids-pub-9021. URL http://nbn-resolving.de/urn:nbn:de:bsz:mh39-90215. (cited on p. 10)

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. OpenWebMath: an open dataset of high-quality mathematical web text. *arXiv preprint*, October 2023. URL https://arxiv.org/abs/2310.06786. (cited on p. 13)

Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. Asleep at the keyboard? assessing the security of github copilot's code contributions. In *2022 IEEE Symposium on Security and Privacy (SP)*, pp. 754–768. IEEE, 2022. (cited on pp. 3 and 33)

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Hamza Alobeidli, Alessandro Cappelli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The RefinedWeb dataset for Falcon LLM: Outperforming curated corpora with web data only. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=kM5eGcdCzq. (cited on p. 10)

Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. The impact of AI on developer productivity: Evidence from GitHub Copilot. *arXiv preprint*, 2023. URL https://arxiv.org/abs/2302.06590. (cited on pp. 2 and 39)

Antoine Pietri, Diomidis Spinellis, and Stefano Zacchiroli. The software heritage graph dataset: Large-scale analysis of public software development history. In *MSR 2020: The 17th International Conference on Mining Software Repositories*, pp. 1–5. IEEE, 2020. doi: 10.1145/3379597.3387510. URL https://arxiv.org/abs/2011.07824https://www.softwareheritage.org/wp-content/uploads/2021/03/msr-2020-challenge.pdf. (cited on p. 3)

Aleksandra Piktus, Christopher Akiki, Paulo Villegas, Hugo Laurençon, Gérard Dupont, Sasha Luccioni, Yacine Jernite, and Anna Rogers. The ROOTS search tool: Data transparency for LLMs. In Danushka Bollegala, Ruihong Huang, and Alan Ritter (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 304–314, Toronto, Canada, July 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-demo.29. URL https://aclanthology.org/2023.acl-demo.29. (cited on p. 34)

Aleksandra Piktus, Odunayo Ogundepo, Christopher Akiki, Akintunde Oladipo, Xinyu Zhang, Hailey Schoelkopf, Stella Biderman, Martin Potthast, and Jimmy Lin. GAIA search: Hugging Face and pyserini interoperability for NLP training data exploration. In Danushka Bollegala, Ruihong Huang, and Alan Ritter (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 588–598, Toronto, Canada, July 2023b. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-demo.57. URL https://aclanthology.org/2023.acl-demo.57. (cited on p. 34)

Nikhil Pinnaparaju, Reshinth Adithyan, Duy Phung, Jonathan Tow, James Baicoianu, , and Nathan Cooper. Stable code 3B: Coding on the edge. *Stability AI*, 2024. URL https://stability.ai/news/stable-code-2024-llm-code-completion-release. (cited on p. 22)

BigCode Project. The stack v2, 2024. URL https://huggingface.co/datasets/bigcode/the-stack-v2/. Accessed: 2024. (cited on p. 37)

Ruchir Puri, David S Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, Veronika Thost, Luca Buratti, Saurabh Pujar, Shyam Ramji, Ulrich Finkler, Susan Malaika, and Frederick Reiss. CodeNet: a large-scale AI for code dataset for learning a diversity of coding tasks. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=6vZVBkCDrHT. (cited on p. 11)

RedPajama Wiki, 2024. URL https://github.com/togethercomputer/RedPajama-Data/tree/rp_v1/data_prep/wiki. (cited on p. 13)

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410. (cited on p. 12)

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint*, 2020. URL https://arxiv.org/abs/2009.10297. (cited on p. 31)

Rosetta Code, 2023. URL https://rosettacode.org/. (cited on pp. 11 and 12)

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code. *arXiv preprint*, August 2023. URL https://arxiv.org/abs/2308.12950. (cited on pp. 2, 20, 22, 23, 25, and 31)

Sane Security, 2024. URL https://sanesecurity.com/usage/signatures. (cited on p. 14)

Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M Rush. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=9Vrb9D0WI4. (cited on p. 25)

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1gR5iR5FX. (cited on p. 12)

ScanCode, 2024. URL https://github.com/nexB/scancode-toolkit. (cited on p. 3)

ScanCode License Categories, 2024. URL https://scancode-licensedb.aboutcode.org/help.html#license-categories. (cited on p. 4)

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilic, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang, Benoît Sagot, Niklas Muennighoff, Albert Villanova del Moral, Olatunji Ruwase, Rachel Bawden, Stas Bekman, Angelina McMillan-Major, Iz Beltagy, Huu Nguyen, Lucile Saulnier, Samson Tan, Pedro Ortiz Suarez, Victor Sanh, Hugo Laurençon, Yacine Jernite, Julien Launay, Margaret Mitchell, Colin Raffel, Aaron Gokaslan, Adi Simhi, Aitor Soroa, Alham Fikri Aji, Amit Alfassy, Anna Rogers, Ariel Kreisberg Nitzav, Canwen Xu, Chenghao Mou, Chris Emezue, Christopher Klamm, Colin Leong, Daniel van Strien, David Ifeoluwa Adelani, and et al. BLOOM: A 176b-parameter open-access multilingual language model. *CoRR*, abs/2211.05100, 2022a. doi: 10.48550/ARXIV.2211.05100. URL https://doi.org/10.48550/arXiv.2211.05100. (cited on p. 2)

