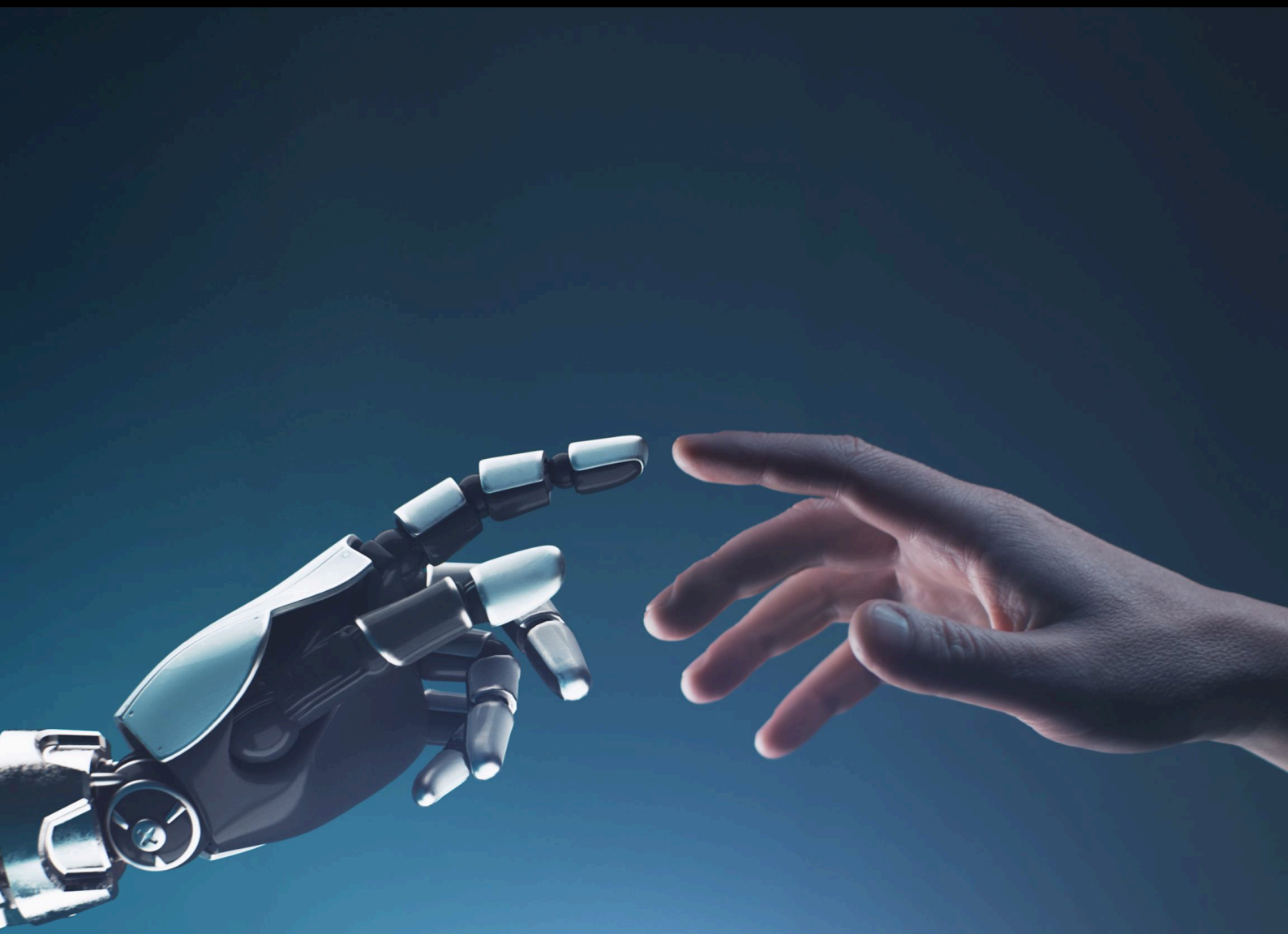
AI for Software Engineering

Robust Dataset Practices for LLMs4Code

Razvan Mihai Popescu

MSc Thesis — Computer Science — Artificial Intelligence Technology



TU Delft

Dataset Development for LLMs4Code: Licensing, Contamination, and Reproducibility Challenges

Thesis submitted to Delft University of Technology in partial fulfillment of the requirements for the degree of

Master of Science

in

Artifical Intelligence Technology

by

Razvan Mihai Popescu
21 February 2025

Razvan Mihai Popescu: *Dataset Development for LLMs4Code: Licensing, Contamination, and Reproducibility Challenges* (2025)

The work in this thesis was conducted under the guidance and in collaboration with:

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands

Thesis advisor:	Prof. Dr. Arie van Deursen
Daily supervisor:	Dr. Maliheh Izadi
Co-reader:	Dr. Jie Yang

Abstract

The rapid rise in the popularity of large language models has highlighted the need for extensive datasets, especially for training on code. However, this growth has also raised important questions about the legal implications of using code in large language model training, particularly regarding the potential infringement of code licenses. At the same time, the availability of clean datasets for evaluating these models is becoming increasingly limited, due to a high risk of contamination which restricts the capacity for reliable research. On top of that, this requires researchers to repeatedly perform data curation steps in order to evaluate their models on downstream tasks, based on previously unseen data. This process is not only time- and resource-intensive but also introduces potential inconsistencies across studies, which can impact their reproducibility.

We address these challenges through a comprehensive licensing analysis and by developing robust datasets to support accurate and reproducible large language model evaluations. We compiled a list of 53 large language models trained on file-level code and analyzed their datasets, discovering pervasive license inconsistencies despite careful selection based on repository licenses. Our analysis, covering 514M code files, reveals 38M exact duplicates of strong copyleft code, and 171M file-leading comments, 16M of which are under copyleft licenses and another 11M discouraging unauthorized copying. To further understand the depth of non-permissive code in public training datasets, we developed StackLessV2, a strong copyleft Java dataset decontaminated against The Stack V2 to facilitate accurate model evaluations. Our results revealed that non-permissive code is also present at the near-duplication level, although, this represents a gray area in terms of legal interpretation, where the boundary between acceptable reuse and license violation is still unclear, emphasizing the need for further legal clarification. Finally, we extend on this and introduce The Heap, a large multilingual copyleft dataset covering 57 programming languages, specifically deduplicated to avoid contamination from existing open training datasets. The Heap offers a solution for conducting fair, reproducible evaluations of large language models without the significant overhead of the data curation process.

Preface

This thesis marks the start of my research journey into AI for Software Engineering. My first real exposure to this field came during my bachelor's thesis, which piqued my interest and motivated me to further explore it. I realized through this experience how essential high-quality data is for large language models performance and discovered that there are still many unexplored areas. My involvement in datasets and benchmarking for large language models provided me with the opportunity to co-author two papers [1, 2], which serve as the foundation for this thesis. Conducting this research has been both a challenging and rewarding experience, pushing me to deepen my knowledge in this area, while working alongside incredibly talented and dedicated people. I hope this work adds something meaningful to the field of AI for Software Engineering and serves as a valuable foundation for future research.

Acknowledgements

Without unnecessary formalities, I want to share a sincere message to those who have supported me and stood by my side throughout this thesis—people I deeply admire, respect, and look up to.

First off, I want to express my deepest gratitude to my thesis advisor, Prof. Dr. Arie van Deursen, for his invaluable support and for believing in me every step of the way.

I also want to share my heartfelt appreciation to my daily supervisor, Asst. Prof. Maliheh Izadi, for her endless support, and simply for being such a genuine and inspiring person. As the Romanian saying goes, 'Omul sfințește locul', meaning it is the people who bring true meaning and value to any place.

I would also like to thank PhD Jonathan Katzy for his constant support, patience, and guidance throughout our collaboration. I consider him not only a colleague, but also a friend, and I am glad to have the opportunity to continue working together.

Finally, I want to express my deepest appreciation to my parents and those close to me for their unconditional support throughout my studies. I know what you have been through to support me on this journey, and your support has never stopped for a second. I cannot thank you enough for that.

Contents

1 | Introduction

Large Language Models (LLMs) have recently taken center stage in research, revolutionizing a wide range of fields with their versatility. Their groundbreaking capabilities have caused a rift among industries, making their adoption essential to keep pace with innovation. The introduction of the Transformer [3] architecture marked a turning point in the domain of Natural Language Processing (NLP), surpassing traditional deep learning models in various natural language understanding tasks. The self-attention mechanism, along with parameter scaling, extensive training data, and higher computing power, played a crucial role in the shift from traditional language models to large language models [4]. These models not only demonstrate improved generalization across tasks, but also excel in transfer learning [5]. Their flexibility has driven progress across new research paths, with LLMs being able to process the complex structure of programming languages.

State-of-the-art models, such as StarCoder 2 [6], Qwen2.5-Coder [7], and CodeGemma [8], have set new records in code generation, outperforming previous models on popular benchmarks including HumanEval [9] and MBPP [10][11]. Furthermore, the growth in parameters contributed to the development of emergent capabilities in LLMs, such as in-context learning, structured reasoning, and instruction following for better representation of human intent [12, 13]. Such advancements have also been brought into industrial environments through tools such as GitHub Copilot[1] or Amazon CodeWhisperer[2], automating code-related tasks and assisting developers throughout the software development cycle [14, 15]. Lastly, the current AI trend is moving towards agents with strong reasoning capabilities, such as GPT-4o [16] and DeepSeek R1 [17], which are trained using reinforcement learning. These models combine logic, decision-making, and adaptability to improve their problem-solving skills. They generate a detailed internal thought process before providing an answer, allowing them to excel in complex tasks, such as coding.

The fuel that powers all these models lies in their datasets, which represent the primary source of knowledge and context for model learning. However, the quality and integrity of these datasets are often overlooked, leading to significant challenges in the development of LLMs, including reduced model performance, compromised generalizability, and biased assessments. The sudden expansion of LLMs also created the need for extensive amounts of data, which, together with inconsistent curation practices, makes it increasingly difficult for researchers to compare results, build upon existing work, or replicate findings with confidence. Many open

[1] GitHub Copilot
[2] Amazon CodeWhisperer

code datasets, ranging from method-level data to entire repositories, have been released to support public LLM development. Despite their immense potential, numerous data smells can be found in these datasets, primarily due to the automated collection methods employed and the lack of data pre-processing [18].

Although not typically referred to as a data smell, a significant yet often overlooked issue arises from the legal aspects tied to LLMs, particularly concerning the data they are trained on. As the size of these models has grown, there has been a notable shift towards adapting them for end-user applications, increasing the interest of companies seeking to leverage them for commercial purposes. However, most datasets these models are trained on are automatically sourced from public repositories on GitHub, without thorough verification for licensing or copyright status. This lack of legal oversight exposes model developers and organizations to potential liabilities from license violations, especially when copyrighted or restricted code is unintentionally included [19, 20, 21]. On top of that, the practice of reusing model weights can introduce further risks for end-users. These challenges not only complicate the legal landscape of AI development but also impact the trustworthiness of LLMs.

Furthermore, the extensive scraping of data led to the development of large-scale training datasets that cover almost all publicly available code [22, 23, 24]. This makes it difficult to evaluate models on downstream tasks using unseen data, due to the high risk of contamination. This also represents a widespread data smell, due to the tendency of developers to often copy code snippets from different projects [25], which can have a major impact on model performance. At the same time, this issue contributes significantly to potential license violations. Additionally, file duplicates not only inflate dataset size and waste computational resources, but can also obscure the validity of the evaluations [26], making it harder to analyze and explain the factors driving model predictions. To fairly assess the performance of LLMs on downsteam tasks, fresh data not seen during training is needed. Otherwise, contaminated data can reinforce the memorization of certain patterns [27], causing overfitting and reducing the ability of the model to generalize, leading to overly optimistic results. In fact, a recent study shows that only 10% of LLM-related investigations deduplicate their data to prevent contamination with the training data [18].

The dominance of public code datasets and the issues related to data contamination also have a significant impact on the reproducibility of LLM evaluations. To assess models on new data, researchers have to redundantly reproduce curation steps such as scraping, filtering, and deduplication across large datasets, requiring substantial time and computational resources. This process can introduce inconsistencies across different implementations, making it harder to ensure reproducibility and compare results between studies. Additionally, many public datasets lack a clear and transparent datasheet [22, 28], further complicating implementation consistency and limiting their usability in LLM studies.

The integrity of LLMs depends on the quality of their data, and issues at this level can disrupt both academic and production environments. While untraining these models is an option, it would require a significant investment of computational power and time. This work seeks to address these issues through a comprehensive licensing analysis and the development of robust datasets that support responsible data handling and facilitate reliable and reproducible LLM evaluations.

To guide this effort, we focus on the following research questions:

RQ1) ***What is the extent of code license infringements in the training data of contemporary LLMs?***

RQ1.1) *How has the inclusion of source code in the training of both generic and specific language models evolved over time?*

RQ1.2) *What is the minimum level of existing strong copyleft-licensed code in the training data of LLMs?*

RQ1.3) *What types of sensitive information can be found in the datasets of LLMs?*

RQ2) ***What is the duplication depth of non-permissive code in LLM training datasets?***

RQ3) ***How can we construct a code dataset that facilitates reliable evaluations on unseen data, supports reproducibility, and simplifies integration into LLM studies?***

First, we conduct a licensing investigation on 53 foundation models trained on file-level code from 6 widely-used public datasets. We evaluate the minimum level of strong copyleft code in these datasets, finding at least 5% overlap across all of them. We also examine the presence of comments indicating licenses or distribution disclaimers, with values reaching up to 15% in some datasets. Building on this licensing analysis, we explore the depth of non-permissive code in The Stack V2 [23], due to its recent release and open availability. To do this, we create StackLessV2, a strong copyleft Java dataset, deduplicated against The Stack V2. Our results reveal that non-permissive code presence in The Stack V2 is more extensive than just exact duplicates, with near duplicates also being observed. Lastly, we expand on this by developing The Heap, a large-scale, multilingual copyleft dataset covering 57 programming languages and decontaminated against various public training datasets, aimed at facilitating reliable and reproducible model evaluations. By tackling the challenges associated with data quality, we aim to strengthen the foundation of LLM development, creating a path for more transparent, reproducible, and legally compliant AI models.

This work makes the following key contributions:

- We provide an in-depth examination of how source code is used as a data source in modern LLMs.
- We compile a comprehensive overview of the code datasets currently used in training LLMs.
- We investigate the potential risks related to copyright and licensing for LLMs, focusing on publicly available code datasets.
- We release a dataset[3] consisting of opening comments from 171 million code files, created to help identify copyright and licensing issues in future studies.

[3]Leading Comments Dataset

- We introduce StackLessV2[4], a strong copyleft Java companion dataset for The Stack V2, designed to facilitate evaluations on unseen data.
- We develop The Heap[5], a multilingual copyleft code dataset, deduplicated against popular training datasets to support valid and reproducible LLM evaluations.

Finally, this paper is organized as follows: Chapter 2 provides an in-depth review of the existing literature on data licensing and contamination, and also describes integrity issues encountered in existing training datasets. Chapters 3 and 4 present our proposed approach and the results corresponding to each research question. Chapter 5 discusses the implications of our findings, offering recommendations and addressing the limitations of the study. Finally, Chapter 6 concludes the paper and outlines directions for future work.

[4] StackLessV2
[5] The Heap

2 | Related Work: Licensing, Contamination, and Integrity Issues in LLM Datasets

The landscape of pre-training datasets has flourished since the introduction of LLMs. The upward trend in LLM scaling not only created the need for more data, but also diversity of data became of outmost importance due to the nature of their architecture capabilities. This led to the emergence of datasets such as Massive Text [29] that surpassed one trillion tokens in size across a wide variety of sources, from web pages to books, news articles, and code. Apart from that, a plethora of exclusive file-level code-related datasets have been released, however, a lot of them are either restricted to the public due to various reasons or can be found behind a paywall[1]. On the other side, the community has been overpowerd by large-scale file-level code datasets, such as The Stack family [22, 23]. These datasets have reached impressive sizes of over 3B files across over 600 programming and markup languages [23], overshadowing smaller and less diverse datasets such as CodeParrot [30].

