Black-Box Reconstruction Attacks on LLMs: A Preliminary Study in Code Summarization

Marco Russodivito[1], Angelica Spina[1], Simone Scalabrino[1], and Rocco Oliveto[1]

University of Molise, Italy {marco.russodivito, angelica.spina, simone.scalabrino, rocco.oliveto}@unimol.it

Abstract. Large Language Models (LLMs) have demonstrated effectiveness in tackling coding tasks, leading to their growing popularity in commercial solutions like GitHub Copilot and ChatGPT. These models, however, may be trained on proprietary code, raising concerns about potential leaks of intellectual property. A recent study indicates that LLMs can memorize parts of the source code, rendering them vulnerable to extraction attacks. However, it used *white-box* attacks which assume that adversaries have partial knowledge of the training set.
This paper presents a pioneering effort to conduct a *black-box* attack (reconstruction attack) on an LLM designed for a specific coding task – code summarization. The results achieved reveal that while the attack is generally unsuccessful (with an average *BLEU score* below 0.1), it succeeds in a few instances, reconstructing versions of the code that closely resemble the original.

Keywords: LLMs for Coding Tasks · Security · Reconstruction Attacks.

1 Introduction

Large Language Models (LLMs) are increasingly integral to software engineering research, proving particularly effective in coding tasks such as bug fixing and code summarization [14]. These models operate by inputting a sequence of textual tokens and outputting another sequence, which may include source code or natural language depending on the task. Modern LLMs employed for coding typically leverage the Transformer architecture [21] and undergo a two-stage training process: initial *pre-training* on extensive codebases to learn language patterns semi-supervised, followed by task-specific *fine-tuning* in a supervised manner (*i.e.,* by providing examples of input and output sequences).

The effectiveness of these models has led to their integration into commercial tools like CoPilot[1] and ChatGPT[2], which are becoming indispensable coding assistants [26]. While research-oriented LLMs are often trained on open-source datasets, commercial variants might use proprietary code that companies wish to keep confidential.

[1] https://www.microsoft.com/en-us/microsoft-copilot (Verified on April 26th, 2024)
[2] https://chat.openai.com

Consider a company that wants to develop a coding assistant from an LLM fine-tuned on its proprietary data from 1,000 software projects and over 5,000 developers, aiming to commercialize this tool without exposing the underlying source code. This introduces potential confidentiality issues if methods exist that can extract specific data instances from the trained model without any other dataset information.

Let us call the (confidential) dataset used for fine-tuning the LLM D_p. If there is a methodology that can extract any instance $p \in D_p$ given only the trained model (as a *black-box*) and no other instance of D_p, we say that the LLM violates confidentiality. Al-Kaswan *et al.* [2] recently proved that LLMs for coding tasks memorize token sequences. Especially, the authors tested whether such LLMs are vulnerable to *white-box* attacks, which assume that the adversary has partial knowledge of the D_p. In most cases, however, adversaries might not have such information. Previous work highlighted that other types of privacy attacks are possible for DNNs [7, 18–20, 22]. *Reconstruction Attacks* (or *Model Inversion Attack*) are a specific type of such attacks that aim to recreate one or more training samples [18] assuming that the adversary has no information about the training instances. It is still unclear, however, to what extent such attacks are effective on LLMs for coding tasks.

In this paper, we present an empirical study assessing the feasibility of a *black-box* attack (*Reconstruction Attack*) to compromise confidentiality in LLMs used for coding tasks. We specifically analyze the T5 model by Mastropaolo *et al.* [14] trained for the task of code summarization ($T5_{summ}$), where the model generates textual descriptions from code snippets. Our experiment involved constructing an *inverse model* that takes textual descriptions and attempts to output original code snippets. To build an *inverse model* for $T5_{summ}$ ($T5^{-1}_{summ}$), we first collected a set S of ∼700k open-source code snippets from state-of-the-art datasets [11–13]. Then, we gave each snippet $s \in S$ to the trained $T5_{summ}$, which generated its textual summary s_c and trained $T5^{-1}_{summ}$ on the pairs $\langle s_c, s \rangle$. Finally, we tried to extract the code snippets in D_p by giving textual summaries s_c to $T5^{-1}_{summ}$.