Teven Le Scao, Thomas Wang, Daniel Hesslow, Lucile Saulnier, Stas Bekman, M Saiful Bari, Stella Biderman, Hady Elsahar, Niklas Muennighoff, Jason Phang, et al. What language model to train if you have one million gpu hours? *arXiv preprint arXiv:2210.15424*, 2022b. (cited on p. 14)

ServiceNow. Text2flow LLM: Automating workflow generation from descriptive text. https://downloads.docs.servicenow.com/resource/enus/infocard/text2flow-llm.pdf, 2024a. (cited on p. 39)

ServiceNow. Text-to-code LLM: transforming natural language into executable code, 2024b. URL https://downloads.docs.servicenow.com/resource/enus/infocard/text-to-code-llm.pdf. (cited on p. 39)

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *CoRR*, abs/1911.02150, 2019. URL http://arxiv.org/abs/1911.02150. (cited on p. 20)

Emily Sheng, Kai-Wei Chang, Premkumar Natarajan, and Nanyun Peng. The woman worked as a babysitter: On biases in language generation. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3407–3412, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1339. URL https://aclanthology.org/D19-1339. (cited on p. 34)

Dan Sholler, Igor Steinmacher, Denise Ford, Mara Averick, Mike Hoye, and Greg Wilson. Ten simple rules for helping newcomers become contributors to open projects. *PLoS Computational Biology*, 15(9):e1007296, 2019. doi: 10.1371/journal.pcbi.1007296. URL https://doi.org/10.1371/journal.pcbi.1007296. (cited on p. 37)

Shivalika Singh, Freddie Vargus, Daniel Dsouza, Börje F. Karlsson, Abinaya Mahendiran, Wei-Yin Ko, Herumb Shandilya, Jay Patel, Deividas Mataciunas, Laura OMahony, Mike Zhang, Ramith Hettiarachchi, Joseph Wilson, Marina Machado, Luisa Souza Moura, Dominik Krzemiński, Hakimeh Fadaei, Irem Ergün, Ifeoma Okoh, Aisha Alaagib, Oshan Mudannayake, Zaid Alyafeai, Vu Minh Chien, Sebastian Ruder, Surya Guthikonda, Emad A. Alghamdi, Sebastian Gehrmann, Niklas Muennighoff, Max Bartolo, Julia Kreutzer, Ahmet Üstün, Marzieh Fadaee, and Sara Hooker. Aya dataset: An open-access collection for multilingual instruction tuning. *arXiv preprint*, 2024. URL https://arxiv.org/abs/2402.06619. (cited on p. 37)

Software Heritage. Swh statement on llm for code, 2023. URL https://www.softwareheritage.org/2023/10/19/swh-statement-on-llm-for-code/. (cited on p. 38)

Software Heritage. Bulk access terms of use, 2024a. URL https://www.softwareheritage.org/legal/bulk-access-terms-of-use/. (cited on p. 37)

Software Heritage, 2024b. URL https://www.softwareheritage.org. (cited on p. 7)

Irene Solaiman. The gradient of generative AI release: Methods and considerations. *arXiv preprint*, 2023. URL https://arxiv.org/abs/2302.04844. (cited on pp. 2, 37, and 38)

Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, Valentin Hofmann, Ananya Harsh Jha, Sachin Kumar, Li Lucy, Xinxi Lyu, Nathan Lambert, Ian Magnusson, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik,

Crystal Nam, Matthew E. Peters, Abhilasha Ravichander, Kyle Richardson, Zejiang Shen, Emma Strubell, Nishant Subramani, Oyvind Tafjord, Pete Walsh, Luke Zettlemoyer, Noah A. Smith, Hannaneh Hajishirzi, Iz Beltagy, Dirk Groeneveld, Jesse Dodge, and Kyle Lo. Dolma: an open corpus of three trillion tokens for language model pretraining research. *arXiv preprint*, 2024. URL https://arxiv.org/abs/2402.00159. (cited on p. 37)

StackExchange Archive, 2024. URL https://archive.org/details/stackexchange. (cited on p. 12)

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *arXiv preprint*, April 2021. URL https://arxiv.org/abs/2104.09864. (cited on p. 20)

Marc Szafraniec, Baptiste Roziere, Hugh James Leather, Patrick Labatut, Francois Charton, and Gabriel Synnaeve. Code translation with compiler representations. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=XomEU3eNeSQ. (cited on p. 11)

Feiyang Tang, Bjarte M. Østvold, and Magiel Bruntink. Helping code reviewer prioritize: Pinpointing personal data and its processing. *Frontiers in Artificial Intelligence and Applications*, 371:109–124, 2023. doi: 10.3233/FAIA230228. (cited on p. 38)

Feiyang Tang and Bjarte M. Østvold. Finding privacy-relevant source code. *arXiv preprint*, 2024. URL https://arxiv.org/abs/2401.07316. (cited on p. 38)

The SWHID Specification Project. The SWHID specification, 2024. URL https://www.swhid.org/. (cited on p. 38)