Consequently, the rapid expansion of data collection shifted the focus towards prioritizing quantity over quality, and led to several shortcomings in the process. These problems can be labeled as data smells, which typically emerge from violating best practices in data handling and management, and can introduce significant technical debt [31]. Fixing such problematic instances was showed to improve model effectiveness [32, 18], and more recent efforts are being made to create higher quality data. Nevertheless, researchers still tend to overlook the significance of thorough data curation, which is essential for tackling the growing prevalence of data smells in datasets as software continues to evolve. This was showed by Vitale et al. who created a catalog of data smells specifically targeting coding tasks, based on the datasets used in pre-training and fine-tunining LLMs, comprising a total of 71 types of smells. Furthermore, the proeminence of these smells was also shown in popular benchmarks, such as CodeSearchNet, raising concerns about the validity of the evaluation techniques employed [18, 33]. In contrast, mainstream pre-training datasets such as The Stack [22], The Stack V2 [23] or GitHub-Code [34] have received little attention when it comes to comprehensively investigating the presence of such data smells, leaving them susceptible to different quality issues [18]. Due to their large scale, pre-training datasets are more likely to contain data smells compared to smaller benchmarking datasets, which can significantly impact the performance of LLMs. As a result, when the foundational understanding of language is compromised, the capacity of a model to handle downstream tasks is severely limited.

[1] BigQuery

2.1 Legal Dimensions of Data

The growing prominence of large language models for code (LLMs4Code) has captured the attention of companies eager to leverage them for commercial end-user applications. This, however, raises questions about the legal implications of incorporating copyrighted data into large-scale pre-training datasets. The importance of using permissive licenses in training LLMs has been acknowledged by scientific initiatives such as The BigCode Project [22] and Together Computer [24]. These organizations have released datasets, such as RedPajama, The Stack and The Stack V2, which they claim consist of permissively licensed or license-free code. Additionally, models such as SantaCoder [35], StarCoder [36] and StarCoder2 [23], trained on these datasets, exemplify efforts to ensure compliance with licensing considerations in LLM development.

The root cause of legal concerns stems from the extensive data scraping carried out without acknowledging ownership rights [37]. Due to that, many legal disputes have surfaced, focusing on allegations of financial losses caused by the fraudulent use of copyrighted data in training or fine-tuning LLMs, as well as the manipulation of copyright information to build a competing business [19, 20, 21]. Moreover, companies have voiced concerns over reputational risks, particularly when their name is associated with misleading information. In some instances, such organizations have demanded the deletion of model weights that were trained on their data, which can lead to significant financial consequences for their creators [21].

On top of that, LLMs exhibit several vulnerabilities closely tied to the legal field, including susceptibility to membership inference attacks (MIAs) and memorization [38, 39, 40, 41, 27]. These problems can lead to the detection of copyrighted data in pre-trained and fine-tuned models, while distributing these models might be seen as redistributing copyrighted content.

2.1.1 Membership Inference on Code

Membership inference attacks, a widely researched topic on traditional machine learning models [41], is a type of attack through which we can determine whether a piece of data has been seen during the training of a model. This represents a key step in identifying potential license infringements, especially for proprietary models that do not disclose their training data. Membership inference attacks have been applied to various tasks, from text generation to image classification and audio recognition, highlighting potential privacy risks [41]. Recently, researchers have focused on using membership inference attacks to extract training data from LLMs. For code models, membership inference can be classified into two distinct categories: representation-generating models and output-generating models.

Under the first category, the BUZZER [42] framework was proposed. This approach is tailored for the code membership inference (CMI) task and deploys several inference techniques, including distilling the target code pre-trained language model, ensemble inference or unimodal and bimodal calibration. The authors seek to detect unauthorized code in pre-training data of models including CodeT5+ [43], UnixCoder [44], CodeBERT [45] and GraphCodeBERT [46], which can cause potential intellectual property infringements. They apply three settings: white-box, gray-box, and black-box. In the white-box setting, BUZZER has access to the internal states of the model. In the gray-box setting, BUZZER can access the internal states of a shadow model. In the black-box setting, BUZZER has no access to any internal states. The authors

showed that BUZZER achieved a 90% accuracy in detecting unauthorized code in pre-training data in the white-box setting, around 80% accuracy in the gray-setting, and approximately 60% accuracy in the black-box setting.

For the output-generating category, Gotcha [47], a novel code membership inference attack, was proposed to analyze potential data leakage in CodeGPT [48]. This approach simultaneously considers three factors, including model input, model output, and ground truth. In this study, surrogate models were trained to replicate the behavior of CodeGPT. A classifier was then used to identify whether specific data was included in CodeGPT's training dataset. Various surrogate model architectures were tested to assess the effectiveness of the attack based on the attacker's level of knowledge about the target model. CodeGPT was evaluated with full knowledge of both its architecture and training data. GPT-2 was assessed with only the architecture known, while the training data remained unknown. Lastly, the Transformer was tested with knowledge limited to the general type of architecture, without specific details, while LSTM was evaluated without any knowledge of either the architecture or the training data. When the model architecture was known and 20% of the training data was available, Gotcha achieved exceptional results, with an error rate of 10% and an AUC of 0.98%. These results highlight the susceptibility of code models to membership inference attacks in code generation tasks. These findings are further validated by BUZZER's high accuracy, denoting the reliability of both approaches.

2.1.2 Memorization

While research on membership inference attacks for LLMs4Code is still emerging, the issue of preventing models from generating memorized code has received significantly more attention.

Memorization is often defined in various ways; nonetheless, a common ground is the emphasis on finding overlaps between model outputs and substrings of the dataset. Some papers focus solely on exact duplicates of outputs [40], while others search for close matches, and some introduce constraints on the minimium length of a substring for it to be considered memorization when produced by a model [27]. Furthermore, it has been demonstrated that, in code memorization, the number of parameters in a model is correlated to the level of memorization [39, 40]. Carlini et al. also showed that memorization grows with the number of times an example has been duplicated and the number of tokens of context used to prompt the model. Moreover, as models continue to grow in size at a rapid pace, the memorization of training code data is becoming an increasingly significant concern. Although the largest models are typically not open-sourced, some studies investigate how quickly models memorize code when the training dataset is accessible.

Yang et al. have found that 81% of the top-100 outputs generated by Incoder [49] are directly copied from GitHub, while 75% of the outputs from StarCoder [36] are also taken from GitHub. Moreover, when evaluating the open-source CodeParrot[2] model, they showed that the repetition of code in the dataset, multiple queries to the model, and longer output sequences increased the likelihood of returning memorized code. Additionally, they demonstrated that, for the same architecture, larger models tend to exhibit higher memorization. This was evident in CodeParrot, where approximately 57% of $20,000$ outputs contained memorized information

[2] CodeParrot

on average, compared to 43% for CodeParrot-small. The authors also created a taxonomy of memorized outputs in code models, the most common type being license information. Although detecting exact memorization might seem straightforward and could be addressed with filters, Copilot, based on the Codex [50] model, manages to evade its own memorization filters [51], giving a false sense of privacy. This creates the illusion that the model is generating original content, even though it does not address the underlying issues with LLMs.

2.1.3 Intersection with Law

Under the fair use doctrine, copyrighted material can be utilized to train LLMs without legal consequences. However, this protection may not apply if the model generates outputs that closely resemble copyrighted data, particularly when such outputs could harm the market of the original content [52]. Henderson et al. showed that widely used foundation models are capable of producing content that closely resembles copyrighted material. The authors conducted an analysis to extract examples of code, under the General Public License (GPL), using Codex-based models. They examined the average match rate of the code-cushman-001, code-davinci-001, and code-davinci-002 models, which was approximately 50% when provided with Linux kernel function signatures as prompts. They further explored problems with language models that remove required copyright information when using code from a file.

To address the issue of copyright violations, the authors propose several technical solutions. First, they recommend selecting training data based on the license assigned to each file, similar to practices used in certain datasets [22]. They also emphasize the importance of focusing on data quality, suggesting the removal of duplicate data, which has been shown to contribute to increased memorization in models [27]. Finally, they propose adding filters to model outputs, although this may lead to overconfidence in the model's ability to generate original content, as previous studies have shown [51].

2.1.4 Lawsuits

To explore the concerns data owners face regarding the use of their content in training datasets, we provide a concise summary of major legal disputes. These include the controversies surrounding the Books3 corpus, the lawsuits over Stable Diffusion models generating copyrighted images, and the legal battle between the New York Times and OpenAI over the incorporation of New York Times articles in model training.

Books3 Corpus Battle. Books3 was a dataset comprising approximately $200,000$ books scraped from online sources that distribute copyrighted content, hosted by a website called The Eye. The dataset was initially designed to reduce the disparity in resources available to large AI corporations and independent researchers, offering individuals working on LLMs an opportunity to compete on more equal footing. This was especially targeting companies such as OpenAI, that trained their models on proprietary datasets called Books1 and Books2 [53]. Books3 was included into a larger dataset that merged existing collections to form an extensive corpus, which was then released as The Pile [54]. Several companies, including Bloomberg, EleutherAI, and Microsoft, utilized this dataset to train commercial LLMs. In the United States of America, a

group of authors whose works were included in Books3 filed a lawsuit demanding that these companies cease using their books permanently, together with compensation for the use of their works [20]. The lawsuit centers on allegations that Bloomberg, Microsoft, and Meta:

- Developed LLMs aware that the training data was copyrighted.
- Failed to attain licenses for the copyrighted materials, knowing that the original works were obtained illegally.
- Chose to use the stolen content to train models for commercial gain.

The Danish organization Rights Alliance has filled a DMCA takedown request for The Pile dataset due to its inclusion of the Books3 corpus, resulting in the entire dataset being removed from download [55].

StableAI vs. Getty Images. In another legal dispute currently unfolding in the United States of America, Getty Images, a digital image provider, accuses StableAI of copyright infringement, claiming that the company used 12 million of its curated images to train the Stable Diffussion model [19]. The lawsuit highlights three key points:

- StableAI's models occasionally produce images containing the distinct Getty Images watermark, which would breach trademark protections.
- The images used for training were scraped from Getty Images' website, violating the website's licensing terms.
- Getty Images was not contacted for permission to use their images, which the company carefully curates, including adding captions, titles, keywords, and other metadata.

OpenAI vs. The New York Times Another significant ongoing lawsuit involves The New York Times, OpenAI, and Microsoft, and focuses on the use of articles owned and copyrighted by The New York Times in the training of LLMs distributed by the two software companies [21]. Similarly to earlier cases, the dispute revolves around the inclusion of copyrighted content in the datasets used to train AI models. The main concerns raised by The New York Times in this case can be summarized as follows:

- The training process for these models relies on copyrighted content produced by The New York Times, which was used without obtaining a license.
- By generating articles on similar topics, the models potentially decrease the incentive for individuals to subscribe to The New York Times.
- The models occasionally hallucinate 'facts' and attribute them to The New York Times, damaging the reputation of the publication.
- These models are capable of reproducing exact copies of New York Times articles, providing access to people without a subscription.

While similar to concerns raised in the Getty Images and Books3 cases, the New York Times lawsuit takes a deeper look at the technical flaws of modern LLMs and their repercussion for those supplying training data.

One of the primary issues highlighted is the tendency of AI models to hallucinate 'facts'. The lawsuit includes screenshots as evidence, showing that Bing Chat often generates responses unrelated to actual New York Times articles when asked for specific content. It also demonstrates how the model falsely attributes quotes or citations to individuals, implying they were mentioned in the publication when they were not. Additionally, Bing Chat has been shown to reference articles that do not exist, claiming they were published by The New York Times.

Furthermore, the lawsuit demonstrates how AI models replicate content from their training data by providing examples of GPT-4 outputs that closely match text published by The New York Times. It also includes instances where GPT-4 generated almost identical copies of articles after being prompted with statements about paywall restrictions.

Unlike the earlier lawsuits, there were attempts at negotiations between The New York Times and OpenAI over licensing the use of their articles, but no agreement was reached. OpenAI's defense against the copyright infringement is based on the argument that using articles to train models is transformative and falls under fair use. They also assert that The New York Times violated OpenAI's terms of service by using unauthorized methods to retrieve their articles from the GPT-4 model. Lastly, The New York Times is not only seeking compensation for the harm they claim to have suffered due to Microsoft and OpenAI, but is also demanding the deletion of all model weights trained on datasets containing their articles, along with the destruction of the datasets themselves.

2.1.5 Code Licensing

The code used in pre-training and fine-tuning LLMs is governed by an extensive range of licenses that impose different constraints on the use and distribution of software. These licenses can be classified into two main categories, namely free licenses and non-free licenses. Since this study focuses on free licenses, we will not delve into the specifics of non-free ones. Free licenses can be further categorized into public domain licenses, permissive licenses, and copyleft licenses[3]. Given the broad scope of these licenses, our focus is mainly on permissive licenses, as well as weak copyleft, strong copyleft, and network copyleft licenses within the copyleft family [56].

Permissive Licenses. Permissive licenses, such as Apache, BSD, and MIT, are commonly used in software projects due to their flexibility. This type of license grants users significant freedom to use, modify, and redistribute the software with minimal restrictions. This adaptability makes them suitable for integration into both proprietary and open-source projects. Modifications and derivative works can be made without the obligation to release them under the same license [57, 56]. Permissive licenses accommodate projects with diverse license requirements, enabling users to apply a different license to the modified code or even keep it proprietary. Unlike copyleft

[3]License Types

licenses, permissive licenses primarily require a copyright notice and disclaim liability for the original author.

Weak Copyleft Licenses. As their name suggests, weak copyleft licenses, such as MPL, LGPL or CECILL, permit the inclusion of code in proprietary projects without necessitating the entire derived work to be open-source [56, 58]. In essence, only the changes made to the original code must be released under the same weak copyleft license. These licenses strike a balance between proprietary software and open-source collaboration. In addition to requiring the sharing of modified code, weak copyleft licenses often require giving credit to the original authors and may include a liability disclaimer.

Strong Copyleft Licenses. Strong copyleft licenses, including GPL or QPL, differ from weak copyleft in that they require any derivative work that modifies or incorporates the original code to be distributed under the same strong copyleft license. This 'share-alike' condition guarantees that the open-source rights remain intact in all future versions and derivatives of the software [56, 58]. Moreover, these licenses typically require attribution of the original authors and include a liability disclaimer to protect creators from responsibility for issues arising from the use of the software.

Network Copyleft Licenses. Network copyleft licenses, such as AGPL, APSL or OSL, extend the requirements of strong copyleft licenses, having a larger applicability. This type of license not only requires the distribution of entire derived work under the same license, but also requires compliance when the software is operated for others to access over a network [56, 58]. They often require attribution of the original author and the inclusion of a liability disclaimer.

2.2 Data Contamination and Its Remedies

Data duplication occurs when identical or nearly identical pieces of information occur repeatedly within a dataset [26]. Data duplication can be divided into two categories: intra-dataset duplication, where similar or identical data exist within the same dataset, and inter-dataset duplication, where similar or identical data is present across different datasets [59]. The latter scenario represents the foundation of data contamination, also known as data leakage. In the context of LLMs4Code, data contamination mainly involves code overlap across the training, validation, or testing datasets, violating their intended isolation. This overlap has been shown to cause inflated evaluation metrics and biased outcomes [18, 26]. Yet, data contamination continues to be a persistent challenge. This is largely due to the constantly evolving nature of software and the common practice among programmers of reusing code snippets [25], which makes it increasingly difficult to gather data free from contamination. Although its consequences are well known, many LLM studies fail to deduplicate the data used in pre-training and fine-tuning their models, which compromises the construct and external validity of their evaluations and limits the ability of the models to generalize effectively on downstream tasks [18].

