Towards Safe, Secure, and Usable LLMs4Code

Al-Kaswan, Ali

DOI
10.1145/3639478.3639803

Publication date
2024

Document Version
Final published version

Published in
Proceedings - 2024 ACM/IEEE 46th International Conference on Software Engineering

Citation (APA)

Important note
To cite this publication, please use the final published version (if applicable). Please check the document version above.
Towards Safe, Secure, and Usable LLMs4Code

Ali Al-Kaswan
a.al-kaswan@tudelft.nl
Delft University of Technology
Delft, The Netherlands

ABSTRACT
Large Language Models (LLMs) are gaining popularity in the field of Natural Language Processing (NLP) due to their remarkable accuracy in various NLP tasks. LLMs designed for coding are trained on massive datasets, which enables them to learn the structure and syntax of programming languages. These datasets are scraped from the web and LLMs memorise information in these datasets. LLMs for code are also growing, making them more challenging to execute and making users increasingly reliant on external infrastructure. We aim to explore the challenges faced by LLMs for code and propose techniques to measure and prevent memorisation. Additionally, we suggest methods to compress models and run them locally on consumer hardware.

CCS CONCEPTS

KEYWORDS
large language models, privacy, memorisation, data leakage, compression

ACM Reference Format:

1 INTRODUCTION
In recent years, Large Language Models (LLMs) have become increasingly popular in Natural Language Processing (NLP) due to their impressive accuracy in a wide range of NLP tasks [1]. As the number of parameters in these models increases from millions to billions, their accuracy and capabilities also improve [2]. LLMs designed for coding (LLMs4Code) are trained on large datasets and can learn the structure and syntax of programming languages. As a result, they are highly proficient in tasks such as generating [3], summarising [4], and completing code [5].

The appeal of scaling up LLMs is the discovery of emergent capabilities [6]. Emergent capabilities cannot be anticipated by extrapolating scaling laws and only become visible at a certain critical model size threshold [6]. This encourages the training of ever-larger models, as abilities such as chain-of-thought prompting [7] and instruction tuning [8] can only be achieved in models with more than 100B parameters [6]. However, this increase in parameter counts makes it increasingly difficult to deploy and run LLMs. Many state-of-the-art open source LLMs4Code such as CodeLlama [9] and WizardCoder [10] cannot be executed on consumer GPUs with less than 32GB of VRAM1.

This excludes many from being able to use current state-of-the-art LLMs. Those who cannot afford the hardware to deploy the models must rely on external services to run the models, such as GitHub Copilot. From a privacy and security perspective, this is not always desirable. Firstly, the source code might contain all types of information about the developer, which is then sent to an external party. Secondly, some organisations do not allow their proprietary source code to leave their premises.

It has also been observed that large language models trained on natural language can memorise and regurgitate training data [11–22]. This issue has not been fully explored for code. Moreover, the issue of memorisation in the source code is different from that of natural language.

The open source code used in LLM training for code is often licenced under non-permissive copy-left licences, such as GPL or the CC-BY-SA licence employed by StackOverflow [11]. Reusing code covered by these licences without making the source code available under the same licence is considered a violation of copyright law. In some jurisdictions, this leaves users of tools such as CoPilot at legal risk [11, 15, 23]. Sharing code without proper licences is also ethically questionable [11, 15, 16].

Memorised data can also include confidential information [24–26], which can include credentials, API keys, emails, and other sensitive data [11, 27]. This means that memorisation could put the private information contained in the training data at risk. Recently, attacks which exploit memorisation have been able to extract (or reconstruct) training data from LLMs [19, 22, 24, 28]. The US National Institute of Standards and Technology (NIST) considers data reconstruction attacks to be the most serious type of privacy attack against machine learning models [29]. OWASP classifies Sensitive Information Disclosure (LLM06) as the sixth most critical vulnerability in LLM applications.2

We propose an approach to measure the rate at which memorisation occurs in LLMs4Code. We then measure the rate at which memorisation occurs for PII and copyrighted code. These findings will then be used to inform dataset construction and model training techniques to prevent memorisation. In parallel, we will also investigate approaches to compress LLMs4Code and the impact of compression on memorisation.