Together Computer. RedPajama: an open dataset for training large language models, October 2023. URL https://github.com/togethercomputer/RedPajama-Data. (cited on p. 12)

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint*, 2023. URL https://arxiv.org/abs/2307.09288. (cited on p. 12)

Ahmet Üstün, Viraat Aryabumi, Zheng-Xin Yong, Wei-Yin Ko, Daniel D'souza, Gbemileke Onilude, Neel Bhandari, Shivalika Singh, Hui-Lee Ooi, Amr Kayid, Freddie Vargus, Phil Blunsom, Shayne Longpre, Niklas Muennighoff, Marzieh Fadaee, Julia Kreutzer, and Sara Hooker. Aya model: An instruction finetuned open-access multilingual language model. *arXiv preprint*, 2024. URL https://arxiv.org/abs/2402.07827. (cited on p. 37)

Bertie Vidgen, Tristan Thrush, Zeerak Waseem, and Douwe Kiela. Learning from the worst: Dynamically generated datasets to improve online hate detection. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1667–1682, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.132. URL https://aclanthology.org/2021.acl-long.132. (cited on p. 34)

Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. Software testing with large language models: Survey, landscape, and vision. *IEEE Transactions on Software Engineering*, pp. 1–27, 2024. doi: 10.1109/TSE.2024.3368208. URL https://arxiv.org/abs/2307.07221. (cited on p. 2)

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=gEZrGCozdqR. (cited on p. 25)

Michael Woelfle, Piero L Olliaro, and Matthew H. Todd. Open science is a research accelerator. *Nature chemistry*, 3 10:745–8, 2011. URL https://api.semanticscholar.org/CorpusID:205289283. (cited on p. 37)

World Economic Forum. Jobs of tomorrow: Large language models and jobs, 2024. URL https://www.weforum.org/publications/jobs-of-tomorrow-large-language-models-and-jobs/. (cited on p. 39)

Yiheng Xu, Hongjin Su, Chen Xing, Boyu Mi, Qian Liu, Weijia Shi, Binyuan Hui, Fan Zhou, Yitao Liu, Tianbao Xie, Zhoujun Cheng, Siheng Zhao, Lingpeng Kong, Bailin Wang, Caiming Xiong, and Tao Yu. Lemur: Harmonizing natural language and code for language agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=hNhwSmtXRh. (cited on p. 37)

Yahoo Finance. ServiceNow Inc (NYSE: NOW) Q4 earnings: What to expect, 2024. URL https://finance.yahoo.com/news/servicenow-inc-nyse-now-q4-154816487.html. (cited on pp. 2 and 39)

Zhou Yang, Zhipeng Zhao, Chenyu Wang, Jieke Shi, Dongsum Kim, Donggyun Han, and David Lo. Gotcha! this model uses my code! evaluating membership leakage risks in code models. *arXiv preprint*, 2023. URL https://arxiv.org/abs/2310.01166. (cited on p. 38)

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint*, August 2023. URL https://arxiv.org/abs/2308.01825. (cited on p. 12)

Jieyu Zhao, Tianlu Wang, Mark Yatskar, Vicente Ordonez, and Kai-Wei Chang. Gender bias in coreference resolution: Evaluation and debiasing methods. In Marilyn Walker, Heng Ji, and Amanda Stent (eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 15–20, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2003. URL https://aclanthology.org/N18-2003. (cited on p. 34)

Terry Yue Zhuo, Yujin Huang, Chunyang Chen, and Zhenchang Xing. Red teaming ChatGPT via jailbreaking: Bias, robustness, reliability and toxicity. *arXiv preprint*, 2023a. URL https://arxiv.org/abs/2301.12867. (cited on p. 38)

Terry Yue Zhuo, Zhou Yang, Zhensu Sun, Yufei Wang, Li Li, Xiaoning Du, Zhenchang Xing, and David Lo. Source code data augmentation for deep learning: A survey. *arXiv preprint*, May 2023b. URL https://arxiv.org/abs/2305.19915. (cited on pp. 2, 11, and 38)

Terry Yue Zhuo, Armel Zebaze, Nitchakarn Suppattarachai, Leandro von Werra, Harm de Vries, Qian Liu, and Niklas Muennighoff. Astraios: Parameter-efficient instruction tuning code large language models. *arXiv preprint*, August 2024. URL https://arxiv.org/abs/2401.00788. (cited on pp. 25 and 37)

Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. Measuring GitHub Copilot's impact on productivity. *Commun. ACM*, 67(3):54–63, feb 2024. ISSN 0001-0782. doi: 10.1145/3633453. URL https://doi.org/10.1145/3633453. (cited on pp. 2 and 39)