2.2.1 Code Duplicate Types

In the literature, code duplicates are often used interchangeably with code clones, a widely-researched topic in software engineering [60, 61]. Although closely related to code clones, code duplicates are generally viewed as a subset of them [26]. However, the distinction between the two is not well-defined, as it largely depends on the specific implementation and similarity function employed. Code duplicates can be classified as follows:

- *Exact Duplicates:* Refer to code snippets that are syntactically identical.
- *Near Duplicates:* Include code snippets that are syntactically identical or partially identical.
- *Semantic Duplicates:* Describe code snippets that are semantically identical or partially identical.

Exact duplicates most closely align with Type-1 clones, while near duplicates correspond to Type-2 clones and may even extend to Type-3 clones [60]. Semantic duplicates, on the other hand, are more comparable to Type-4 clones. However, the definition of ”partially identical” is highly subjective, and depending on the chosen implementation, the scope of duplicates can either broaden to cover multiple clone types or be refined to a more specific subset. Additionally, code duplicates can occur at different levels of granularity, which can be classified into different categories [62]:

- *Chunk-Level*: This refers to duplicates that appear within smaller code fragments, such as lines, blocks, or even functions.
- *File-Level*: This occurs when duplicates are found across entire code files.
- *Repository-Level*: This involves duplicates that exist between entire code repositories, covering all the code files within them.

2.2.2 Code Copies in the Wild

As the reliance on extensive collections of open-source projects grows, the effects of code duplication are becoming more and more noticeable [26]. Lopes et al. performed an analysis of 4.5M non-fork GitHub-hosted projects, including over 428M files written in Java, JavaScript, Python, and C++. Their findings revealed that only 85M of these files were unique, indicating that 70% of GitHub’s code base consists of file-level exact and near duplicates. Furthermore, duplication rates varied significantly across programming languages. JavaScript exhibited the highest level of file duplication, with only 6% of files being unique, whereas Java had the lowest, with 60% of files being distinct. Additionally, their study found that between 9% and 31% of the projects had at least 80% of their files duplicated elsewhere.

In another study, Golubev and Bryksin analyzed over $23,000$ open-source Java projects from GitHub, examining clones at both method- and file-level. Their investigation covered exact and near-miss clones while also assessing the size and age of duplicated code fragments. Their findings indicate that code copying has persisted throughout the history of Java development.

Additionally, method-level cloning was found to be significantly more common than file-level cloning, with only 35.4% of methods remaining unique and free from clones.

Similarly, Ossher et al. investigated the prevalence of developers copying entire files or groups of files into their applications. Analyzing over 13,000 Java projects from various open-source repositories, their method identified more than 10% of files as clones and found that over 15% of all projects contained at least one cloned file. Through manual inspection of reported clones, they observed that most duplication involved Java extension classes and widely used third-party libraries. Projects appearing across multiple repositories were often the result of forking or being split into smaller sub-projects. Along the same lines, Gharehyazie et al. examined cross-project cloning by analyzing over 5,000 Java projects scraped from GitHub. Their findings align with Ossher's study, showing that cross-project clones account for 10% to 30% of all code clones within projects. Additionally, they observed that code cloning follows an onion-like pattern: most clones originate within the same project, followed by projects in the same application domain, and finally from projects in entirely different domains.

Lastly, Yang et al. explored the overlap between code snippets shared on Stack Overflow and those found in GitHub projects. They examined 909,000 non-fork Python repositories on GitHub, containing 290M function definitions, alongside 1.9M Python snippets from Stack Overflow. While their qualitative analysis indicated that exact duplicates between the two platforms accounted for less than 1%, they identified a significant number of near duplicates. Specifically, 405,000 distinct GitHub code blocks closely resembled snippets from Stack Overflow, and 35,000 unique Stack Overflow code blocks had counterparts within GitHub.

Regardless of the level of granularity at which clones or duplicates are examined, these studies share a common aspect: the presence of copied code within open-source repositories, primarily from GitHub. Collectively, they raise concerns about projects and research relying on such data, as duplication can negatively impact their results. In the context of LLMs, GitHub serves as a primary source of code for training and fine-tuning models, introducing several challenges, including potential license violations, privacy risks, and biases that can alter evaluation results and ultimately compromise their validity. Finally, the high occurrence of duplicates, driven by efforts to collect more data, makes the evaluation reproducibility increasingly demanding, due to the constant need of data deduplication.

2.2.3 Data Duplication Impact on LLMs

The widespread impact of code duplication has also been observed in machine learning (ML) models. One of the pioneering studies by Allamanis examined how the overlap between training and test splits in code datasets affects model performance. The results indicated that the reported performance metrics were occasionally inflated by up to 100% when tested on duplicated code corpora, in comparison to the performance on deduplicated corpora.

Similar effects have been noted in LLMs, raising concerns about the models' true capabilities and their generalizability. Hernandez et al. explored the implications of training transformer models on data which is mostly unique, however, a small fraction of it is highly repeated. The authors observed a strong double descent phenomenon, where repeated data led to a significant increase in test loss midway through training, causing a severe degradation in model performance. They demonstrated that by repeating just 0.1% of the data 100 times, while keeping the rest of

the training tokens unique, the performance of an 800M model could degrade to that of a model half its size. This data repetition disproportionately harmed structures related to generalization, such as induction heads, which may explain the shift from generalization to memorization.

In addition, many recent studies have showed that various LLMs are contaminated with data from popular code benchmarks, which affects the validity of their evaluations [69, 70]. The findings of LiveCodeBench [71] indicate that, models such as DS-INS-1.3B, DS-INS-6.7B, CodeQwen, and MC-6.7B achieved higher results on HumanEval+, due to possible contamination, compared to Gemini-Pro, Mistral-L or Claude-2. Consistent with these findings, Riddell et al. showed that 3.6% to 20.8% of HumanEval and MBPP solutions can be found in popular pre-training datasets including The Pile and The Stack, by looking at both surface and semantic level similarity. This was showed to inflate the results of models trained on this data, including StarCoder-15.5B, Pythia-12B or CodeGen-NL-16B. These results are also supported by Yang et al. who discovered that 8-18% of the HumanEval benchmark overlaps with RedPajama and The Stack, by looking at rephrased entries. Similarly, StarCoder was found to be contaminated with code from the Defects4J benchmark. Specifically, 35% of test cases and 39% of buggy methods from the benchmark were included in the model's training data, along with code from the 2.0 version of the benchmark [74].

Finally, Singh et al. pointed out the contamination between The Pile, and HumanEval, MBPP, and BigBench benchmarks, using metrics such as token-match, n-gram match or longest-match. They also showed that these benchmarks can also be found in the LLama 1 corpus, with an average contamination rate of over 50%. As a result, models trained on leaked benchmark data risk overfitting to its patterns, which can degrade performance when evaluated on different tasks [76].

2.2.4 Deduplication Techniques

Deduplication is a crucial step in any data pre-processing pipeline, which ensures that duplicate entries are removed from the corpus. This process can be categorized into three types: exact deduplication, near deduplication, and semantic deduplication. In this study we focus on the first two.

For exact deduplication, computing hash fingerprints of file contents is a widely used approach due to its efficiency, scalability, and deterministic nature [22, 23, 35]. It enables fast, memory-efficient comparisons, making it well-suited for large datasets. Since identical inputs produce the same hash, duplicates can be easily identified and removed. However, the performance of this method is highly dependent on the hash function employed.

Furthermore, MinHash Locality Sensitive Hashing (LSH) has proven to be an efficient method for detecting near duplicates [22, 23, 35]. The process begins by decomposing the input into a set of n-grams, also known as shingles. These shingles are then transformed into MinHash signatures, which serve as compact representations of the input that minimize memory usage [77]. To create a signature, each shingle is hashed using a hash function. This process is repeated for a set number of permutations. For each permutation, the minimum hash value is selected from the shuffled shingles, ultimately resulting in a fixed-length vector that forms the MinHash signature. The number of matching values between two signatures approximates the Jaccard Similarity of the sets from which the signatures were derived. Then, LSH groups

similar items by splitting MinHashes into bands, hashing them, and placing items with matching hash values into the same bucket. MinHash and LSH improve efficiency and scalability for large datasets by limiting comparisons to items within the same bucket [78]. Nonetheless, it is important to note that the effectiveness of this method is strongly influenced by the chosen parameters, emphasizing the trade-off between efficiency and accuracy.

2.3 Reliable Data for Reproducible and Valid LLM Evaluations

Despite efforts of popular pre-training datasets to promote open data and accelerate progress [22, 23], their dominance over GitHub-sourced code raises concerns about the reproducibility and accuracy of LLM evaluations. Since many LLMs are pre-trained on these datasets, assessing their performance on new unseen data for downstream tasks becomes challenging due to the high risk of contamination. On top of that, data integrity issues in these pre-training datasets, together with insufficient dataset documentation, can also negatively impact the reproducibility and validity of LLM research [79].

```
{
   text: "<!DOCTYPE html> <!--[if IE 8]><html class="no-js
       lt-ie9" lang="en" > <![endif]--> <!--[if gt IE
       8]><!--> <html class="no-js" lang="en" > <!--<![
       endif]--> <head> <meta charset="utf-8"> ...",
   meta: {
       repo_name: "metpy/MetPy",
       path: "v0.7/api/generated/metpy.calc.
           relative_humidity_from_mixing_ratio.html",
       language: [{"name": "Jupyter Notebook", "bytes": "
           989941"}, {"name": "Python", "bytes": "551868"}]
   }

}
```

Figure 2.1: Example of language misalignment in RedPajama

For example, RedPajama[4] provides access to URLs for downloading the dataset, which is not pre-filtered or split by programming language. This means researchers must filter the data themselves. While the metadata lists all languages present in each repository, it does not specify the language of each file. As a result, researchers must parse the file paths from the metadata and filter based on file extensions. This can introduce variability in the reproducibility of studies, depending on the extensions selected and the fact that the same extension can be shared across different languages. We give an example of such a case in Figure 2.1, where HTML is not even included in the list of languages.

Similarly, CodeClippy[5] contains incorrect file names, which can cause mismatches between its content language and the file extensions. Moreover, there are a lot of instances where the

[4]RedPajama Structure
[5]CodeClippy File Issue

```
{
   text: ".ol-translation-container {\n  padding: 20px;\n}\
       n\n.ol-translation-group {\n  display: block;\n
       position: relative;\n  margin: 0 auto;\n  margin-
       bottom: 30px;\n  background-color: #ddd; ...",
   meta: {
       file_name:"app.js",
       repo_name:"openl10n\/openl10n-app",
       repo_language:"",
       stars:12,
       mime_type:"text\/plain"
   }

}
```

Figure 2.2: Example of a missing language and wrong file name and extension in CodeClippy

repository language provided is empty. Figure 2.2 demonstrates how the repository language field is missing, and although the file extension suggests this is a JavaScript file, the content is in fact CSS. We also noticed that the same filename and repository metadata are incorrectly repeated across many entries, even though the content of the files is completely different. Figure 2.3 illustrates one such example. These deviations can skew model training and affect the validity and reproducibility of evaluations, even when external tools are used to identify the correct language type, due to their limited accuracy. This is also the case for The Pile[6], which has been removed and re-uploaded, losing information about the programming language of a file.

These issues not only affect the accuracy of results and reproducibility of studies but also highlight the lack of effective data curation techniques employed in existing datasets.

[6]The Pile

```
{
   text: "<!DOCTYPE html>\n<html>\n<head>\n\t<title>Server
       Dashboard<\/title>\n <meta name=\"robots\"content=\"
       all\">\n<meta http-equiv=\"Content-Type\" ...",
   meta: {
       file_name:"script.js",
       repo_name:"patschi\/serverdashboard",
       repo_language:"PHP,HTML,JavaScript,CSS",
       stars:14,
       mime_type:"text\/plain"
   }

   text: "@font-face {\n  font-family: 'Droid Sans';\n
       font-style: normal;\n  font-weight: 400;\n  src:
       local('Droid Sans'), local('DroidSans'), url(fonts\/
       droidsans_v3_400.woff) format('woff') ...",
   meta: {
       file_name:"script.js",
       repo_name:"patschi\/serverdashboard",
       repo_language:"PHP,HTML,JavaScript,CSS",
       stars:14,
       mime_type:"text\/plain"
   }

}
```

Figure 2.3: Example of repeated metadata in CodeClippy entries, with different file content

3 | Proposed Approach: A Licensing Analysis and Robust Code Datasets

In this chapter, we outline the methods we used to address each research question. We detail the steps taken to evaluate the extent of license infringements in open code datasets, which is the central focus of our study. Building on this analysis, we develop a Java dataset to highlight the depth of non-permissive code presence in mainstream code datasets and to support more accurate LLM evaluations. Finally, we expand upon this by creating a large-scale multilingual dataset designed to facilitate reproducible and reliable LLM evaluations.

3.1 Addressing the Legal Pitfalls of Dataset Sourcing

We begin by conducting a tertiary study of recent surveys on LLMs. We then analyze papers, repositories, and blog posts to identify datasets used in pre-training them. After gathering the data, we analyze these datasets to check if they contain any copies of code that are also released under a strong copyleft license. Additionally, we inspect the first comment in each file to identify any other potential sources of confidential information that are being embedded into the weights of the models. A visual overview of our approach is provided in Figure 3.1.

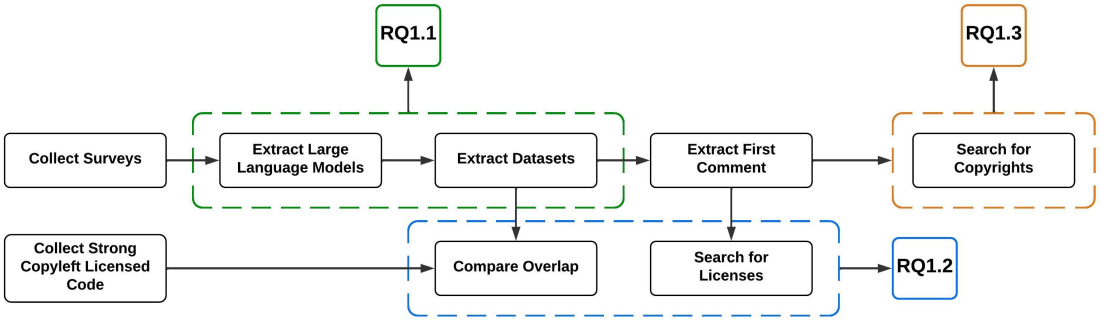


Figure 3.1: Graphical overview of the licensing analysis approach

3.1.1 Study Collection

To understand the scale of licensed code in datasets, it is crucial to first identify the models trained on code. To achieve this, we gather surveys of LLMs from scientific databases, providing a comprehensive view of the datasets frequently used in the literature and the specific models trained on them. We limit our survey search space to only those published in 2023.

After compiling a list of LLMs from the surveys, we gather relevant papers, or blog posts/repositories when papers are unavailable, related to each model. We then filter these sources to identify models trained exclusively on permissively licensed code. First, we remove papers for models not trained on code, and then we remove those not trained on file-level code. We choose file-level code as this refers to code extracted directly from repositories without being altered. Therefore, we prioritize file-level datasets as they are more suitable for identifying duplicate code. We avoid extracting methods or classes as that may lead to false positives, primarily from common components such as getters, setters or common algorithms. Additionally, we exclude models trained exclusively on data from sources such as Stack Overflow or competition platforms. Once the models are filtered, we first evaluate the availability of their training datasets. We then gather all datasets that are publicly accessible and extract details from the corresponding papers or documentation, including the dataset's source and whether it has a specified name. We also include a category for "custom datasets", which are created by the authors but are neither named nor publicly released. These datasets are often scraped from online repositories, but insufficient information is provided to fully replicate them.

3.1.2 Licensing Analysis

Based on our analysis of the data collected, we identify a set of publicly available datasets that include code extracted directly from code bases. To assess whether a code file might be licensed, we examine two key aspects of the dataset.

We begin by generating SHA-256 hashes for all code files, as this method has demonstrated robust performance in identifying exact duplicates within a dataset [22]. We focus solely on exact duplicates because including near duplicates can generate debates over whether a piece of code is truly a duplicate or a slightly variant of a common structure or algorithm [80]. Since the main objective of this paper is to assess concerns about licensed code in datasets, we address this by defining a lower bound on duplication, focusing on exact copies.