The obtained results revealed that in almost all the cases it is not possible to extract training instances (*BLEU score* < 0.1). Still, we found some cases in which the *BLEU score* is high and the attack generated closely-resembling versions of the source code in D_p. Our results call for future work aimed at investigating more in depth the possibility of running such attacks on other tasks and propose solutions to mitigate such a problem.

2 Related Work

Reconstruction Attacks have been extensively studied in both *black-box* and *white-box* settings. In the former, the adversary lacks information about the model parameters, architecture, and training data. In the latter, the adversary has complete knowledge of the target model [18]. Such a concept was first introduced by Fredrikson *et al.* [6], who showed that an attacker could predict a patient's genetic markers using only demographic information and model access.

This led to a broader application of model inversion attacks, as demonstrated by Fredrikson *et al.* [5] in subsequent work. In this work, the authors proposed a more generic model inversion attack applicable either in *black-box* or *white-box* settings and against different models (*e.g.,* decision trees and neural networks for face recognition) which exploits confidence values obtained from prediction APIs. Hidano *et al.* [9] introduced a *black-box* technique targeting online prediction systems that adapt over time, demonstrating that these systems could be influenced by poisoning attacks to make them more susceptible to *Reconstruction Attack*. Some attacks can even involve Generative Adversarial Networks (GANs) [10]. Their approach uses real-time learning in collaborative deep learning models to train a GAN to generate private and prototypical examples of the training set. Another example is the attack by Zhang *et al.* [24], which leverages partial public data (*e.g.,* blurred or corrupted images) to learn a distributional prior using GANs and employs it to guide the inversion process.

Recently, many studies have focused on the security of LLMs, to understand the degree of training data storage that these models achieve. This type of attacks is called *Training Data Extraction Attacks*. Like Model Inversion Attacks, Training Data Extraction Attacks aim to reconstruct training samples, but in a *verbatim* way [4, 25]. A popular approach [4] consists of two main steps: a generation phase where different potential suffixes are obtained by querying several times the model with a set of prefixes, and an inference step used to infer if each suffix, or just some of them [1], belongs to the target training set. This can be done in different ways: one can perform fuzzy match like 3-gram fuzzy match or using BLEU-4 score, he can apply an exact match, or even a combination of all the techniques [2, 4]. The exact match check also includes the case of Type-1 clone detection to identify code memorization presented by Yang *et al.* [22], which tries to reconstruct small snippets belonging to the training set. Another interesting approach involves the use of canaries, *i.e.,* the insertion of N clones of predefined samples into the training set, which researchers then attempt to reconstruct. This reconstruction can be performed either by employing a canary completion task [16] or by exploiting the *exposure* concept illustrated by Carlini *et al.* [3].

Yang *et al.* [23] tested the execution of a *Reconstruction Attack* for Large Language Models. Such an attack (i) is a *black-box* attack that works without special assumptions on adversary knowledge; (ii) can be applied at inference time (*i.e.,* when the target model is already trained); (iii) is effective against complex models, such as CNNs. To the best of our knowledge, our work is the first attempt to use an adaptation of the work by Yang *et al.* [23] to exploit LLMs for coding tasks.

3 Empirical Study Design

Our study aims to determine the feasibility of a *Reconstruction Attack* on LLMs for code summarization tasks. addressing the following Research Question (RQ):

To what extent is a code summarization model vulnerable to a Reconstruction Attack?

3.1 The Reconstruction Attack Method

In the context of our study we exploit the *Reconstruction Attack* methodology proposed by Yang *et al.* [23], originally designed for CNNs, adapting it to LLMs for coding tasks. The core concept involves training an *inverse model* ($M_{inverse}$) that functions oppositely to the target model (M_{target}), enabling the extraction of training instances by querying $M_{inverse}$. We assume that the adversary lacks knowledge of the dataset used to train M_{target} (*i.e.,* the one we want to reconstruct), and she has *black-box* access to M_{target}, allowing her to input data and receive prediction outputs. Additionally, the adversary understands the semantics of the input and has access to the *tokenizer* used by M_{target}, which converts textual data into numerical vectors. We assume that the target model uses a publicly available tokenizer that the adversary can access.