A Data Curation

A.1 Excluded Extensions

AL (al), AngelScript (as), AsciiDoc (asc), AspectJ (aj), Bison (bison), Boogie (bpl),
C++ (<empty extension>), Cabal Config (project), ChucK (ck), CODEOWNERS (<empty extension>),
Common Lisp (l, sexp), Common Workflow Language (cwl), CoNLL-U (conll, conllu), Cue Sheet (cue),
CWeb (w), desktop (desktop, in, service), DIGITAL Command Language (com), DTrace (d), edn (edn),
Elixir (lock), Factor (factor), GAP (g, gd), Gemfile.lock (lock), Gettext Catalog (pot),
Git Config (gitmodules), GLSL (geo), Glyph Bitmap Distribution Format (bdf), GN (gn),
Ignore List (dockerignore, eslintignore, gitignore, npmignore), INI (cfg, prefs, url),
JAR Manifest (mf), Java Properties (properties), Jest Snapshot (snap), JetBrains MPS (mps),
JSONLD (jsonld), LiveScript (ls), Makefile (d, make), Mathematica (cdf, nb), MAXScript (ms),
mIRC Script (mrc), NASL (inc), nesC (nc), Nunjucks (njk), OpenEdge ABL (p, w),
Pascal (<empty extension>, dpr, inc, pp), Perl (al, ph), PLSQL (pck, pls, tps, trg, vw),
Protocol Buffer Text Format (pbt), Puppet (<empty extension>), PureBasic (pb), Racket (rkt, rktd),
ReScript (res), reStructuredText (rest), Rich Text Format (rtf), Roff (<empty extension>, 1, 1d, 2,
5, 7, 8, 9, in), Roff Manpage (<empty extension>, 1d, 2, 3d, 4, 6, 9, man), Scala (sc), Scilab (tst),
SELinux Policy (te), Shell (env), Slash (sl), Smalltalk (cs), SmPL (cocci), SQL (tab), Standard ML (sig),
Stata (ihlp, sthlp), SuperCollider (sc), SWIG (i), TeX (aux, ltx, toc), TOML (lock), Turtle (ttl),
VBA (frm, frx), Vim Snippet (snippet), Wavefront Material (mtl), Wikitext (wikitext),
Windows Registry Entries (reg), wisp (w), World of Warcraft Addon Data (toc), X BitMap (xbm),
XML (kml, pt, resx, rss), XML Property List (plist, tmcommand, tmlanguage, tmsnippet, tmtheme), Yacc (yy).

A.2 Excluded Programming Languages

2-Dimensional Array,AGS Script,Bicep,Checksums,COLLADA,CSV,Diff,DirectX 3D File,E-mail,G-code,
Gerber Image,Git Revision List,Gnuplot,Go,Checksums,IRC log,Jupyter Notebook,KiCad Layout,
KiCad Legacy Layout,KiCad Schematic,Lasso,Linux,Kernel Module,Max,
Microsoft Developer Studio Project,Microsoft Visual Studio Solution,Pickle,PostScript,
POV-Ray SDL,Public Key,Pure Data,Raw token data,robots.txt,STL,SubRip Text,SVG,TSV,
Unity3D Asset,Wavefront Object,WebVTT,X PixMap

A.3 License detection

```
license_file_names = [
    "li[cs]en[cs]e(s?)",
    "legal",
    "copy(left|right|ing)",
    "unlicense",
    "[al]?gpl([-_ v]?)(\d\.?\d?)?",  # AGPLv3
    "bsd(l?)",  # BSDL
    "mit(x?)",  # MITX
    "apache",
    "artistic",  # Artistic.txt
    "copying(v?)(\d?)",  # COPYING3, COPYINGv3
    "disclaimer",
    "eupl",
    "gfdl",
    "[cm]pl",
    "cc0",
    "al([-_ v]?)(\d\.?\d)?",  # AL2.0
    "about",
    "notice",
```

```
    "readme",
    "guidelines",
]

license_file_re = re.compile(
    rf"^(|.*[-_. ])({'|'.join(license_file_names)})(|[-_. ].*)$", re.IGNORECASE
)
```