Second, we extract the first comment from each file, if there is any. We define this as any comment block that starts within the first 20 characters and may extend beyond that point. This enables us to search for licensing information, copyright notices, and disclaimers regarding content ownership and distribution.

3.1.3 Strong Copyleft Code Collection

A lot of datasets scraped from GitHub focus on including only permissively licensed code. In an ideal scenario, there would be no overlap between code from permissively licensed repositories and those under strong copyleft licenses. To assess the real-world extent of this overlap and whether it varies between datasets that check for code licenses and those that do not, we scrape

our own dataset from GitHub. As discussed, strong copyleft licenses require any derivative work incorporating or modifying the original code to be released under the same license. This is why we focus on these licenses because of their strong commitment to preserving the open-source nature of the code base across all iterations and contributions.

We gather a list of GitHub repositories licensed under GPL-2.0, GPL-3.0, or AGPL-3.0, and created until August 2023, inclusive. We scrape up to $10,000$ repositories per language, selecting those where the primary language matches one of the languages included in our study. We exclude any files written in languages that are secondary within a repository. Lastly, we extracted code files from the repositories based on their extensions without applying any additional filtering by length or content variety.

3.1.4 Investigation Breakdown

To showcase how we detect the extent of code license infringements in open datasets, we present a step-by-step breakdown of our approach across sub-questions.

RQ1.1 - Interest in Licensed Code. To evaluate the extent of licensed code in datasets used for training LLMs, we analyze the data collected during the tertiary study. This includes examining the publication dates, verifying whether the models are trained on code, and checking if they claim the inclusion of permissively licensed code. From this, we can identify trends on an annual basis.

RQ1.2 - Strong Copyleft Infringements. To analyze the presence of strong copyleft-licensed code files in datasets, we generate SHA-256 hashes for all code files in the collected datasets and for code files scraped from GitHub, sourced exclusively from repositories with strong copyleft licenses. This allows us to compare the licensed set of hashes with those in each dataset and measure the overlap based on the number of matching files. Additionally, we extract the first comment from each code file in the collected datasets and search for references to license names, specifically checking for mentions of GPL or AGPL licenses.

RQ1.3 - Distribution Disclaimers. To assess whether the authors of a code file might object to its further distribution, we follow the same approach used for detecting strong copyleft licenses in the first comment. However, we modify the search criteria by excluding the GPL and AGPL boilerplate license declarations and instead focus on language related to ownership and restrictions on sharing. This includes phrases like *confidential*, *please do not share*, and *following conditions are met*.

3.2 Mitigating Data Contamination for Accurate LLM Evaluations

To get a better understanding of the depth of non-permissive code in open datasets, we build upon our licensing analysis and introduce StackLessV2, a strong copyleft file-level Java dataset designed to complement The Stack V2. We take a more detailed approach by also looking for near duplicates between our dataset and The Stack V2. This additional filtering step should

reduce the risk of any additional contamination, making this dataset also a valuable source of fresh data for reliable LLM evaluations on downstream tasks. We choose The Stack V2 as the target dataset due to its recent release and because it is the largest open dataset containing permissive and unlicensed code, on which multiple models have been trained.

3.2.1 Data Collection

We use the same collection process as in the licensing analysis, which is explained in greater detail here. We scrape 10,500 public repositories using the GitHub */search* API, with a search query based on license type, repository star count, and repository creation date. We aim to include 10,000 repositories in the dataset and select an extra 500 to account for repositories potentially being removed between collection and downloading. We only target repositories licensed under strict copyleft licenses including GPL-2.0, GPL-3.0, and AGPL-3.0, where the main language of the repository is Java. Moreover, we select repositories created until April 2024, inclusive, in descending order by star count, as this has been used as a loose quality metric before [35]. Furthermore, for the */search* API, GitHub permits up to 30 authenticated requests per minute for personal accounts. Additionally, due to the resource-intensive nature of search queries, each query is limited to retrieving a maximum of 1,000 results (including pagination). These constraints make extracting a large number of repositories challenging.

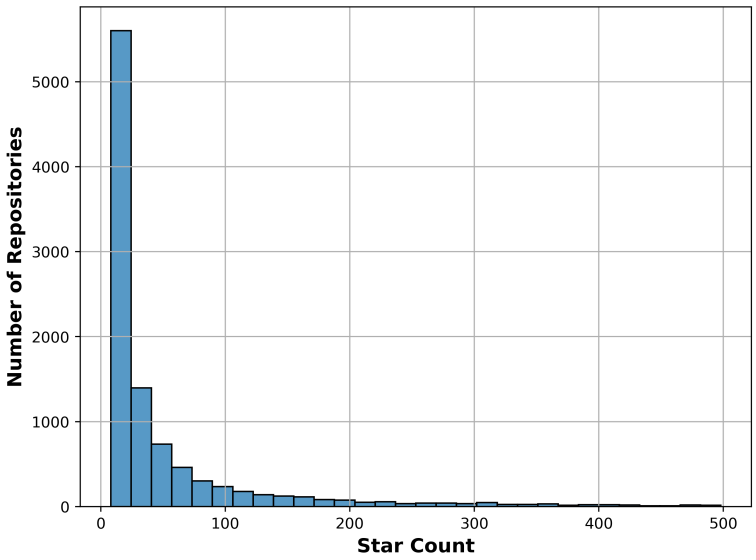


Figure 3.2: Distribution of Java repositories by star count

To prevent hitting request limits, we query the */rate_limit* API to monitor the remaining requests and determine the reset time as we approach the limit. We then apply timeouts, with a small buffer, until the request limit resets. Additionally, we utilize pagination, requesting up to

100 repositories per call, and apply filters based on repository license, star count, and creation date. Instead of retrieving all results in a single response, pagination divides large data into smaller, more manageable chunks, also known as pages. Combined with filtering, this approach not only minimizes the number of requests but also allows us to bypass the 1,000-repository limit per query when more results are available. We perform the filtering as follows.

We retrieve repositories within a specific star range for the entire time frame mentioned above. If there are less than 1,000 repositories for that star range, then we store them and reduce the range. Otherwise, we restart the iteration with the same star range, but this time, we process the time frame month by month using a tumbling window. Then we store the collected repositories and reduce the star range. This enables us to iterate over the time frame in increasingly granular segments, thereby capturing more repositories and avoiding the limit. Since we scrape repositories in descending order by star count, we first retrieve those with more than 900 stars. For repositories between 900 and 100 stars, we divide the range into intervals of 50 stars; between 100 and 10 stars, we use 10-star intervals; and for those with fewer than 10 stars, we treat each star count individually. Given that most repositories fall within the 0–100 star range, as shown in Figure 3.2, filtering by repository creation date and star count helps narrow the search space, allowing us to capture more data in an incremental way. We also eliminate any duplicate repositories that may arise from the pagination process. Finally, we extract the code files according to the the programming language extension list[1] from The Stack.

3.2.2 Data Cleaning

After collecting the data from GitHub, we perform some cleaning steps. Building on the approaches used for The Stack datasets [22, 23], we exclude low-quality training data, including files larger than 50 MB and those with fewer than 10 words, as they may introduce noise and negatively impact model performance. Similarly, we filter out auto-generated files by searching for specific keywords in the first 5 lines of each file. This list of keywords is based on the Stack V2 approach and the manual file inspection we performed: *generated by*, *autogenerated*, *auto-generated*, *this file was generated*, *this file is generated*, *generated automatically*, *automatically generated*.

3.2.3 Data Deduplication

To further reduce the risk of contamination and assess the presence of strong copyleft code in The Stack V2, we implement near deduplication on top of exact deduplication. First, we remove exact duplicates within our dataset, followed by applying exact deduplication against the Java subset of The Stack V2. We use the SHA-256 hash function because it ensures a low probability of collisions and provides a uniform distribution of hash values.

Then, we perform near deduplication between our dataset and the Java subset of The Stack V2 using the MinHashLSH approach from the *datasketch*[2] library. We apply the same hash function, together with 128 permutations and a precision-recall weight distribution of 40%–60%.

[1] Programming language extension list from The Stack

[2] MinHashLSH

These design choices help reduce hash collisions while maintaining a balanced trade-off, prioritizing higher recall with a controlled increase in false positives. Before splitting the files into shingles, we lowercase their content and remove any whitespace. This helps in reducing token variability and formatting differences, which improves recall. Then, we use a shingle size of 7 characters, considering that code files generally contain a smaller set of characters compared to large research articles, where $k = 9$ [78]. This choice helps minimize the occurrence of overly common shingles, which could otherwise artificially increase similarity scores, especially when using a smaller k value. Files with a Jaccard similarity greater than 0.7 are marked as near duplicates, a threshold proven to be effective for detecting duplicates [26].

3.3 Supporting Reproducible LLM Evaluations

We extend on StackLessV2 by creating The Heap, a large-scale multilingual file-level dataset covering 57 programming languages, and multiple variants of copyleft licenses. We deduplicate The Heap against a wide range of open-source training datasets to provide clean data for LLM evaluations and support ongoing research efforts.

3.3.1 Data Scraping

We use the same collection process as with StackLessV2 because it mirrors the data distribution of other large-scale datasets [22, 23]. This approach minimizes the likelihood of including confounding factors [81], such as drifts in data representation. We now scrape up to $50,000$ repositories or as many as are available per language, and we extend our search space until August 2024, inclusive. Since existing datasets already include nearly all open-source code under permissive or unlicensed terms, we use a broader set of copyleft licenses as our initial filter for repositories. Focusing on copyleft licenses for our datasets enables us to gather more unseen data that can be used for evaluating models on downstream tasks.

Moreover, the exclusion of non-permissively licensed code in other datasets is driven by community concerns that models' inferred outputs could potentially violate license terms [22, 82]. Our datasets are designed solely for research purposes and are not meant for pre-training models intended for end users. The use of exclusively non-permissively licensed code also serves as a deterrent for developers from training LLMs on our datasets, helping to preserve their relevance for downstream tasks. Training on such code is unappealing, as it could require the end user to publicly release *all* code in their code base.

3.3.2 Data Curation

As previously stated, the selection of only copyleft licenses is intended to prevent contamination with other open training datasets. However, since code borrowing is a common practice among developers [66], the boundaries of a license can become blurred, potentially leading to license violations. By also employing exact and near deduplication when building our datasets, we aim to minimize the level of contamination and reduce any concerns about data integrity.

We follow the same cleaning and deduplication process as before, but this time we filter out files larger than 10 MB. Additionally, we remove all comments and whitespace according to the

programming language before performing exact and near deduplication. By focusing on code modifications, this approach ensures that minor edits, such as removing a license comment or adjusting whitespace, still allow duplicates to be identified. The final files included in The Heap are the original, unmodified versions scraped directly from GitHub. Lastly, rather than removing all exact duplicates from our dataset, we only eliminate the ones coming from our own files.

4 | Results & Applications: Accelerating LLM Research with Clean Data

This chapter presents the results of our licensing analysis and our findings on the depth of non-permissive code duplicates in open training datasets. Additionally, we outline the structure of our code datasets and the rationale behind our design choices.

4.1 RQ1 - License Infringements in LLM Training Data

We begin by describing the results of our tertiary study, as well as the distribution of strong copyleft repositories collected by programming language. Based on the collected data, we showcase the extent of license infringements in open datasets by addressing each of the sub-questions.

4.1.1 Tertiary Study

We collected 6 literature surveys on LLMs conducted in 2023, which are summarized in Table 4.1. Based on these surveys, we identified 106 LLMs, which we classified into the following categories: 31 models trained exclusively on natural language and 75 models trained on code. Among the code-trained models, 22 were trained on method-level code, 23 on permissively licensed file-level code, and 30 on non-permissively licensed file-level code. The distribution of these models is shown in Table 4.2. Superscripts for the file-level code models correspond to the datasets they were trained on, as indicated by the reference numbers in Table 4.3.

Table 4.1: Literature surveys used to identify LLMs

Title	Reference
Software Testing with Large Language Model: Survey, Landscape, and Vision	[83]
A bibliometric review of large language models research from 2017 to 2023	[84]
A survey of large language models	[13]
Large Language Models for Software Engineering: A Systematic Literature Review	[85]
Large Language Models Meet NL2Code: A Survey	[86]
A Survey on Large Language Models for Software Engineering	[87]

We also identified a total of 14 datasets. Among them, 6 were not released by the authors, 5 were fully open, 1 was selectively released to practitioners who applied, 1 was behind a paywall, and another 1 was taken down due to a DMCA request but later re-uploaded by a third party with the problematic content removed. In addition to these datasets, some models were trained on datasets scraped from GitHub by the authors themselves. In many cases, there was insufficient information to fully replicate the datasets as described, and these cases were excluded from this study.

Table 4.2: Division of LLMs based on training data

Only Natural Language	**Method-Level Code**	**Permissive File-Level Code**	**Non-permissive File-Level Code**
GPT-3	PLBART	CodeGen[1,2,9]	Codex[15]
T5	Tk-Instruct	InCoder[15]	CodeT5[1]
BART	ERNIE-Code	FLAN-T5[3]	BLOOM[1,7]
mT5	PyMT5	LLaMa[1]	Galactica[15]
CPM-2	LaMDA	CodeGen 2[2,3,9]	Baichuan 2[15]
PanGu-α	InstructGPT	StarCoder[3]	QWEN[15]
T0	CodeBERT	Gopher[10]	Skywork[8]
UL2	CodeRetriever	CodeT5Mix[11]	Pythia[2]
OPT	TraceBERT	CodeRL[11]	Jurassic-1[2]
NLLB	GraphCodeBERT	AlphaCode[15]	JuPyT5[15]
GLM	BERT Overflow	PaLM[6]	MT-NLG[2]
FLM	CoText	LLaMa 2[1]	PyCodeGPT[15]
GShard	PanGu-Coder	WizardLM[1]	U-PaLM[6]
HyperClova	CodeGPT	CodeT5+[11]	PanGu-Σ[15]
Yuan 1.0	CodeGPT-adapted	WizardCoder[3]	PaLM 2[15]
GLaM	CoditT5	SantaCoder[3]	Mistral[15]
AlexaTM	SPT-Code	PaLM-Coder[6,13]	GPT-C[15]
WeLM	FLAN	Vicuna[1]	PolyCoder[15]
BERT	UnixCoder	Stable Code[3]	GPT-3.5[15]
mBART	PanGu-Coder-FT	StableLM[2,3,4]	Code LLaMa[1,14]
GPT-1	PanGu-Coder 2	StableLM Zephyr[2,3,4]	GPT-NeoX[2]
XLNet	T5-Learning	Japanese StableLM[4]	CodeGeeX[2,5,15]
Sparrow		Stable Beluga[1]	CodeParrot[5]
PRCBERT			GPT-CC[12]
seBERT			Chinchilla[15]
ALBERT			GPT-J[2]
RoBERTa			FIM[15]
OPT-IML			GPT-Neo[2]
ERNIE 3.0			Falcon[2]
GPT-2			CuBERT[1]
WebGPT			

Table 4.3: File-level code datasets used for training LLMs

Ref	Dataset	Available	Count
1	Big Query	Pay-wall	10
2	The Pile	DMCA-takedown	12
3	The Stack	Open	8
4	RedPajama	Open	3
5	CodeParrot	Open	2
6	PaLM Dataset	Not Released	3
7	Roots	Not Open to All	1
8	SkyPile	Not Released	1
9	BigPython	Not Released	2
10	MassiveText	Not Released	1
11	GitHub-Code Dataset	Open	3
12	CodeClippy Dataset	Open	1
13	ExtraPythonData	Not Released	1
14	Code LLaMa Dataset	Not Released	1
15	Custom Dataset	Not Released	17

The datasets selected for further investigation are: *The Stack*[1], *The Pile*[2], *RedPajama*[3], *CodeParrot*[4], *Github Code*[5], and *CodeClippy*[6]. Among these datasets, The Stack, RedPajama, GitHub Code, and CodeParrot include a license field within their data structure. Additionally, permissive code is specifically mentioned for The Stack and RedPajama, while CodeParrot and GitHub Code feature both permissive and copyleft licenses in their code. For CodeClippy and The Pile, which lack a license field or explicit mention of permissive code, an analysis of their data reveals a mix of both permissive and copyleft licenses.