The attack process is structured around the phases of the *Kill Chain*. Initially, in the *Reconnaissance phase*, the adversary analyzes the target model to understand its input and output characteristics. Next, during the *Weaponization phase*, the adversary compiles an auxiliary set (D_{aux}) of semantically compatible samples, in our case, source code snippets. This set is used to probe M_{target}, creating an attack set (D_{atk}) during the *Delivery phase*. This set consists of pairs $\langle M_{target}(a), a\rangle$, which are then used to train the *inverse model* ($M_{inverse}$) in the Exploitation phase. Ultimately, the adversary queries $M_{inverse}$ with the intention of reconstructing original training instances.

3.2 Context of the Study

The context of our study is composed of two types of *objects*: (i) a target model and a dataset that we use as the *auxiliary dataset*.

As for the former, we focus on the T5 model [17] adapted to support coding tasks [14]. The authors introduced specialized models to supports four tasks: *Automatic Bug Fixing*, *Injection of Code Mutants*, *Generation of Assertions*, and *Code Summarization*. We focus on the model that tackles the code-summarization task, $T5_{summ}$. The dataset we want to reconstruct (*i.e.,* the one used to train $T5_{summ}$) is a variation of the dataset provided by Haque *et al.* [8]. Specifically, the instances of their dataset are tuples $\langle s, a_s, c_s, d\rangle$, where s is the source code, a_s is the AST, c_s is the remaining code of the Java class, and d is the textual description of the method. Instead, our set consists of pairs $\langle s, d\rangle$. We focus on code summarization because it is the only task that assumes that the adversary has no information about the source code. Ideally, the adversary only needs a textual description that is close to the one used in the training set to retrieve the protected source code.

We use as the *auxiliary dataset* a combination of three state-of-the-art dataset of code snippets. The first one is CodeSearchNet [13], which contains about 6M functions from open-source code involving six different programming languages. We only focus on Java instances. The second one is TL-CodeSum [12], which has been collected from GitHub mining Java projects with at least 20 stars and create from 2015 to 2016.

Such a dataset is frequently used for code-summarization tasks, but it is different from the dataset used in the target model. The third one is DeepCom [11], which has been built from 9,714 open-source projects from GitHub and is used as a benchmark for comment generation tasks. We merged these datasets, removing duplicate entries and code comments, and leveraged the tokenizer from the target model to adapt all auxiliary set samples to the input format of the target model. After preprocessing, the final auxiliary dataset comprised 722,067 instances.

3.3 Experimental Procedure

To construct the *inverse model* of $T5_{summ}$ ($T5^{-1}_{summ}$), we chose to use the same architecture of the *target model* (*i.e.,* T5). We fine-tuned such a model using the *attack dataset* built using the procedure reported in Section 3.1.

In order to address our research question, we optimistically assume that the adversary can access the exact summaries used in the training set. We then run $T5^{-1}_{summ}$ once for each summary and evaluate the similarity between the code generated by $T5^{-1}_{summ}$ and the original code from the training set. It is important to note that this scenario relaxes the conditions described in Section 3.1, as it assumes prior knowledge about the training set. Therefore, our findings should be considered an upper bound of potential outcomes, given that such information might not be available to an adversary in practice.

To quantify the reconstruction quality, we utilized the *BLEU score* (Bilingual Evaluation Understudy) [15], which measures the similarity of n-grams between the reconstructed and original sequences on a scale from 0 to 1. A value closer to 1 indicates a higher similarity between the two code snippets.

4 Empirical Study Results

This section provides a discussion on the results achieved and the limitations of our study.

4.1 Analysis of the Results

Figure 1 shows the distribution of the *BLEU score* between the reconstructed code and the actual code in the training set. The average score is lower than 0.1. Nevertheless, there are several outliers with relatively high *BLEU score*, suggesting that our reconstruction attack can sometimes recover significant portions of the original training data.

We conducted a manual analysis of some reconstructions with high *BLEU score* to determine if the reconstruction attack was successful in these instances. Figure 2 presents examples of Java methods that were partially reconstructed. Notably, all examples are concise (three lines of code). Even though the reconstructions are not perfect, they always allow to get the core logic of the methods. In detail, Figure 2-a displays a method with a different (but semantically similar) name.

