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:

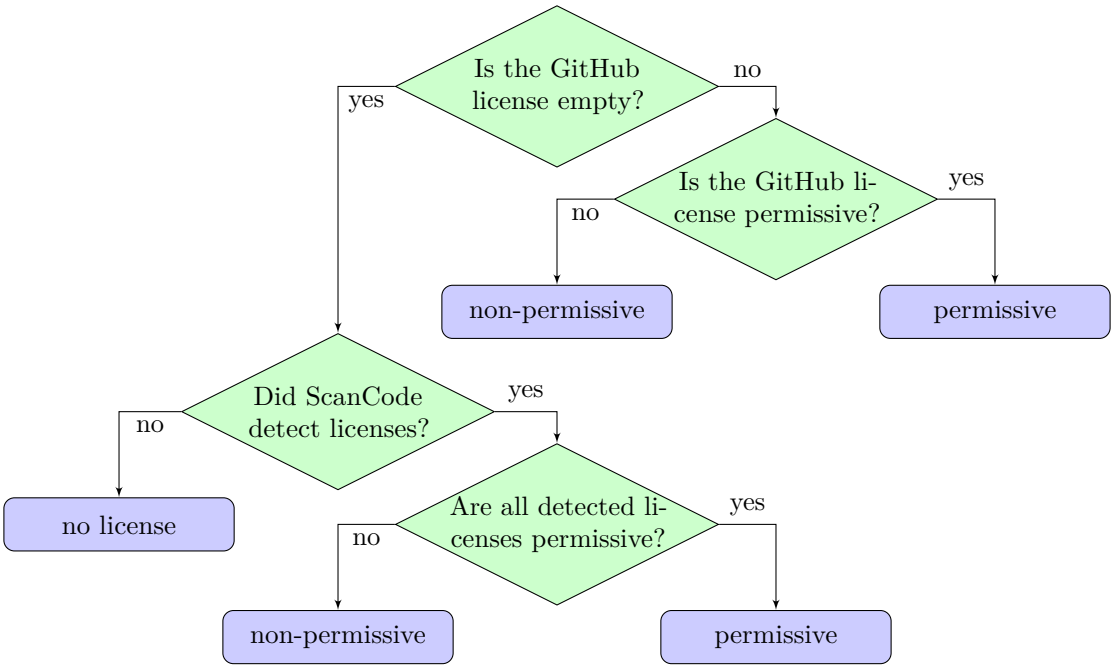

Figure 1: *File-level license assignment logic.*

- 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.