Finally, Table 4.4 presents an overview of the number of strong copyleft repositories extracted per language, covering a total of 32 programming languages. Our focus on these particular languages is based on their widespread use in the file-level code datasets we had access to. After the extraction process, we obtained approximately 35M code files.

4.1.2 RQ1.1 - Interest in Licensed Code

To assess the level of interest in the presence of licensed code in training data, we examine both its inclusion in training setups and how code licenses are referenced in research papers. The findings of this analysis are presented in Figures 4.1 and 4.2.

[1]The Stack
[2]The Pile
[3]RedPajama
[4]CodeParrot
[5]GitHub-Code
[6]CodeClippy

Table 4.4: Breakdown of strong copyleft repositories by programming language

Programming Languages	Repositories
C, C#, C++, Go, JavaScript, Java, Kotlin, Lua, Matlab, Perl, PHP, Python, R, Ruby, Rust, Shell, Swift, TypeScript	10000
Assembly, Dart, Haskell	5000–9999
DM, Elixir, Fortran, Julia, Lisp, OCaml, Pascal, Scala	1000–4999
Agda, Erlang, SQL	< 1000

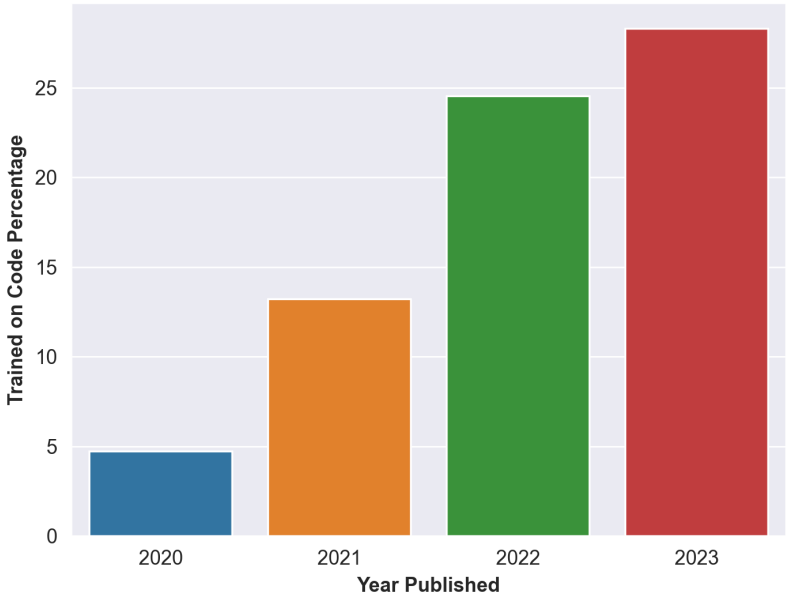


Figure 4.1: Annual percentage of LLMs trained on code

Since 2020, the use of code in training setups has gradually increased. We attribute this trend to the growing popularity of tools such as ChatGPT and Copilot, as well as the advantages of using code data when training models for natural language reasoning [43].

We further evaluate the focus on permissively licensed code in model training and observe a significant rise in datasets that exclusively contain such code. As shown in Figure 4.2, there is a noticeable spike in papers referencing permissively licensed datasets. We attribute this trend to the growing legal concerns surrounding generative models such as Stable Diffusion [88] and datasets such as The Pile [54]. Additionally, this surge in interest aligns with the release of The Stack, one of the datasets under investigation, which emphasizes the inclusion of only permissively licensed code.

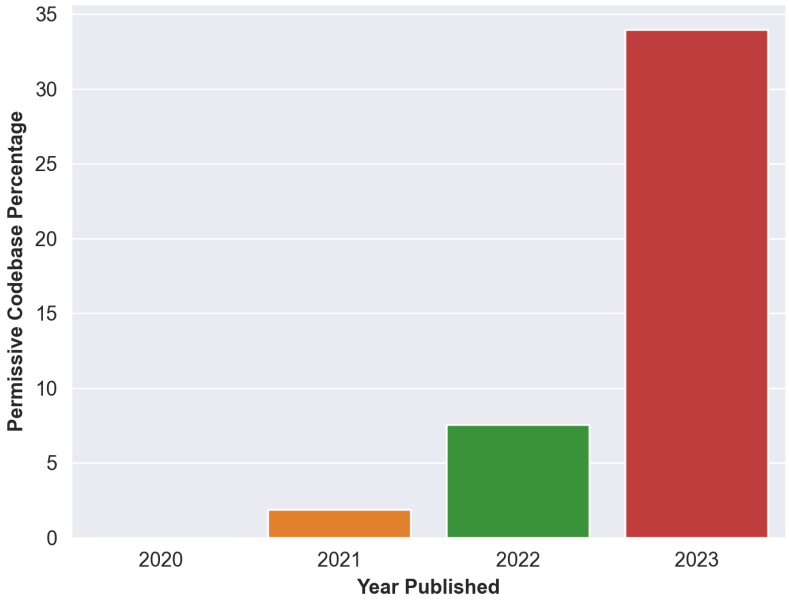


Figure 4.2: Annual percentage of LLMs trained on permissive file-level code

4.1.3 RQ1.2 - Strong Copyleft Inconsistencies

As mentioned earlier, we address this question by first compiling a dataset from repositories that exclusively use strong copyleft licenses, as shown in Table 4.4. Next, we identify exact duplicates of these files within publicly available datasets. Additionally, we extract the first comment from each file and search for substrings matching the standard license disclaimers commonly found in strong copyleft licenses.

In Table 4.5 we show the number of exact duplicates and occurrences of comments indicating strong copyleft licenses across all datasets. The dotted line distinguishes datasets that claim exclusive use of permissive licenses (listed at the top) from the rest. Our analysis reveals a significant overlap of exact duplicates between the dataset composed solely of strong copyleft licensed code and other datasets. Compared to other datasets such as The Pile, CodeParrot, and CodeClippy, which do not apply license filtering, The Stack and RedPajama exhibit a lower percentage of overlapping files. However, even among datasets that do license checks, overlap remains higher than in GitHub-Code, which did not check for licenses.

When examining comments that include the license of the file, we find that datasets that verify repository licenses show a significantly lower match percentage with strong copyleft licenses compared to those that do not.

Table 4.5: Number of code files identified with a strong copyleft license

Dataset	Files	Exact Duplicates		License Comments	
		Count	Percentage	Count	Percentage
The Stack	262,678,972	16,122,976	6.14%	2,067,830	0.78%
RedPajama	28,793,312	1,579,521	5.49%	15,544	0.05%
The Pile	18,044,000	4,113,263	22.80%	823,546	4.56%
CodeParrot	18,695,559	2,681,590	14.34%	2,844,150	15.21%
GitHub-Code	115,086,922	5,537,734	4.81%	7,548,615	6.56%
CodeClippy	71,140,482	7,993,768	11.24%	2,823,923	3.97%

4.1.4 RQ1.3 - Distribution Disclaimers

Lastly, we examine the occurrence of any language in the opening comments of all files. Our goal is to identify whether any comments indicate that the author of the file intended to restrict distribution. Table 4.6 summarizes the findings, showing the number of copyright disclaimers detected in the first comments together with the total count of first comments across datasets. Figure 4.3 provides an example to further highlight the nature of these comments.

When analyzing the proportion of first comments that include distribution disclaimers, we find that their occurance ranges from 5% to 7.5% across all datasets, except for RedPajama, which has only 1.3%. This indicates that many code files impose some restrictions on redistribution but do not rely on a standard disclaimer or an explicit license.

Table 4.6: Number of code files containing an ownership or copyright disclaimer

Dataset	Copyright		First Comments
	Count	Percentage	
The Stack	5,073,823	6.54%	77,595,559
RedPajama	30,500	1.34%	2,281,378
ThePile	501,877	7.39%	6,794,995
CodeParrot	773,062	5.38%	14,372,397
GitHub-Code	2,669,845	5.89%	45,301,797
CodeClippy	1,695,556	6.72%	25,223,157

```
1  <Company> all rights reserved.
2  this software contains proprietary and confidential
3  information of <Company> and its contributors.
4  use, disclosure and reproduction is prohibited without
5  prior consent.
```

Figure 4.3: Restrictions on sharing and distributing code found in a file from the RedPajama dataset

4.2 RQ2 - Non-Permissive Contamination in Open Datasets

This section begins with a brief overview of the structure of StackLessV2, followed by an analysis of strong copyleft code leakage in The Stack V2. Finally, we discuss how our dataset has been used in other research.

4.2.1 Dataset Structure

In Figure 4.4, we illustrate the features we scrape for each repository. This layout is consistent across all research questions. We include temporal indicators, such as the repository creation date, last push date, and retrieval date, to facilitate reproducibility and provide a basis for tracking the relevance of the data. Furthermore, we present the structure of StackLessV2 in Figure 4.5. Since we focus on near duplicates, instead of removing them as we do with exact-duplicates, we flag them, allowing for a more flexible evaluation approach. Each file in our dataset is linked to a list of file IDs, if applicable, that represent the near duplicates identified in The Stack V2.


```
{
   id: 126178683,
   full_name: "halo-dev/halo",
   html_url: "https://github.com/halo-dev/halo",
   stargazers_count: 29961,
   forks_count: 9142,
   watchers_count: 29961,
   open_issues_count: 330,
   language: "Java",
   created_at: "2018-03-21T12:56:52Z",
   pushed_at: "2023-12-24T14:40:08Z",
   license: {
     key: "gpl-3.0",
     name: "GNU General Public License v3.0",
     spdx_id: "GPL-3.0",
     url: "https://api.github.com/licenses/gpl-3.0",
     node_id: "MDc6TGljZW5zZTk="
   },
   retrieval_date: "12/24/2023, 6:57:14 PM (Europe/
       Amsterdam)"
}
```


Figure 4.4: Example of features extracted for each repository

4.2.2 Contamination Assessment

A summary of the number of files removed from StackLessV2 after each pre-processing step, as well as the final dataset size, can be seen in Table 4.7. Approximately 5.63M files out of the total 7.8M are exact duplicates. Of these exact duplicates, around 4.73M were found within our dataset, while 900,000 originated from the Java subset of The Stack V2. On top of that, we

```
{
   file_name: "HeadTest.java",
   file_path: "jbossas_httpserver/src/test/java/HeadTest.
       java",
   content: "public class HeadTest { @org.junit.Test public
        void testThis()..."
   file_size: 4,668,
   language: "Java",
   extension: ".java",
   repo_name: "jbossas/httpserver",
   repo_stars: 41,
   repo_forks: 38,
   repo_open_issues: 1,
   repo_created_at: 2011-05-09T19:29:49Z,
   repo_pushed_at: 2020-10-12T18:36:29Z,
   near_dups_stkv2_idx: [125903722, 167385005]
}
```


Figure 4.5: Example of one entry structure from StackLessV2

identified another approximately 800,000 near duplicates coming from the Java subset of The Stack V2. We observe a significant amount of near duplicates, particularly of strong copyleft code, suggesting that the contamination is more pervasive and subtle than anticipated, which can impact the reliability of evaluations. Moreover, the presence of strong copyleft code could extend to other programming languages in The Stack V2, potentially affecting the integrity of the dataset and leading to licensing issues. Finally, as previously mentioned, we flag these near duplicates instead of removing them, resulting in a final dataset size of 2.13M files.

Step	#Files
Raw data	7.80 M
Auto-generated	0.04 M
Exact deduplication	5.63 M
Near deduplication	0.80 M
Final	1.33 M

Table 4.7: Files removed at each step and final StackLessV2 size after filtering out near duplicates

4.2.3 Uncovering Attention Patterns at Scale in LLMs

We illustrate the application of our dataset in Katzy et al.'s study, which is currently under review. While LLMs continue to evolve at a rapid pace, the development of methods for effectively interpreting and understanding their outputs has not kept up with this growth. The authors propose a novel approach to analyze LLM behavior at model-level, enabling scalable comparisons

across different inferences and scenarios. They argue that current methods, which focus on individual components, are complex and hard to scale. To provide high-level explanations, they track attention patterns across predictions and introduce AP-MAE, a vision transformer-based masked auto-encoder trained to encode the attention patterns generated in LLMs.

The StarCoder2 models with 3B, 7B, and 15B parameters are selected as target models due to their accessibility and transparent training process. They perform fill-in-the-middle (FiM) inference at random locations with these models to generate attention patterns, which are used to train an AP-MAE model for each. Then, they focus on code related to specific tasks, such as closing brackets, identifiers, and boolean operators, running 11,000 FiM inferences. The resulting attention patterns are encoded and clustered to analyze their distribution across heads and to assess whether LLMs produce consistent patterns when prompted with actual code versus noise, and if these patterns vary between correct and incorrect predictions. The attention patterns for training each AP-MAE model are generated by prompting the StarCoder2 models with StackLessV2 data, because it is already deduplicated from their training set. Near duplicates are filtered out to minimize interference from behaviors such as memorization, which could affect the accuracy of the results. Additionally, they chose a Java dataset for its structured nature, enabling the extraction of specific tasks from real-world data to identify consistent LLM behavior.

Their results show that vision transformers trained in a masked auto-encoder setting can encode attention patterns and generalize to unseen attention heads. They observed various patterns across clusters, such as diagonal, vertical, and square. Additionally, models used at most 8.6% of possible pattern locations, indicating behavior is localized by task, with smaller models reusing more pattern locations than larger ones. Finally, they found that models generate distinct patterns when making incorrect predictions.

4.3 RQ3 - Dataset for Reliable and Reproducible LLM Evaluations

In this section, we present the composition of The Heap and discuss its layout, highlighting features designed to improve usability and facilitate integration into other research.

4.3.1 Dataset Composition

The Heap is designed to support diverse research needs, so we select programming languages based on specific criteria to ensure broad coverage. Our collection includes languages with varied syntactic structures, such as Assembly, C, Haskell, Lisp and Python. We also cover multiple programming paradigms: procedural languages such as Cobol, C, and Pascal; object-oriented languages such as C#, Java, and Python; and functional languages like Clojure and Erlang. To address specialized needs, we include domain-specific languages such as Coq, Emacs-Lisp, and Mathematica. The complete list of 57 languages is shown in Table 4.8. The third column shows the number of files obtained after filtering based on file size and word count, while the last column indicates the number of files remaining after we removed exact duplicates within our dataset. Additionally, we present an overview of the licenses used for The Heap in Table 4.9. We select a broader range of copyleft licenses to collect more unseen data, supporting accurate model evaluations on downstream tasks.