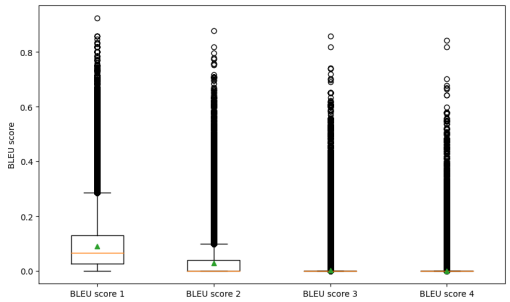

Fig. 1: Box plots for the *BLEU score* among all performed *Reconstruction Attacks*.

(a) PROMPT: Adds new attribute

ORIGINAL

```
public void putAttribute(String name, String value){
    attributes.put(name, value);
}
```

RECONSTRUCTED

```
public void addAttribute(String name, String value){
    attributes.put(name, value);
}
```

(b) PROMPT: Log the given message with the tag of info

ORIGINAL

```
public void info(String message) {
    writeInfoToken(message);
}
```

RECONSTRUCTED

```
public static void log(String message) {
    log(message, null);
}
```

(c) PROMPT: Deletes the named file

ORIGINAL

```
public void delete(String filename) {
    new File(filename).delete();
}
```

RECONSTRUCTED

```
public void deleteFile(String filename)
                                    throws IOException {
  if (filename == null) {
    throw new IllegalArgumentException("File name cannot be null");
  }
  if (filename.endsWith(File.separator)) {
    filename = filename.substring(0,
                    filename.length() - File.separator.length());
  }
  new File(filename).delete();
}
```

Fig. 2: Examples of partially reconstructed Java methods.

The reconstructed code in Figure 2-b is more generic compared to the reference one. The reconstruction in Figure 2-c exhibits additional logic in method body. These findings indicate that while the reconstruction attack does not always replicate the training data verbatim, in some cases it is able to produce meaningful and logically coherent code snippets.

4.2 Threats to Validity

As for the threats to **construct validity**, we assumed the adversary could access the tokenizer to encode and decode raw input and output data. In practical scenarios, an adversary might might not have such a piece of information. In terms of **internal validity**, merging three datasets to create the auxiliary set (*auxiliary dataset*) might introduce duplicates, as all datasets were sourced from GitHub. However, we found that the overlap of common samples was under 4%, and these were subsequently removed. Finally, as for the threats to **external validity**, our study focused on a single code summarization model. Testing the attack against other models could enhance the generalizability of our results.

5 Conclusion and Future Work

We present an initial exploration of performing a *Reconstruction Attack* on an LLM designed for a coding task – specifically, code summarization – with the objective of reconstructing the source code used for training. The results indicate that the attack fails on most instances, suggesting that state-of-the-art LLMs for coding tasks largely maintain confidentiality and do not disclose source code. However, there were some instances where the attack successfully reconstructed versions of the source code that closely resembled the original. Future research should focus on developing training methods for LLMs that further reduce the likelihood of successful reconstruction attacks, even in these rare successful instances.

Acknowledgment

This publication is part of the project PNRR-NGEU which has received funding from the MUR – DM 118/2023. This work has been supported by the European Union - NextGenerationEU through the Italian Ministry of University and Research, Projects PRIN 2022 "QualAI: Continuous Quality Improvement of AI-based Systems", grant n. 2022B3BP5S , CUP: H53D23003510006.

References

1. Al-Kaswan, A., Izadi, M., van Deursen, A.: Targeted attack on GPT-neo for the SATML language model data extraction challenge. arXiv:2302.07735 (2023)
2. Al-Kaswan, A., Izadi, M., Van Deursen, A.: Traces of memorisation in large language models for code. In: IEEE/ACM International Conference on Software Engineering. pp. 1–12 (2024)
3. Carlini, N., Liu, C., Erlingsson, Ú., Kos, J., Song, D.: The secret sharer: Evaluating and testing unintended memorization in neural networks. In: USENIX Security Symp. pp. 267–284 (2019)
4. Carlini, N., Tramer, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., Roberts, A., Brown, T., Song, D., Erlingsson, U., et al.: Extracting training data from large language models. In: USENIX Security Symp. pp. 2633–2650 (2021)
5. Fredrikson, M., Jha, S., Ristenpart, T.: Model inversion attacks that exploit confidence information and basic countermeasures. In: ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 1322–1333 (2015)
6. Fredrikson, M., Lantz, E., Jha, S., Lin, S., Page, D., Ristenpart, T.: Privacy in pharmacogenetics: An {End-to-End} case study of personalized warfarin dosing. In: USENIX Security Symp. pp. 17–32 (2014)
7. Ganju, K., Wang, Q., Yang, W., Gunter, C.A., Borisov, N.: Property inference attacks on fully connected neural networks using permutation invariant representations. In: Computer and Communications Security Conference. pp. 619–633 (2018)
8. Haque, S., LeClair, A., Wu, L., McMillan, C.: Improved automatic summarization of subroutines via attention to file context. In: International Conference on Mining Software Repositories. pp. 300–310 (2020)