A.4 Permissive licenses

SPDX-recognized license IDs 0BSD, AAL, Abstyles, AdaCore-doc, Adobe-2006, Adobe-Glyph, ADSL, AFL-1.1, AFL-1.2, AFL-2.0, AFL-2.1, AFL-3.0, Afmparse, AMDPLPA, AML, AMPAS, ANTLR-PD, Apache-1.0, Apache-1.1, Apache-2.0, APAFML, App-s2p, Artistic-1.0, Artistic-1.0-cl8, Artistic-1.0-Perl, Artistic-2.0, Baekmuk, Bahyph, Barr, Beerware, Bitstream-Charter, Bitstream-Vera, BlueOak-1.0.0, Boehm-GC, Borceux, Brian-Gladman-3-Clause, BSD-1-Clause, BSD-2-Clause, BSD-2-Clause-Patent, BSD-2-Clause-Views, BSD-3-Clause, BSD-3-Clause-Attribution, BSD-3-Clause-Clear, BSD-3-Clause-LBNL, BSD-3-Clause-Modification, BSD-3-Clause-No-Nuclear-License-2014, BSD-3-Clause-No-Nuclear-Warranty, BSD-3-Clause-Open-MPI, BSD-4-Clause, BSD-4-Clause-Shortened, BSD-4-Clause-UC, BSD-4.3RENO, BSD-4.3TAHOE, BSD-Advertising-Acknowledgement, BSD-Attribution-HPND-disclaimer, BSD-Source-Code, BSL-1.0, bzip2-1.0.6, Caldera, CC-BY-1.0, CC-BY-2.0, CC-BY-2.5, CC-BY-2.5-AU, CC-BY-3.0, CC-BY-3.0-AT, CC-BY-3.0-DE, CC-BY-3.0-NL, CC-BY-3.0-US, CC-BY-4.0, CDLA-Permissive-1.0, CDLA-Permissive-2.0, CECILL-B, CERN-OHL-1.1, CERN-OHL-1.2, CERN-OHL-P-2.0, CFITSIO, checkmk, ClArtistic, Clips, CMU-Mach, CNRI-Jython, CNRI-Python, CNRI-Python-GPL-Compatible, COIL-1.0, Community-Spec-1.0, Condor-1.1, Cornell-Lossless-JPEG, Crossword, CrystalStacker, Cube, curl, DL-DE-BY-2.0, DOC, Dotseqn, DRL-1.0, DSDP, dtoa, dvipdfm, ECL-1.0, ECL-2.0, EFL-1.0, EFL-2.0, eGenix, Entessa, EPICS, etalab-2.0, EUDatagrid, Fair, FreeBSD-DOC, FSFAP, FSFULLR, FSFULLRWD, FTL, GD, Giftware, Glulxe, GLWTPL, Graphics-Gems, GStreamer-exception-2005, HaskellReport, HP-1986, HPND, HPND-Markus-Kuhn, HPND-sell-variant, HPND-sell-variant-MIT-disclaimer, HTMLTIDY, IBM-pibs, ICU, IJG, IJG-short, ImageMagick, iMatix, Info-ZIP, Intel, Intel-ACPI, ISC, Jam, JasPer-2.0, JPNIC, JSON, Kazlib, Knuth-CTAN, Latex2e, Latex2e-translated-notice, Leptonica, Libpng, libpng-2.0, libtiff, Linux-OpenIB, LLVM-exception, LOOP, LPL-1.0, LPL-1.02, LPPL-1.3c, Martin-Birgmeier, metamail, Minpack, MirOS, MIT, MIT-0, MIT-advertising, MIT-CMU, MIT-enna, MIT-feh, MIT-Festival, MIT-Modern-Variant, MIT-open-group, MIT-Wu, MITNFA, mpich2, mplus, MS-LPL, MS-PL, MTLL, MulanPSL-1.0, MulanPSL-2.0, Multics, Mup, NAIST-2003, NASA-1.3, Naumen, NBPL-1.0, NCSA, Net-SNMP, NetCDF, Newsletr, NICTA-1.0, NIST-PD-fallback, NIST-Software, NLOD-1.0, NLOD-2.0, NRL, NTP, NTP-0, O-UDA-1.0, ODC-By-1.0, OFFIS, OFL-1.0, OFL-1.0-no-RFN, OFL-1.0-RFN, OFL-1.1-no-RFN, OFL-1.1-RFN, OGC-1.0, OGDL-Taiwan-1.0, OGL-Canada-2.0, OGL-UK-1.0, OGL-UK-2.0, OGL-UK-3.0, OGTSL, OLDAP-1.1, OLDAP-1.2, OLDAP-1.3, OLDAP-1.4, OLDAP-2.0, OLDAP-2.0.1, OLDAP-2.1, OLDAP-2.2, OLDAP-2.2.1, OLDAP-2.2.2, OLDAP-2.3, OLDAP-2.4, OLDAP-2.5, OLDAP-2.6, OLDAP-2.7, OLDAP-2.8, OML, OpenSSL, OPUBL-1.0, PHP-3.0, PHP-3.01, Plexus, PostgreSQL, PSF-2.0, psfrag, psutils, Python-2.0, Python-2.0.1, Qhull, Rdisc, RSA-MD, Ruby, Saxpath, SCEA, SchemeReport, Sendmail, SGI-B-1.1, SGI-B-2.0, SGP4, SHL-0.5, SHL-0.51, SHL-2.0, SHL-2.1, SMLNJ, snprintf, Spencer-86, Spencer-94, Spencer-99, SSH-OpenSSH, SSH-short, SunPro, Swift-exception, SWL, TCL, TCP-wrappers, TermReadKey, TPDL, TTWL, TU-Berlin-1.0, TU-Berlin-2.0, UCAR, Unicode-DFS-2015, Unicode-DFS-2016, UnixCrypt, UPL-1.0, Vim, VSL-1.0, W3C, W3C-19980720, W3C-20150513, w3m, Widget-Workshop, Wsuipa, X11, X11-distribute-modifications-variant, Xdebug-1.03, Xerox, Xfig, XFree86-1.1, xinetd, xlock, Xnet, xpp, XSkat, Zed, Zend-2.0, Zlib, zlib-acknowledgement, ZPL-1.1, ZPL-2.0, ZPL-2.1