Table 4.8: Languages included in The Heap

Language	Repositories	Raw Files	Unique Files
Ada	676	41,367	34,068
Agda	142	5,483	3,021
ANTLR	101	564	538
Apex	253	17,833	7,561
Assembly	7,100	208,896	101,093
C	50,000	16,585,280	3,076,470
C#	50,000	5,906,716	3,257,456
C++	50,000	14,891,856	4,469,823
Clojure	27,107	380,567	269,118
Cobol	341	2,242	1,172
Common Lisp	796	45,083	13,922
Coq	477	54,137	22,549
Crystal	368	11,606	6,818
Cuda	1,191	26,948	12,418
D	1,185	185,630	54,034
Dart	11,907	484,935	412,675
EJS	1,475	15,513	12,832
Elixir	2,371	643,856	102,874
Emacs Lisp	377	8,260	7,312
Erlang	1,240	55,932	27,322
F#	876	22,152	13,282
Forth	222	28,287	5,129
Go	50,000	7,506,379	2,328,529
Groovy	2,198	60,299	47,366
Hack	1,379	84,916	37,189
Haskell	8,023	122,788	106,583
Java	50,000	9,989,601	5,168,193
JavaScript	50,000	8,289,901	1,907,803
Julia	2,859	46,284	36,830
Kotlin	21,665	1,467,343	1,042,136
Less	433	17,276	7,308
Lua	42,241	4,605,230	905,120
Mathematica	1,528	164,498	21,208
MATLAB	20,828	1,051,354	599,085
NetLogo	332	900	855
NewLisp	35	5,819	5,123
Nix	1,892	75,093	70,407
Objective-C	7,700	1,899,714	520,332
OCaml	1,961	121,890	60,863
Pascal	5,218	330,832	180,652
Perl	14,673	1,798,520	224,753
PHP	50,000	12,707,727	3,310,243
Processing	2,950	24,723	20,304
Prolog	1,071	38,995	17,570
Python	50,000	2,290,182	1,595,919
R	44,993	589,139	11,679
Raku	158	1,384	689
Ruby	13,378	1,579,655	662,915
Rust	42,847	2,496,177	802,707
Scala	5,893	749,370	210,630
Scheme	1,878	106,620	50,222
Scilab	199	4,531	3,896
SQL	130	47,185	40,800
Starlark	146	524	487
Swift	13,924	633,819	434,849
Vue	14,858	457,605	321,502
WebAssembly	68	834	544
Total	733,663	98,945,943	32,666,778

Table 4.9: Copyleft licenses included in The Heap

License	Family
CECILL-1.0, CECILL-1.1, CECILL-2.0, CECILL-2.1, CECILL-C, EPL-1.0, EPL-2.0, LGPL-2.1, LGPL-3.0, MS-RL, MPL-2.0	Weak Copyleft
GPL-2.0, GPL-3.0	Strong Copyleft
AGPL-3.0, EUPL-1.1, EUPL-1.2, OSL-3.0	Network Copyleft

The datasets selected for deduplication are drawn from the list we curated during the license analysis, with the addition of The Stack V2, as shown in Table 4.10. Since comment removal depends on the programming language, we cannot accurately determine the correct language for The Pile and CodeClippy, as detailed in Section 2.3. While we could predict the languages in these datasets, the tools available for this task often return incorrect predictions, which could lead to undetected duplicates. To ensure our dataset remains free from contamination, we exclude these two datasets from consideration.

Table 4.10: List of publicly-available datasets used for deduplication against The Heap

Dataset	Source
The Stack V2 [23]	All files with permissive licenses, as well as unlicensed ones, gathered from the Software Heritage [90] archive.
The Stack [22]	All permissively licensed repositories collected in the GHArchive [91] and scraped from GitHub.
Red Pajama [24]	Repositories licensed under MIT, BSD, or Apache from the GitHub dataset hosted by Google BigQuery [92].
GitHub-Code [34]	Repositories from the GitHub dataset hosted by Google BigQuery [92].
CodeParrot [30]	All Python files from the GitHub dataset hosted by Google BigQuery [92].

4.3.2 Dataset Layout

The Heap is divided into multiple subsets, with each representing a specific programming language. Within each subset, the dataset entries fall into three categories: file content and metadata, quality indicators, and duplicates. This layout promotes ease of use by organizing the data in a structured and accessible manner. An example of the structure can be seen in Figure 4.6.

File Content and Metadata. This category includes the actual content of the file, together with details about the filename and path, as these have been part of the pre-training process for some LLMs [8, 23, 36].

```
{
    id: 200,
    file_name: "kernel.lisp",
    file_path: "whily_yalo/cc/kernel.lisp",
    content: "REPL: loop (jmp short read-start) ;; ...",
    size: 4,099,
    language: "Common Lisp",
    extension: ".lisp",
    total_lines: 125,
    avg_line_length: 27.52,
    max_line_length: 104,
    alphanum_fraction: 0.59,
    repo_name: "whily/yalo",
    repo_stars: 571,
    repo_forks: 32,
    repo_open_issues:1,
    repo_license: "GPL-2.0",
    repo_extraction_date: "9/19/2024, 11:24:32 AM",
    exact_duplicates_stackv1: False,
    exact_duplicates_stackv2: True,
    near_duplicates_stackv1: False,
    near_duplicates_stackv2: True,
    ...
  }
```

Figure 4.6: Example of one entry structure from The Heap

Quality Indicators. To support the selection of files for downstream tasks, we incorporated several quality indicators previously used in related works [22, 23], making it easier to filter and choose relevant files. These indicators include file-specific metrics such as the *number of total lines*, *average line length*, *maximum line length*, and *alphanumeric fraction*, as well as repository-wide statistics including *repository stars*, *repository forks*, *open issues*, and the *repository extraction date*.

Duplicates. Since we deduplicate The Heap against several publicly available datasets, we add two columns for each dataset. One column indicates whether an exact duplicate of the file exists, while the other shows if a near duplicate is present. Instead of removing files, we use a Boolean mask to preserve as much data as possible from each dataset. This approach helps us retain a larger portion of the available data. Moreover, this reduces both time and computational overhead by eliminating the need for researchers to handle duplicate detection, enabling reliable and reproducible evaluations. They can easily filter the data by language or by files that are either exact or near duplicates, allowing them to adjust the dataset according to their needs.

4.3.3 Data Extraction Attacks Before and After Fine-Tuning

The Heap was also used in a study by Salerno et al., which investigates the extent to which LLMs4Code leak data from their pre-training and fine-tuning corpora through data extraction

attacks. The authors state that fine-tuning LLMs on downstream tasks is becoming more dominant than pre-training the models, as it requires fewer resources. Although fine-tuning requires smaller amounts of data, this can still raise concerns about memorizing proprietary or sensitive information, which can be extracted from the fine-tuned models.

The authors create a custom benchmark to assess the vulnerability of both pre-training and fine-tuning samples to extraction attacks. They also introduce a framework based on a data extraction security game, where extractability serves as a proxy for measuring memorization. Experiments are conducted on the StarCoder2 models with 3B, 7B, and 15B parameters, due to their recent release, strong performance, and open availability. They fine-tune these models for code completion, and compare the extractability of pre-training data before and after fine-tuning, as well as the extractability of fine-tuning data. Finally, they examine whether certain data categories become more or less extractable post-fine-tuning. For fine-tuning, the authors use a Java subset from The Heap, filtering out duplicates to ensure no overlap with the original training data of the model, which could affect the results. They also choose a Java dataset as prior studies have focused primarily on Python.

Their results indicate that fine-tuning decreases the extractability of pre-training data. However, fine-tuning smaller models increases their susceptibility to data extraction attacks on fine-tuning data. They showed that approximately 55% of extractable pre-training data could be retrieved from StarCoder2-15B, but this number decreased to 23.5% after fine-tuning. Smaller models were more vulnerable to fine-tuning data extraction, with around 63% of samples extracted from StarCoder2-3B. Lastly, most memorized data belonged to categories such as data carriers and license information.

5 | Discussion

Our findings underscore the difficulties in maintaining clean, legally compliant data that supports reliable and reproducible LLM research. We have seen a significant shift towards permissively licensed datasets, reflected in the rising number of papers referencing such sources. This trend mirrors the growing legal interest in generative models, particularly those trained using large amounts of web-scraped data. Despite the efforts of creating permissive datasets that promote openness, we identified a substantial overlap between these datasets and non-permissive data, which leads to license infringements. Even datasets that verify licenses can contain code with conflicting legal obligations, suggesting that automated filtering alone may be insufficient. We have observed a similar case with The Stack V2, where despite its recent release, we found that it contains non-permissive code. This can be attributed to the way licensing detection is performed in The Stack V2. The authors rely on external tools for detecting the file-level license type, as GHArchive does not provide repo-level license information for 96.93% of the repositories [23]. This raises concerns about the accuracy of the detection tools employed, as some licenses may be missed or misclassified as 'no license', particularly those in non-standard formats. While GitHub repositories can also be unlicensed, this implies that default copyright laws apply, meaning explicit permission is required to modify or distribute their code.

Furthermore, we have seen that legal concerns can arise from more subtle forms of restriction, such as disclaimers at the file-level. Such instances further complicate the detection of licenses, and emphasize the need for a more fine-grained approach to identifying and managing licensed code. In addition, we found an overlap between open datasets and non-permissive code at a deeper level, in the form of near duplicates. These duplicates create a legal and ethical gray area, as they blur the line between original and derivative works, making it difficult to determine whether licensing terms are being violated or circumvented.

Our analysis reveals that relying only on the legal and ethical constraints of specific license types, particularly at the repository level, is insufficient to prevent contamination from public datasets, even if they claim to not include those licenses. This highlights both the shortcomings in the licensing detection methods employed, and the tendency of developers to copy code across projects without considering the associated licenses. To minimize any contamination risk, we incorporate both exact and near deduplication in the creation process of our datasets, alongside the selection of copyleft code, providing an additional layer of protection. These design decisions facilitate the creation of datasets with fresh data, contributing to more reliable LLM evaluations. Moreover, using a diverse set of programming languages and decontaminating data against multiple public pre-training datasets improves flexibility and facilitates study reproducibility. The Heap serves as a stable reference point across studies, minimizing variability in configurations.

As shown, researchers can seamlessly integrate it into their work without the redundancy of repeatedly scraping or deduplicating new data, processes that can introduce inconsistencies due to implementation mismatches or data drift. Its structured format also facilitates usability, enabling researchers to filter data based on specific features, according to their needs.

5.0.1 Implications

The real-world implications of our study impact various stakeholders. For dataset maintainers and curators, we have highlighted that code comments contain more information about code distribution than just the license. We also found significant overlap in repositories with different licenses, making it difficult to track the origin of the code and determine the correct license to apply. This also impacts practitioners who train LLMs, as they face the risk of their models being targeted if trained on licensed code. Lastly, end users of LLMs must be cautious about unintentionally incorporating licensed code into their code bases, as doing so with strong copyleft code could potentially require them to open-source their own code. The key question these implications raise for the field of LLMs4Code is: *who is responsible for the training data and the output of the final models?*

5.0.2 Recommendations

To address the issues identified during our investigation, we propose several recommendations for developers involved in training LLMs. First, our findings show that no LLM has been trained on a dataset entirely free from licensing issues. Since fine-tuning a model retains the information embedded in its weights, it is essential to be aware of this when selecting a model for the fine-tuning process.

Our investigation also uncovered that even when providers of permissively licensed source code datasets have good intentions, straightforward searches often lead to multiple inconsistencies. Therefore, we recommend conducting a comprehensive analysis of the datasets to detect and resolve any licensing issues before fine-tuning or training a model from scratch. This step is crucial, since removing information from the model's weights has not been proven to be a reliable solution [94].

Finally, we suggest adapting and scaling up current license detection methods to handle large code datasets, helping to minimize license inconsistencies in datasets scraped from online repositories.

5.0.3 Limitations

This subsection outlines the key limitations of this study. In the context of the licensing analysis, the primary limitations stem from two factors. First, the dynamic nature of code means that copied snippets can evolve over time. Second, many entities lack transparency about their training methodologies.

Regarding the dataset limitations, there are two additional concerns. First, other parties may choose to train their models on this data, thereby compromising its intended benefits. Second,

developers may object to their code being included in this dataset. These limitations affect both StackLessV2 and The Heap, however, we will only refer to the latter for simplicity.

Data Drift. As code bases are constantly evolving, we established a fixed point in time for collecting code. Nevertheless, obtaining exact copies of our strong copyleft-licensed dataset proves to be challenging. Since the creation of our dataset, some repositories may have been deleted or switched to private, while new repositories have been added. As a result, our datasets may slightly differ in composition. Nevertheless, it is important to emphasize any discrepancies in dataset composition do not affect the conclusions of the licensing analysis.

Non-Reproducible Papers. A key limitation we encountered while gathering information on the training of proprietary models and analyzing their datasets was the lack of transparency from some companies regarding their training processes. In many cases, essential details for replicating the dataset were unavailable, such as the scraping date, the specific repositories involved, or any filtering criteria applied to the code files. Additionally, when companies discussed their models in blog posts or papers, the descriptions were often vague, making it difficult to determine the precise data used for training. Unfortunately, given the competitive nature of LLMs and the costs associated with data curation and model training, this trend is likely to persist.

Model Training. To ensure a fair evaluation of an LLM using The Heap, researchers must confirm that the target model has not been trained on The Heap. Our deduplication process guarantees this for current LLMs, while our data collection method adds an additional layer of protection against the inclusion of The Heap in training datasets. As the trend in LLM training has moved towards using only permissively licensed data, this would exclude The Heap from this scope. Additionally, restricting The Heap to research purposes helps mitigate issues with author attribution in LLM outputs, as trained models are not intended for end-user use [95]. Moreover, the development of membership inference attacks, now extended to entire datasets [96], will soon allow for retroactive testing to identify whether The Heap was included in the training data of a model.

Ethics. As public repositories have increasingly been used to train LLMs4Code, many authors of older repositories were unaware that their code could be included for such purposes, leaving them without the option to opt out. Furthermore, there is currently no standard method for developers to choose whether their code is included in datasets. We recognize these ethical concerns surrounding the use of code for AI purposes, and provide repository owners with the option to opt out of having their code included in our dataset. While this solution is not perfect, as it places the responsibility on the authors, it is in line with current best practices [23].

6 | Conclusion

In this study, we analyzed the extent of license infringements in LLM training data. Then, we investigated the duplication depth of non-permissive code within The Stack V2 by creating a new code dataset, which serves as a valuable resource for reliable LLM evaluations. Finally, extending on this, we developed another multilingual code dataset that not only addresses contamination concerns but also supports the reproducibility of LLM evaluations.

We collected data from 6 literature surveys, extracting 106 foundation models, of which 53 were trained on file-level code, and 23 specifically mentioned using only permissively licensed code. We extracted 30 datasets that worked with file-level code, out of which 16 were custom GitHub scrapes. Additionally, we could access only 6 datasets that were publicly available without restrictions. Of these, 4 did not filter code based on licenses, while 2 did. We collected a total of 514 million code files, across all 6 datasets, to evaluate the presence of strong copyleft licenses.

To assess the presence of strong copyleft-licensed code, we scraped GitHub for repositories released under GPL or AGPL licenses, covering 32 programming languages, and resulting in a dataset of 35M code files. We computed the SHA-256 hashes for all collected datasets, including the strong copyleft dataset we created. Using these hashes, we analyzed the overlap of exact duplicates. All datasets showed significant overlap with the strong copyleft dataset. While those that focused on permissively licensed code exhibited less overlap, they still shared at least 5% of the strong copyleft dataset. We also extracted comments from the datasets to identify file-level licenses. Datasets that claimed to include only permissively licensed code performed better, with 0.05% and 0.8% of files having a license comment. Nonetheless, this still results in over 2M files with a strong copyleft license in the case of The Stack.

In addition, we found a higher prevalence of distribution-related comments within the datasets, in comparison to license-related comments. To support future research on detecting and removing licensed code from public datasets, we are releasing a dataset containing 171 M code comments with all personally identifiable information removed.

Furthermore, to study the depth of non-permissive code in The Stack V2, we developed StackLessV2, a companion strong copyleft Java dataset designed to facilitate valid LLM evaluations on unseen data. Our findings align with the licensing analysis results, revealing an overlap between our strong copyleft dataset and the Java subset of The Stack V2. Among the 7.8M scraped files, we identified approximately $900,000$ exact duplicates from The Stack V2, along with an additional $800,000$ near duplicates.

Finally, we introduced The Heap, a multilingual copyleft code dataset that has been deduplicated against commonly used LLM pre-training datasets. Covering 57 programming languages

and a wide range of copyleft licenses, The Heap was designed to facilitate the extraction of unseen data while minimizing contamination. It enables researchers to study the behavior and performance of code LLMs without the need for extensive deduplication across other datasets. This addresses the limitation of 90% of LLM studies not testing for data leakage [18], resulting in more reliable conclusions and supporting reproducible evaluations. Lastly, its structured format facilitates usability, making model evaluations easier, more flexible, and accessible to researchers, as illustrated by the applications provided.