9. Hidano, S., Murakami, T., Katsumata, S., Kiyomoto, S., Hanaoka, G.: Model inversion attacks for prediction systems: Without knowledge of non-sensitive attributes. In: IEEE International Conference on Privacy, Security, and Trust. pp. 115–11509 (2017)
10. Hitaj, B., Ateniese, G., Perez-Cruz, F.: Deep models under the gan: information leakage from collaborative deep learning. In: ACM SIGSAC Conference on Computer and Communications Security. pp. 603–618 (2017)
11. Hu, X., Li, G., Xia, X., Lo, D., Jin, Z.: Deep code comment generation. In: IEEE/ACM International Conference on Program Comprehension. pp. 200–210 (2018)
12. Hu, X., Li, G., Xia, X., Lo, D., Lu, S., Jin, Z.: Summarizing source code with transferred api knowledge. In: International Joint Conference on Artificial Intelligence. p. 2269–2275 (2018)
13. Husain, H., Wu, H.H., Gazit, T., Allamanis, M., Brockschmidt, M.: Codesearchnet challenge: Evaluating the state of semantic code search. arXiv:1909.09436 (2019)
14. Mastropaolo, A., Scalabrino, S., Cooper, N., Palacio, D.N., Poshyvanyk, D., Oliveto, R., Bavota, G.: Studying the usage of text-to-text transfer transformer to support code-related tasks. In: IEEE/ACM International Conference on Software Engineering. pp. 336–347 (2021)
15. Papineni, K., Roukos, S., Ward, T., Zhu, W.J.: BLEU: a method for automatic evaluation of machine translation. In: Annual Meeting of the ACL. pp. 311–318 (2002)
16. Parikh, R., Dupuy, C., Gupta, R.: Canary extraction in natural language understanding models. In: Annual Meeting of the ACL (2022)
17. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J.: Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of Machine Learning Research (JMLR) **21**(1), 5485–5551 (2020)
18. Rigaki, M., Garcia, S.: A survey of privacy attacks in machine learning. ACM Computing Surveys **56**(4), 1–34 (2023)
19. Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models. In: IEEE Symposium on Security and Privacy (SP). pp. 3–18 (2017)
20. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction APIs. In: USENIX Security Symp. pp. 601–618 (2016)
21. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
22. Yang, Z., Zhao, Z., Wang, C., Shi, J., Kim, D., Han, D., Lo, D.: Unveiling memorization in code models. In: IEEE/ACM International Conference on Software Engineering. pp. 856–856 (2024)
23. Yang, Z., Zhang, J., Chang, E.C., Liang, Z.: Neural network inversion in adversarial setting via background knowledge alignment. In: ACM SIGSAC Conference on Computer and Communications Security. pp. 225–240 (2019)
24. Zhang, Y., Jia, R., Pei, H., Wang, W., Li, B., Song, D.: The secret revealer: Generative model-inversion attacks against deep neural networks. In: IEEE/CVF Computer Vision and Pattern Recognition Conference. pp. 253–261 (2020)
25. Zhang, Z., Wen, J., Huang, M.: ETHICIST: Targeted training data extraction through loss smoothed soft prompting and calibrated confidence estimation. In: Annual Meeting of the ACL (2023)
26. Ziegler, A., Kalliamvakou, E., Li, X.A., Rice, A., Rifkin, D., Simister, S., Sittampalam, G., Aftandilian, E.: Measuring github copilot's impact on productivity. Communication of ACM **67**(3), 54–63 (2024)