ScanCode-specific license IDs LicenseRef-scancode-{3com-microcode, 3dslicer-1.0, 4suite-1.1, accellera-systemc, adi-bsd, adrian, agere-bsd, alexisisaac-freeware, amd-historical, ams-fonts, anu-license, apache-patent-exception, apple-attribution, apple-attribution-1997, apple-excl, apple-sscl, aravindan-premkumar, argouml, arm-llvm-sga, array-input-method-pl, asmus, asn1, atkinson-hyperlegible-font, bakoma-fonts-1995, bea-2.1, beal-screamer, beri-hw-sw-1.0, bigdigits, bigelow-holmes, biopython, bitzi-pd, blas-2017, bohl-0.2, boost-original, boutell-libgd-2021, bpmn-io, brent-corkum, brian-clapper, brian-gladman, brian-gladman-3-clause, broadcom-cfe, broadcom-linux-timer, brocade-firmware, bruno-podetti, bsd-1-clause-build, bsd-1988, bsd-2-

clause-plus-advertizing, bsd-3-clause-devine, bsd-3-clause-fda, bsd-3-clause-jtag, bsd-3-clause-no-change, bsd-3-clause-no-trademark, bsd-3-clause-sun, bsd-ack-carrot2, bsd-artwork, bsd-atmel, bsd-axis-nomod, bsd-credit, bsd-dpt, bsd-export, bsd-innosys, bsd-mylex, bsd-new-derivative, bsd-new-nomod, bsd-new-tcpdump, bsd-no-disclaimer, bsd-no-disclaimer-unmodified, bsd-original-muscle, bsd-original-voices, bsd-plus-mod-notice, bsd-simplified-darwin, bsd-simplified-intel, bsd-simplified-source, bsd-top, bsd-top-gpl-addition, bsd-unchanged, bsd-unmodified, bsd-x11, bsla-no-advert, bytemark, can-ogl-alberta-2.1, can-ogl-british-columbia-2.0, can-ogl-nova-scotia-1.0, can-ogl-ontario-1.0, can-ogl-toronto-1.0, careware, carnegie-mellon, cavium-malloc, cc-by-2.0-uk, cecill-b-en, cern-attribution-1995, cgic, chicken-dl-0.2, chris-maunder, chris-stoy, classic-vb, clear-bsd-1-clause, click-license, cmu-mit, cmu-simple, cmu-template, code-credit-license-1.0.1, code-credit-license-1.1.0, codeguru-permissions, codesourcery-2004, commonj-timer, compass, componentace-jcraft, compuphase-linking-exception, cosl, cpm-2022, cpp-core-guidelines, crcalc, cryptopp, csprng, cve-tou, cwe-tou, cximage, d-zlib, damail, dante-treglia, dbad-1.1, delorie-historical, dhtmlab-public, dl-de-by-1-0-de, dl-de-by-1-0-en, dl-de-by-2-0-en, dmalloc, dmtf-2017, docbook, douglas-young, drl-1.1, dropbear, dropbear-2016, dtree, dwtfnmfpl-3.0, dynamic-drive-tou, ecfonts-1.0, egenix-1.0.0, ellis-lab, emit, emx-library, energyplus-bsd, epaperpress, eric-glass, errbot-exception, etalab-2.0-en, fabien-tassin, far-manager-exception, fastbuild-2012-2020, fatfs, fftpack-2004, filament-group-mit, flex-2.5, flora-1.1, font-alias, fpl, fplot, fraunhofer-iso-14496-10, free-art-1.3, freebsd-boot, freebsd-first, freemarker, fsf-notice, fujion-exception-to-apache-2.0, gareth-mccaughan, gary-s-brown, gdcl, geoff-kuenning-1993, ghostpdl-permissive, glut, good-boy, greg-roelofs, gregory-pietsch, gtpl-v1, gtpl-v2, gtpl-v3, happy-bunny, hdf4, hdf5, hdparm, hidapi, historical-ntp, homebrewed, hp-snmp-pp, html5, httpget, ian-kaplan, ian-piumarta, ibm-as-is, ibm-dhcp, ibm-icu, ibm-nwsc, ibm-sample, ibpp, icot-free, idt-notice, ietf, ietf-trust, ilmid, indiana-extreme, infineon-free, info-zip-1997-10, info-zip-2001-01, info-zip-2002-02, info-zip-2003-05, info-zip-2004-05, info-zip-2005-02, info-zip-2007-03, info-zip-2009-01, inno-setup, intel-bsd, intel-bsd-2-clause, intel-osl-1989, intel-osl-1993, intel-royalty-free, iso-14496-10, iso-8879, itu, ja-sig, jason-mayes, jasper-1.0, java-app-stub, jdbm-1.00, jdom, jetty, jgraph, jpnic-mdnkit, jpython-1.1, jscheme, jsfromhell, jython, kalle-kaukonen, keith-rule, kerberos, kevan-stannard, kevlin-henney, khronos, kumar-robotics, lcs-telegraphics, ldap-sdk-free-use, libgeotiff, libmib, libmng-2007, libsrv-1.0.2, lil-1, lilo, linux-device-drivers, linuxbios, linuxhowtos, llnl, logica-1.0, lucre, make-human-exception, matt-gallagher-attribution, matthew-kwan, mattkruse, mediainfo-lib, mgopen-font-license, michael-barr, michigan-disclaimer, mit-1995, mit-license-1998, mit-modification-obligations, mit-nagy, mit-no-advert-export-control, mit-no-trademarks, mit-old-style, mit-old-style-sparse, mit-readme, mit-specification-disclaimer, mit-synopsys, mit-taylor-variant, mit-veillard-variant, mod-dav-1.0, motorola, mpeg-iso, mpeg-ssg, ms-sspl, ms-ws-routing-spec, msj-sample-code, mulanpsl-1.0-en, mulanpsl-2.0-en, mulle-kybernetik, musl-exception, mx4j, netcat, netcomponents, netron, newlib-historical, newran, nice, niels-ferguson, nilsson-historical, nist-srd, node-js, nonexclusive, nortel-dasa, notre-dame, nrl-permission, ntlm, ntpl-origin, nvidia, nvidia-2002, nvidia-gov, nwhm, nysl-0.9982, nysl-0.9982-jp, o-young-jong, oasis-ws-security-spec, object-form-exception-to-mit, odl, odmg, ogc, ogl-1.0a, ogl-canada-2.0-fr, ogl-wpd-3.0, openmarket-fastcgi, openorb-1.0, opensaml-1.0, openssl, opml-1.0, opnl-1.0, opnl-2.0, oreilly-notice, oswego-concurrent, other-permissive, owtchart, ozplb-1.0, ozplb-1.1, paolo-messina-2000, paraview-1.2, patent-disclaimer, paul-mackerras, paul-mackerras-binary, paul-mackerras-new, paul-mackerras-simplified, paulo-soares, paypal-sdk-2013-2016, pcre, pd-mit, pd-programming, perl-1.0, peter-deutsch-document, philippe-de-muyter, phorum-2.0, php-2.0.2, pine, pngsuite, politepix-pl-1.0, ppp, protobuf, psf-3.7.2, psytec-freesoft, purdue-bsd, pybench, pycrypto, pygres-2.2, python-cwi, qlogic-microcode, qpopper, qualcomm-turing, quirksmode, radvd, red-hat-attribution, red-hat-bsd-simplified, reportbug, ricebsd, richard-black, robert-hubley, rsa-1990, rsa-cryptoki, rsa-demo, rsa-md4, rtools-util, rute, ryszard-szopa, saas-mit, saf, sash, sata, sbia-b, scancode-acknowledgment, scanlogd-license, scansoft-1.2, scintilla, scribbles, script-asylum, secret-labs-2011, service-comp-arch, sgi-cid-1.0, sgi-glx-1.0, sglib, shital-shah, simpl-1.1, softfloat, softfloat-2.0, softsurfer, sparky, speechworks-1.1, ssleay, ssleay-windows, stanford-pvrg, stlport-2000, stlport-4.5, stream-benchmark, stu-nicholls, sun-rpc, sun-source, sunsoft, supervisor, svndiff, swig, symphonysoft, synopsys-mit, synthesis-toolkit, takao-abe, takuya-ooura, tcg-spec-license-v1, tekhvc, tested-software, tex-live, things-i-made-public-license, tiger-crypto, tigra-calendar-3.2, tigra-calendar-4.0, tim-janik-2003, timestamp-picker, tso-license, ttcl, ttyp0, tumbolia, twisted-snmp, ubc, unicode, unicode-icu-58, unicode-mappings, unlimited-binary-use-exception, unpbook, us-govt-unlimited-rights, usrobotics-permissive, utopia, vcalendar, vince, visual-idiot, visual-numerics, vixie-cron, w3c-03-bsd-license, westhawk, whistle, whitecat, wide-license, william-alexander, wingo, wol, wordnet, wrox, ws-addressing-spec, ws-policy-specification, ws-trust-specification, wtfnmfpl-1.0, wxwidgets, wxwindows-u-3.0, x11-acer, x11-adobe, x11-adobe-dec, x11-dec1, x11-dec2, x11-doc,