6.0.1 Future Work

Based on the results gathered, we have identified several opportunities for future work. Future efforts could focus on extracting licenses and intent from file comments, as well as determining the exact licenses associated with code. In terms of dataset development, there are various improvements to explore, such as refining the deduplication process, releasing new training datasets, providing more detailed information about the natural languages included, and incorporating topic modeling.

Code Comment Intent. We have observed that, in addition to the legal concerns regarding potential license violations, there is also the issue of whether authors have consented to having their code included in training data. While some curators offer an opt-out system to remove code from the dataset [22], it should be possible to infer an author's intent from the comments, if available.

License Infringement Detection. A key limitation of detecting only duplicates is that we cannot fully determine whether a license violation has occurred when code is included in a non-permissive dataset versus a permissive one, as we can only identify duplication. An intriguing area for future research could involve analyzing code repository networks, such as GitHub, to automatically detect whether a file has been copied in a manner that violates licensing terms. This could involve a simple analysis of license changes in forks or a more detailed investigation tracking code modifications and version histories across different repositories over time. It should be noted that this problem is not unique to LLM datasets [97, 98].

New Datasets. The primary objective of The Heap is to reduce the effort of deduplicating datasets used for downstream tasks in future research. This goal can only be achieved if the dataset is deduplicated against all available datasets. As new datasets are released, we plan on putting them through the same deduplication process.

Deduplication. We tackled dataset deduplication using two common techniques, exact deduplication through hashing and near deduplication using locality-sensitive hashing. However, there is a lack of comprehensive research on what defines an optimal deduplication approach. Potential problems could arise from duplicates at a finer granularity than file-based deduplication. Once research on the effects of different deduplication methods is conducted, we plan to incorporate these insights as an additional feature in the dataset.

Topic Modeling. Programming languages can help define the focus of an analysis, such as using Mathematica for mathematics or JavaScript for web development, but many languages are versatile and can be applied across various domains. Adopting a topic modeling approach similar to FineWeb for code datasets would offer useful annotations for the code files and highlight any imbalances based on different topics within the dataset.

Natural Language. One area of research that has not been explored much involves the presence of multiple natural languages within code. Since natural languages are often mixed within a single file [99], we aim to adopt a Parts of Speech-like tagging [100] system to identify the different languages present in each file. This approach will provide insights into how code models perform when the code is not in English. It will benefit the development of non-English code LLMs, and also help evaluate English-based models by focusing only on English content.

Bibliography

[1] Jonathan Katzy, Razvan Mihai Popescu, Arie van Deursen, and Maliheh Izadi. The heap: A contamination-free multilingual code dataset for evaluating large language models, 2025. URL https://arxiv.org/abs/2501.09653.

[2] Jonathan Katzy, Razvan Popescu, Arie Van Deursen, and Maliheh Izadi. An exploratory investigation into code license infringements in large language model training datasets. In *Proceedings of the 2024 IEEE/ACM First International Conference on AI Foundation Models and Software Engineering*, FORGE '24, page 74–85, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706097. doi: 10.1145/3650105.3652298. URL https://doi.org/10.1145/3650105.3652298.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[4] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeff Wu, and Dario Amodei. Scaling laws for neural language models. *ArXiv*, abs/2001.08361, 2020. URL https://api.semanticscholar.org/CorpusID:210861095.

[5] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. *ArXiv*, abs/1902.00751, 2019. URL https://api.semanticscholar.org/CorpusID:59599816.

[6] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc

Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. Starcoder 2 and the stack v2: The next generation, 2024. URL https://arxiv.org/abs/2402.19173.

[7] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-coder technical report, 2024. URL https://arxiv.org/abs/2409.12186.

[8] CodeGemma Team. Codegemma: Open code models based on gemma, 06 2024.

[9] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé, Jared Kaplan, Harrison Edwards, Yura 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, David W. Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William H. Guss, Alex Nichol, Igor Babuschkin, Suchir Balaji, Shantanu Jain, Andrew Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew M. 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*, abs/2107.03374, 2021. URL https://api.semanticscholar.org/CorpusID:235755472.

[10] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

[11] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation, 2024. URL https://arxiv.org/abs/2406.00515.

[12] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL https://openreview.net/forum?id=yzkSU5zdwD. Survey Certification.

[13] Wayne Zhao, Kun Zhou, Li Junyi, Tang Tianyi, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, and Ji-Rong Wen. A survey of large language models, 03 2023.

[14] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. The impact of ai on developer productivity: Evidence from github copilot, 02 2023.

[15] Burak Yetiştiren, Işık Özsoy, Miray Ayerdem, and Eray Tüzün. Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt, 04 2023.

[16] OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Madry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, Alex Paino, Alex Renzin, Alex Tachard Passos, Alexander Kirillov, Alexi Christakis, Alexis Conneau, Ali Kamali, Allan Jabri, Allison Moyer, Allison Tam, Amadou Crookes, Amin Tootoochian, Amin Tootoonchian, Ananya Kumar, Andrea Vallone, Andrej Karpathy, Andrew Braunstein, Andrew Cann, Andrew Codispoti, Andrew Galu, Andrew Kondrich, Andrew Tulloch, Andrey Mishchenko, Angela Baek, Angela Jiang, Antoine Pelisse, Antonia Woodford, Anuj Gosalia, Arka Dhar, Ashley Pantuliano, Avi Nayak, Avital Oliver, Barret Zoph, Behrooz Ghorbani, Ben Leimberger, Ben Rossen, Ben Sokolowsky, Ben Wang, Benjamin Zweig, Beth Hoover, Blake Samic, Bob McGrew, Bobby Spero, Bogo Giertler, Bowen Cheng, Brad Lightcap, Brandon Walkin, Brendan Quinn, Brian Guarraci, Brian Hsu, Bright Kellogg, Brydon Eastman, Camillo Lugaresi, Carroll Wainwright, Cary Bassin, Cary Hudson, Casey Chu, Chad Nelson, Chak Li, Chan Jun Shern, Channing Conger, Charlotte Barette, Chelsea Voss, Chen Ding, Cheng Lu, Chong Zhang, Chris Beaumont, Chris Hallacy, Chris Koch, Christian Gibson, Christina Kim, Christine Choi, Christine McLeavey, Christopher Hesse, Claudia Fischer, Clemens Winter, Coley Czarnecki, Colin Jarvis, Colin Wei, Constantin Koumouzelis, Dane Sherburn, Daniel Kappler, Daniel Levin, Daniel Levy, David Carr, David Farhi, David Mely, David Robinson, David Sasaki, Denny Jin, Dev Valladares, Dimitris Tsipras, Doug Li, Duc Phong Nguyen, Duncan Findlay, Edede Oiwoh, Edmund Wong, Ehsan Asdar, Elizabeth Proehl, Elizabeth Yang, Eric Antonow, Eric Kramer, Eric Peterson, Eric Sigler, Eric Wallace, Eugene Brevdo, Evan Mays, Farzad Khorasani, Felipe Petroski Such, Filippo Raso, Francis Zhang, Fred von Lohmann, Freddie Sulit, Gabriel Goh, Gene Oden, Geoff Salmon, Giulio Starace, Greg Brockman, Hadi Salman, Haiming Bao, Haitang Hu, Hannah Wong, Haoyu Wang, Heather Schmidt, Heather Whitney, Heewoo Jun, Hendrik Kirchner, Henrique Ponde de Oliveira Pinto, Hongyu Ren, Huiwen Chang, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian O'Connell, Ian Osband, Ian Silber, Ian Sohl, Ibrahim Okuyucu, Ikai Lan, Ilya Kostrikov, Ilya Sutskever, Ingmar Kanitscheider, Ishaan Gulrajani, Jacob Coxon, Jacob Menick, Jakub Pachocki, James Aung, James Betker, James Crooks, James Lennon, Jamie Kiros, Jan Leike, Jane Park, Jason Kwon, Jason Phang, Jason Teplitz, Jason Wei, Jason Wolfe,

Jay Chen, Jeff Harris, Jenia Varavva, Jessica Gan Lee, Jessica Shieh, Ji Lin, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joanne Jang, Joaquin Quinonero Candela, Joe Beutler, Joe Landers, Joel Parish, Johannes Heidecke, John Schulman, Jonathan Lachman, Jonathan McKay, Jonathan Uesato, Jonathan Ward, Jong Wook Kim, Joost Huizinga, Jordan Sitkin, Jos Kraaijeveld, Josh Gross, Josh Kaplan, Josh Snyder, Joshua Achiam, Joy Jiao, Joyce Lee, Juntang Zhuang, Justyn Harriman, Kai Fricke, Kai Hayashi, Karan Singhal, Katy Shi, Kavin Karthik, Kayla Wood, Kendra Rimbach, Kenny Hsu, Kenny Nguyen, Keren Gu-Lemberg, Kevin Button, Kevin Liu, Kiel Howe, Krithika Muthukumar, Kyle Luther, Lama Ahmad, Larry Kai, Lauren Itow, Lauren Workman, Leher Pathak, Leo Chen, Li Jing, Lia Guy, Liam Fedus, Liang Zhou, Lien Mamitsuka, Lilian Weng, Lindsay McCallum, Lindsey Held, Long Ouyang, Louis Feuvrier, Lu Zhang, Lukas Kondraciuk, Lukasz Kaiser, Luke Hewitt, Luke Metz, Lyric Doshi, Mada Aflak, Maddie Simens, Madelaine Boyd, Madeleine Thompson, Marat Dukhan, Mark Chen, Mark Gray, Mark Hudnall, Marvin Zhang, Marwan Aljubeh, Mateusz Litwin, Matthew Zeng, Max Johnson, Maya Shetty, Mayank Gupta, Meghan Shah, Mehmet Yatbaz, Meng Jia Yang, Mengchao Zhong, Mia Glaese, Mianna Chen, Michael Janner, Michael Lampe, Michael Petrov, Michael Wu, Michele Wang, Michelle Fradin, Michelle Pokrass, Miguel Castro, Miguel Oom Temudo de Castro, Mikhail Pavlov, Miles Brundage, Miles Wang, Minal Khan, Mira Murati, Mo Bavarian, Molly Lin, Murat Yesildal, Nacho Soto, Natalia Gimelshein, Natalie Cone, Natalie Staudacher, Natalie Summers, Natan LaFontaine, Neil Chowdhury, Nick Ryder, Nick Stathas, Nick Turley, Nik Tezak, Niko Felix, Nithanth Kudige, Nitish Keskar, Noah Deutsch, Noel Bundick, Nora Puckett, Ofir Nachum, Ola Okelola, Oleg Boiko, Oleg Murk, Oliver Jaffe, Olivia Watkins, Olivier Godement, Owen Campbell-Moore, Patrick Chao, Paul McMillan, Pavel Belov, Peng Su, Peter Bak, Peter Bakkum, Peter Deng, Peter Dolan, Peter Hoeschele, Peter Welinder, Phil Tillet, Philip Pronin, Philippe Tillet, Prafulla Dhariwal, Qiming Yuan, Rachel Dias, Rachel Lim, Rahul Arora, Rajan Troll, Randall Lin, Rapha Gontijo Lopes, Raul Puri, Reah Miyara, Reimar Leike, Renaud Gaubert, Reza Zamani, Ricky Wang, Rob Donnelly, Rob Honsby, Rocky Smith, Rohan Sahai, Rohit Ramchandani, Romain Huet, Rory Carmichael, Rowan Zellers, Roy Chen, Ruby Chen, Ruslan Nigmatullin, Ryan Cheu, Saachi Jain, Sam Altman, Sam Schoenholz, Sam Toizer, Samuel Miserendino, Sandhini Agarwal, Sara Culver, Scott Ethersmith, Scott Gray, Sean Grove, Sean Metzger, Shamez Hermani, Shantanu Jain, Shengjia Zhao, Sherwin Wu, Shino Jomoto, Shirong Wu, Shuaiqi, Xia, Sonia Phene, Spencer Papay, Srinivas Narayanan, Steve Coffey, Steve Lee, Stewart Hall, Suchir Balaji, Tal Broda, Tal Stramer, Tao Xu, Tarun Gogineni, Taya Christianson, Ted Sanders, Tejal Patwardhan, Thomas Cunninghman, Thomas Degry, Thomas Dimson, Thomas Raoux, Thomas Shadwell, Tianhao Zheng, Todd Underwood, Todor Markov, Toki Sherbakov, Tom Rubin, Tom Stasi, Tomer Kaftan, Tristan Heywood, Troy Peterson, Tyce Walters, Tyna Eloundou, Valerie Qi, Veit Moeller, Vinnie Monaco, Vishal Kuo, Vlad Fomenko, Wayne Chang, Weiyi Zheng, Wenda Zhou, Wesam Manassra, Will Sheu, Wojciech Zaremba, Yash Patil, Yilei Qian, Yongjik Kim, Youlong Cheng, Yu Zhang, Yuchen He, Yuchen Zhang, Yujia Jin, Yunxing Dai, and Yury Malkov. Gpt-4o system card, 2024. URL https://arxiv.org/abs/2410.21276.

[17] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

[18] Antonio Vitale, Rocco Oliveto, and Simone Scalabrino. A catalog of data smells for coding tasks. *ACM Trans. Softw. Eng. Methodol.*, December 2024. ISSN 1049-331X. doi: 10.1145/3707457. URL https://doi.org/10.1145/3707457. Just Accepted.

[19] Getty images (us), inc. v. stability ai, inc. United States District Court for the District of Delaware, February 3 2023. Case No. 1:23-cv-00135-UNA.

[20] Mike huckabee, relevate group, david kinnaman, tsh oxenreider, lysa terkeurst, and john blase, plaintiffs, v. meta platforms, inc., bloomberg l.p., bloomberg finance, l.p., microsoft corporation, and the eleutherai institute, defendants. United States District Court Southern District of New York, October 17 2023. Case No. 1:23-cv-09152-LGS.

[21] The new york times company v. microsoft corporation, openai, inc., openai lp, openai gp, llc, openai llc, openai opco llc, openai global llc, oai corporation, llc, and openai holdings, llc. United States District Court Southern District of New York, December 27 2023. Case No. 1:23-cv-11195.

[22] 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.

[23] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. Starcoder 2 and the stack v2: The next generation, 2024. URL https://arxiv.org/abs/2402.19173.

[24] Together Computer. Redpajama: An open source recipe to reproduce llama training dataset, 2023. URL https://github.com/togethercomputer/RedPajama-Data.

[25] Mohammad Gharehyazie, Baishakhi Ray, Mehdi Keshani, Masoumeh Soleimani Zavosht, Abbas Heydarnoori, and Vladimir Filkov. Cross-project code clones in github. *Empirical Softw. Engg.*, 24(3):1538–1573, June 2019. ISSN 1382-3256. doi: 10.1007/s10664-018-9648-z. URL https://doi.org/10.1007/s10664-018-9648-z.

[26] Miltiadis Allamanis. The adverse effects of code duplication in machine learning models of code, 2019. URL https://arxiv.org/abs/1812.06469.

[27] Zhou Yang, Zhipeng Zhao, Chenyu Wang, Jieke Shi, Dongsun Kim, Donggyun Han, and David Lo. Unveiling memorization in code models. *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*, pages 867–879, 2023. URL https://api.semanticscholar.org/CorpusID:261048934.

[28] Zhou Yang, Zhipeng Zhao, Chenyu Wang, Jieke Shi, Dongsun Kim, Donggyun Han, and David Lo. Unveiling memorization in code models. *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*, pages 867–879, 2023. URL https://api.semanticscholar.org/CorpusID:261048934.

[29] Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John F. J. Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsen, Siddhant M. Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, L. Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsimpoukelli, N. K. Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Tobias Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew G. Johnson, Blake A. Hechtman, Laura Weidinger, Iason Gabriel, William S. Isaac, Edward Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem W. Ayoub, Jeff Stanway, L. L. Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. Scaling language models: Methods, analysis & insights from training gopher. *ArXiv*, abs/2112.11446, 2021. URL https://api.semanticscholar.org/CorpusID:245353475.