1Can you run it? LLM version: https://huggingface.co/spaces/Vokturz/can-it-run-llm
2OWASP Top 10 for Large Language Model Applications: https://owasp.org/www-project-top-10-for-large-language-model-applications/


2 BACKGROUND AND RELATED WORK

2.1 Memorisation

Memorisation in language models is the capacity to remember and recall details of the data it has been trained on. This happens when the model is too specific and does not generalise well to new or unseen data [20, 30]. As a result, the model can accurately reproduce phrases, sentences, or even entire documents from the training data. Apart from the privacy issues discussed in section 1, memorisation also leads to an overestimation of performance. For example, Codex has been observed to be able to solve HackerRank problems without receiving the full task description [18].

Memorisation can lead to high accuracy, but it does not necessarily mean that the model will generalise well to new or unseen data. This can lead to poor performance in real-world applications. Furthermore, memorisation can reduce the model’s ability to adjust its output to particular use cases. For instance, when slightly altering HackerRank problems, Codex [31] has difficulty producing the correct solution, instead repeating the answer for the original problem [18, 32].

2.2 Data Extraction Attacks

Data extraction attacks are a type of attack in which an adversary extracts a data point from the training data of a model. Attacks can be divided into two types for LLMs, namely guided and unguided attacks [28].

In an unguided attack, the adversary does not know the sample to be extracted from the model. The adversary simply attempts to extract any training point, contained anywhere in the training corpus [14, 24, 25, 33]. Targeted attacks are more security-critical as they allow the targeting of specific information, such as the extraction of emails [15, 25, 28, 34, 35].

2.3 Model Compression

Model Compression for LLMs can roughly be divided into three techniques. Namely, knowledge distillation, pruning, and quantisation [36, 37].

Knowledge distillation transfers the knowledge of the large teacher model to a smaller and simpler student model [36]. Pruning reduces the size of the model by removing unneeded parameters [36, 38, 39]. Quantisation is a relatively simple technique that reduces the precision of the model by reducing floating point numbers to integers or smaller representations [36, 40, 41].

A number of methods, including XTC [37], have been developed to combine multiple techniques to achieve a higher compression rate. While these hybrid approaches have been used to compress models from the natural language domain, their application to software models has yet to be fully explored.

3 APPROACH

First, we explore the different risks and implications posed by LLMs4Code. In our position paper, we map the existing privacy problems in LLMs to the source code domain. We also identify other code-specific issues, namely licensing and security [11].

3.1 Memorisation

To measure memorisation, we create a set of potentially extractable samples for a given model using a targeted data extraction attack. The process of finding memorised data is relatively simple [28], by changing the number of input tokens, we can change the difficulty of the sample, which in turn allows us to compare the rate between different models and prompting techniques. This work has already been completed and was accepted into the main ICSE track.

Using this framework for measurement, we can extend the evaluation to also look at specific types of data. Using techniques like those described by Niu et al. [42], we can identify code that contains PII and use it as input to our evaluation. We can similarly extend our evaluation to include copyrighted code as well.

Based on these findings, we can identify patterns that elicit memorisation in LLMs4Code and can put the user at risk. These patterns can then be used to design datasets and training regimes that reduce the memorisation rate.

3.2 Compression

Finally, for the Model Compression, we plan to adapt different techniques for compression from the natural language domain to code. We measure the parameter count, disk size, size in VRAM, inference time and accuracy for each given model and compression technique. We further investigate the impact of compressing the LLMs on the rate of memorisation, and the relation between overparametrisation and memorisation.

4 EXPECTED CONTRIBUTIONS

As Large Language Models for Code (LLMs4Code) continue to gain widespread adoption, our research aims to enhance their usability and instil trust among users. By developing robust techniques for measuring memorisation in LLMs4Code, we empower users with the knowledge to make well-informed decisions regarding the models they choose to employ.

Our research contributes to the evolution of LLMs4Code by addressing concerns related to memorisation, thereby reducing the likelihood and associated risks of unintended memorisation in model outputs. This proactive approach ensures that users can have confidence in the reliability and generalisation capabilities of the models they rely on, fostering a more secure and dependable ecosystem for utilising LLMs4Code.

Moreover, our work on compressing LLMs is a significant step towards democratizing access to these powerful tools by substantially reducing the hardware requirements traditionally associated with their deployment. Our efforts will make LLMs4Code more accessible to a broader audience, paving the way for wider adoption and greater participation, which would enable more individuals to benefit from the use of LLMs4Code.

REFERENCES


[2] Language Models Training Data Extraction Challenge: https://github.com/google-research/lm-extraction-benchmark