x11-dsc, x11-hanson, x11-lucent-variant, x11-oar, x11-opengl, x11-quarterdeck, x11-realmode, x11-sg, x11-stanford, x11-tektronix, x11-x11r5, x11-xconsortium-veillard, xfree86-1.0, xmldb-1.0, xxd, yale-cas, yensdesign, zeusbench, zpl-1.0, zsh, zuora-software, zveno-research}

Non-licenses The following contributor license agreements, warranty disclaimers, and other license amendments were not considered during license labeling: LicenseRef-scancode-{dco-1.1, generic-cla, google-cla, jetty-ccla-1.1, newton-king-cla, generic-exception, generic-export-compliance, generic-tos, generic-trademark, warranty-disclaimer}

A.5 Pull Requests

Table 24 shows the volume of PR renderings for various sequence lengths (measured in characters). We list the volume of the base files for the top 20 languages in Table 25.

A.6 StackOverflow

We used the following prompt to

```
Below is an instruction from a user and a candidate's answer. Evaluate whether or not the answer is
a good example of how AI Assistant should respond to the user's instruction. Please assign a score
using the following 10-point scale:

1: The response is entirely off-topic, contains significant inaccuracies, or is incomprehensible.
It fails to address the user's query in any meaningful way.

2: The answer is largely irrelevant, vague, or controversial. It contains some elements that relate
to the topic but misses the core of the user's question or includes substantial misinformation.

3: The response is somewhat relevant but remains incomplete or contains elements that are
off-topic or controversial. Key aspects of the user's query are left unaddressed.

4: The answer addresses the user's question to some extent but lacks depth or clarity. It may be
somewhat helpful but is not comprehensive or detailed.

5: The response is relevant and offers a basic answer to the user's question but lacks detail or
specificity. It's helpful but not fully developed or insightful.

6: The answer is moderately helpful and addresses most aspects of the user's question. It might
lack some depth or contain minor inaccuracies or irrelevant information.

7: The response is quite helpful and addresses the user's query well, but it might not be from an
AI Assistant's perspective. It could resemble content from other sources like blog posts or web pages.

8: The answer is comprehensive and relevant, written from an AI assistant's perspective. It
addresses the user's query effectively but may have minor areas for improvement in focus,
conciseness, or organization.

9: The response is almost perfect, providing a clear, comprehensive, and well-organized answer from an
AI assistant's perspective. It might have very minor areas for improvement in terms of engagement or
insight.

10: The answer is exemplary, perfectly addressing the user's query from an AI Assistant's perspective.
It is highly informative, expertly written, engaging, and insightful, with no discernible areas
for improvement.
```

Table 24: *Volume of the pull requests dataset when we restrict the sequence length.*

Seqlen (characters)	Volume (GB)
25000	19.6
50000	38.7
75000	54.34
100000	67.31
200000	103.52
300000	126.8
400000	143.65
500000	156.76
600000	167.21
700000	175.94
800000	183.18
900000	189.32
1000000	194.58

Table 25: *Size of base files range of changes for top 20 languages in Pull Requests.*

Language	Volume (GB)
Python	13.46
JavaScript	9.55
Java	8.37
Markdown	7.34
C++	5.89
Go	5.59
JSON	4.13
TypeScript	3.96
C#	3.76
YAML	3.1
XML	2.55
C	2.34
HTML	2.31
Rust	2.27
PHP	2.09
Ruby	1.73
project.pbxproj	1.51
Scala	1.25
TSX	1.2
Swift	0.9

```
Please write "Score: <rating>" in the last line, and then provide a brief reasoning you used to derive the rating score.
```

A.7 Kaggle Notebooks templates

We remove the following templates if they appear at the beginning of a Kaggle notebook:

```
TEMPLATE_1 = '# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
import numpy as np  # linear algebra
import pandas as pd  # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory
import os

for dirname, _, filenames in os.walk("/kaggle/input"):
    for filename in filenames:
        print(os.path.join(dirname, filename))
# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session'
TEMPLATE_2 = '# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python\n'
```

Table 26: *Top 10 detected malware signatures.*

Signature	Count
Sanesecurity.Malware.28845.BadVBS	11876
winnow.compromised.ts.jsexploit.5	2251
Sanesecurity.Malware.26492.JsHeur	2247
Sanesecurity.Spam.8879	1597
Sanesecurity.Malware.25834.JsHeur	1560
Sanesecurity.Malware.27112.JsHeur	1258
Sanesecurity.Malware.26222.JsHeur	888
Porcupine.Malware.52833	814
Sanesecurity.SpamL.8887	792
Sanesecurity.Malware.26557.JsHeur	728

Table 27: *Top 10 languages by the number of potentially malicious files.*

Language	Count
Text	13281
HTML	11336
JavaScript	10210
VBScript	7947
Logos	3283
Markdown	2736
Linker Script	1390
XML	1260
VBA	990
JSON	547

B Processing Pipeline

B.1 Malware removal

We show the top-10 detected malware signatures in Table 26 and the top-10 languages by potentially malicous files in Table 27.

C Data Composition

C.1 TheStackV2-train-smol

- Configuration languages
  - Ant Build System
  - CMake
  - Dockerfile
  - Go Module
  - Gradle
  - INI
  - Java Properties
  - Makefile
  - Maven POM
  - TOML
- Configuration files:
  - CMakeLists.txt
  - Cargo.toml
  - DESCRIPTION
  - Gemfile
  - Makefile
  - Makefile.am
  - NAMESPACE
  - Package.swift
  - Pipfile
  - build.gradle
  - build.gradle.kts
  - composer.json
  - conda.yml
  - configure.ac
  - docker-compose.yaml
  - docker-compose.yml
  - go.mod
  - package.json
  - pom.xml
  - pyproject.toml
  - requirements-dev.txt
  - requirements-prod.txt
  - requirements.in
  - requirements.test.txt
  - requirements.txt
  - setup.cfg
  - tsconfig.json
  - yarn.lock

C.2 TheStackV2-train-full

In Table 28, we summarize the data volume for the subsamples languages.

Table 28: *Subsampling volumes for languages in the Stack v2 dataset.*

Final volume	**Languages**
200GB	Java, JavaScript
100GB	HTML
8GB	CSS, Java Server Pages, JSON, SCSS, Smali, XML, YAML
1GB	BibTeX, Gettext Catalog, Graphviz (DOT), Java Properties, Roff, Roff Manpage, Web Ontology Language