[30] Thomas Wolf, Leandro von Werra, and Lewis Tunstall. Codeparrot dataset. *https://huggingface.co/datasets/transformersbook/codeparrot*.

[31] Harald Foidl, Michael Felderer, and Rudolf Ramler. Data smells: categories, causes and consequences, and detection of suspicious data in ai-based systems. In *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*, CAIN '22, page 229–239, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392754. doi: 10.1145/3522664.3528590. URL https://doi.org/10.1145/3522664.3528590.

[32] Lin Shi, Fangwen Mu, Xiao Chen, Song Wang, Junjie Wang, Ye Yang, Ge Li, Xin Xia, and Qing Wang. Are we building on the rock? on the importance of data preprocessing for code summarization. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2022, page 107–119, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394130. doi: 10.1145/3540250.3549145. URL https://doi.org/10.1145/3540250.3549145.

[33] Jose Antonio Hernandez Lopez, Boqi Chen, Mootez Saad, Tushar Sharma, and Daniel Varro. On Inter-Dataset Code Duplication and Data Leakage in Large Language Models . *IEEE Transactions on Software Engineering*, 51(01):192–205, January 2025. ISSN 1939-3520. doi: 10.1109/TSE.2024.3504286. URL https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3504286.

[34] Github-code dataset. *https://huggingface.co/datasets/codeparrot/github-code*.

[35] Loubna Allal, Raymond Li, Denis Kocetkov, Chenghao Mou, Christopher Akiki, Carlos Ferrandis, Niklas Muennighoff, Mayank Mishra, Alex Gu, Manan Dey, Logesh Umapathi, Carolyn Anderson, Yangtian Zi, Joel Poirier, Hailey Schoelkopf, Sergey Troshin, Dmitry Abulkhanov, Manuel Romero, Michael Lappert, and Leandro Werra. Santacoder: don't reach for the stars!, 01 2023.

[36] 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, Joel Lamy-Poirier, Joao Monteiro, Nicolas Gontier, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Ben Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason T Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Urvashi Bhattacharyya, Wenhao Yu, Sasha Luccioni, Paulo Villegas, Fedor Zhdanov, Tony Lee, Nadav Timor, Jennifer Ding, Claire S Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, 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! *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=KoFOg41haE. Reproducibility Certification.

[37] Ali Al-Kaswan and Maliheh Izadi. The (ab)use of open source code to train large language models. pages 9–10, 05 2023. doi: 10.1109/NLBSE59153.2023.00008.

[38] Ali Al-Kaswan, Maliheh Izadi, and Arie van Deursen. Targeted attack on gpt-neo for the satml language model data extraction challenge, 2023. URL https://arxiv.org/abs/2302.07735.

[39] Ali Al-Kaswan, Maliheh Izadi, and Arie van Deursen. Traces of memorisation in large language models for code. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400702174. doi: 10.1145/3597503.3639133. URL https://doi.org/10.1145/3597503.3639133.

[40] Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. Quantifying memorization across neural language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=TatRHT_1cK.

[41] Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, Philip S. Yu, and Xuyun Zhang. Membership inference attacks on machine learning: A survey. *ACM Comput. Surv.*, 54(11s), September 2022. ISSN 0360-0300. doi: 10.1145/3523273. URL https://doi.org/10.1145/3523273.

[42] Sheng Zhang and Hui Li. Code membership inference for detecting unauthorized data use in code pre-trained language models. *arXiv preprint arXiv:2312.07200*, 2023.

[43] Yue Wang, Hung Le, Akhilesh Gotmare, Nghi Bui, Junnan Li, and Steven Hoi. CodeT5+: Open code large language models for code understanding and generation. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1069–1088, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.68. URL https://aclanthology.org/2023.emnlp-main.68/.

[44] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. Unixcoder: Unified cross-modal pre-training for code representation, 2022. URL https://arxiv.org/abs/2203.03850.

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

[46] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie LIU, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. Graphcode{bert}: Pre-training code representations with data flow. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=jLoC4ez43PZ.

[47] Zhou Yang, Zhipeng Zhao, Chenyu Wang, Jieke Shi, Dongsun Kim, DongGyun Han, and David Lo. Gotcha! this model uses my code! evaluating membership leakage risks in code models. *IEEE Transactions on Software Engineering*, 50(12):3290–3306, 2024. doi: 10.1109/TSE.2024.3482719.

[48] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, MING GONG, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie LIU. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. URL https://openreview.net/forum?id=6lE4dQXaUcb.

[49] Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Scott Yih, Luke Zettlemoyer, and Mike Lewis. Incoder: A generative model for code infilling and synthesis. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=hQwb-lbM6EL.

[50] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray,

Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

[51] Daphne Ippolito, Florian Tramer, Milad Nasr, Chiyuan Zhang, Matthew Jagielski, Katherine Lee, Christopher Choquette Choo, and Nicholas Carlini. Preventing generation of verbatim memorization in language models gives a false sense of privacy. In C. Maria Keet, Hung-Yi Lee, and Sina Zarrieß, editors, *Proceedings of the 16th International Natural Language Generation Conference*, pages 28–53, Prague, Czechia, September 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.inlg-main.3. URL https://aclanthology.org/2023.inlg-main.3/.

[52] Peter Henderson, Xuechen Li, Dan Jurafsky, Tatsunori Hashimoto, Mark A. Lemley, and Percy Liang. Foundation models and fair use, 2023. URL https://arxiv.org/abs/2303.15715.

[53] Rizwan Choudhury. Anti-piracy group shuts down books3, a popular dataset for ai models. *Interesting Engineering*, 2023. URL https://interestingengineering.com/innovation/anti-piracy-group-shuts-down-books3-a-popular-dataset-for-ai-models.

[54] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling, 2020. URL https://arxiv.org/abs/2101.00027.

[55] Rettigheds Alliancen. Rights alliance removes the illegal books3 dataset used to train artificial intelligence, August 14 2023. URL https://rettighedsalliancen.com/rights-alliance-removes-the-illegal-books3-dataset-used-to-train-artificial-intelligence/.

[56] Blue Oak Council. The blue oak guide to copyleft, 2025. URL https://blueoakcouncil.org/copyleft.

[57] The Apache Software Foundation. Apache license, version 2.0, 2004. URL https://www.apache.org/licenses/LICENSE-2.0.

[58] GNU Operating System. What is copyleft?, 2022. URL https://www.gnu.org/licenses/licenses.html#WhatIsCopyleft.

[59] Jos'e Antonio Hern'andez L'opez, Boqi Chen, Mootez Saad, Tushar Sharma, and D'aniel Varr'o. On inter-dataset code duplication and data leakage in large language models. *IEEE Transactions on Software Engineering*, 51:192–205, 2024. URL https://api.semanticscholar.org/CorpusID:266999336.

[60] Chanchal Roy and James Cordy. A survey on software clone detection research. *School of Computing TR 2007-541*, 01 2007.

[61] Chanchal K. Roy, James R. Cordy, and Rainer Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, 74(7):470–495, 2009. ISSN 0167-6423. doi: https://doi.org/10.1016/j.scico.2009.02.007. URL https://www.sciencedirect.com/science/article/pii/S0167642309000367.

[62] Siming Huang, Tianhao Cheng, J. K. Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J. Yang, J. H. Liu, Chenchen Zhang, Linzheng Chai, Ruifeng Yuan, Zhaoxiang Zhang, Jie Fu, Qian Liu, Ge Zhang, Zili Wang, Yuan Qi, Yinghui Xu, and Wei Chu. Opencoder: The open cookbook for top-tier code large language models, 2024. URL https://arxiv.org/abs/2411.04905.

[63] Cristina V. Lopes, Petr Maj, Pedro Martins, Vaibhav Saini, Di Yang, Jakub Zitny, Hitesh Sajnani, and Jan Vitek. Déjàvu: a map of code duplicates on github. *Proc. ACM Program. Lang.*, 1(OOPSLA), October 2017. doi: 10.1145/3133908. URL https://doi.org/10.1145/3133908.

[64] Yaroslav Golubev and Timofey Bryksin. On the Nature of Code Cloning in Open-Source Java Projects . In *2021 IEEE 15th International Workshop on Software Clones (IWSC)*, pages 22–28, Los Alamitos, CA, USA, October 2021. IEEE Computer Society. doi: 10.1109/IWSC53727.2021.00010. URL https://doi.ieeecomputersociety.org/10.1109/IWSC53727.2021.00010.

[65] Joel Ossher, Hitesh Sajnani, and Cristina Lopes. File cloning in open source java projects: The good, the bad, and the ugly. pages 283–292, 09 2011. doi: 10.1109/ICSM.2011.6080795.

[66] Mohammad Gharehyazie, Baishakhi Ray, and Vladimir Filkov. Some from here, some from there: Cross-project code reuse in github. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 291–301, 2017. doi: 10.1109/MSR.2017.15.

[67] Di Yang, Pedro Martins, Vaibhav Saini, and Cristina Lopes. Stack overflow in github: any snippets there? In *Proceedings of the 14th International Conference on Mining Software Repositories*, MSR '17, page 280–290. IEEE Press, 2017. ISBN 9781538615447. doi: 10.1109/MSR.2017.13. URL https://doi.org/10.1109/MSR.2017.13.

[68] Danny Hernandez, Tom Brown, Tom Conerly, Nova DasSarma, Dawn Drain, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Tom Henighan, Tristan Hume, Scott Johnston, Ben Mann, Chris Olah, Catherine Olsson, Dario Amodei, Nicholas Joseph, Jared Kaplan, and Sam McCandlish. Scaling laws and interpretability of learning from repeated data, 2022. URL https://arxiv.org/abs/2205.10487.

[69] Cheng Xu, Shuhao Guan, Derek Greene, and Tahar Kechadi. Benchmark data contamination of large language models: A survey, 06 2024.

[70] Mathieu Ravaut, Bosheng Ding, Fangkai Jiao, Hailin Chen, Xingxuan Li, Ruochen Zhao, Chengwei Qin, Caiming Xiong, and Shafiq R. Joty. How much are large language models contaminated? a comprehensive survey and the llmsanitize library. 2024. URL https://api.semanticscholar.org/CorpusID:268819579.

[71] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

[72] Martin Riddell, Ansong Ni, and Arman Cohan. Quantifying contamination in evaluating code generation capabilities of language models. In *Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://api.semanticscholar.org/CorpusID:268297237.

[73] Shuo Yang, Wei-Lin Chiang, Lianmin Zheng, Joseph E. Gonzalez, and Ion Stoica. Rethinking benchmark and contamination for language models with rephrased samples, 2023. URL https://arxiv.org/abs/2311.04850.

[74] Jae Yong Lee, Sungmin Kang, Juyeon Yoon, and Shin Yoo. The github recent bugs dataset for evaluating llm-based debugging applications, 2023. URL https://arxiv.org/abs/2310.13229.

[75] Aaditya K. Singh, Muhammed Yusuf Kocyigit, Andrew Poulton, David Esiobu, Maria Lomeli, Gergely Szilvasy, and Dieuwke Hupkes. Evaluation data contamination in llms: how do we measure it and (when) does it matter?, 2024. URL https://arxiv.org/abs/2411.03923.

[76] Kun Zhou, Yutao Zhu, Zhipeng Chen, Wentong Chen, Wayne Xin Zhao, Xu Chen, Yankai Lin, Ji-Rong Wen, and Jiawei Han. Don't make your llm an evaluation benchmark cheater. *ArXiv*, abs/2311.01964, 2023. URL https://api.semanticscholar.org/CorpusID:265019021.

[77] Andrei Z. Broder. Identifying and filtering near-duplicate documents. In Raffaele Giancarlo and David Sankoff, editors, *Combinatorial Pattern Matching*, pages 1–10, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-45123-5.

[78] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, USA, 2nd edition, 2014. ISBN 1107077230.

[79] Md Tahmid Rahman Laskar, Sawsan Alqahtani, M Saiful Bari, Mizanur Rahman, Mohammad Abdullah Matin Khan, Haidar Khan, Israt Jahan, Amran Bhuiyan, Chee Wei Tan, Md Rizwan Parvez, Enamul Hoque, Shafiq Joty, and Jimmy Huang. A systematic survey and critical review on evaluating large language models: Challenges, limitations, and recommendations. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 13785–13816, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.764. URL https://aclanthology.org/2024.emnlp-main.764/.

[80] Joshua Bloch and Pamela Samuelson. Some misconceptions about software in the copyright literature. In *Proceedings of the 2022 Symposium on Computer Science and Law*, CSLAW '22, page 131–141, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392341. doi: 10.1145/3511265.3550449. URL https://doi.org/10.1145/3511265.3550449.

[81] Tolga Bolukbasi, Adam Pearce, Ann Yuan, Andy Coenen, Emily Reif, Fernanda Vi'egas, and Martin Wattenberg. An interpretability illusion for bert. *ArXiv*, abs/2104.07143, 2021. URL https://api.semanticscholar.org/CorpusID:233241181.

[82] Software Freedom Conservancy. If Software is My Copilot, Who Programmed My Software?, February 2022. URL https://sfconservancy.org/blog/2022/feb/03/github-copilot-copyleft-gpl/. Accessed: 2025-02-14.

[83] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. Software testing with large language model: Survey, landscape, and vision. *arXiv preprint arXiv:2307.07221*, 2023.

[84] Lizhou Fan, Lingyao Li, Zihui Ma, Sanggyu Lee, Huizi Yu, and Libby Hemphill. A bibliometric review of large language models research from 2017 to 2023. *arXiv preprint arXiv:2304.02020*, 2023.

[85] 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 arXiv:2308.10620*, 2023.

[86] Daoguang Zan, Bei Chen, Fengji Zhang, Dianjie Lu, Bingchao Wu, Bei Guan, Wang Yongji, and Jian-Guang Lou. Large language models meet NL2Code: A survey. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7443–7464, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.411. URL https://aclanthology.org/2023.acl-long.411.

[87] 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 arXiv:2310.03533*, 2023.

[88] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022. URL https://arxiv.org/abs/2112.10752.

[89] Jonathan Katzy, Erik Mekkes, Razvan Mihai Popescu, Maliheh Izadi, and Arie van Deursen. Automated attention pattern discovery at scale in large language models, 2025. Under Review.

[90] Software heritage. *https://docs.softwareheritage.org/index.html*.

[91] Gharchive. *https://www.gharchive.org/*.

[92] Google bigquery. *https://cloud.google.com/bigquery/public-data*.

[93] Fabio Salerno, Ali Al-Kaswan, and Maliheh Izadi. How much do code language models remember? an investigation on data extraction attacks before and after fine-tuning, 01 2025.

[94] Xinwei Wu, Junzhuo Li, Minghui Xu, Weilong Dong, Shuangzhi Wu, Chao Bian, and Deyi Xiong. DEPN: Detecting and editing privacy neurons in pretrained language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2875–2886, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.174. URL https://aclanthology.org/2023.emnlp-main.174/.

[95] 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, 2023. URL https://arxiv.org/abs/2310.16787.

[96] Pratyush Maini, Hengrui Jia, Nicolas Papernot, and Adam Dziedzic. LLM dataset inference: Did you train on my dataset? In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=Fr9d1UMc37.

[97] Daniel Germán and Ahmed E. Hassan. License integration patterns: Addressing license mismatches in component-based development. pages 188–198, 01 2009. doi: 10.1109/ICSE.2009.5070520.

[98] Julius Davies, Daniel M. German, Michael W. Godfrey, and Abram Hindle. Software bertillonage: finding the provenance of an entity. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, page 183–192, New York, NY,

USA, 2011. Association for Computing Machinery. ISBN 9781450305747. doi: 10.1145/1985441.1985468. URL https://doi.org/10.1145/1985441.1985468.

[99] Timo Pawelka and Elmar Jürgens. Is this code written in english? a study of the natural language of comments and identifiers in practice. *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 401–410, 2015. URL https://api.semanticscholar.org/CorpusID:6831029.

[100] Alebachew Chiche and Betselot Yitagesu. Part of speech tagging: a systematic review of deep learning and machine learning approaches. *Journal of Big Data*, 9(1):10, 2022.