Gotcha! This Model Uses My Code! Evaluating Membership Leakage Risks in Code Models

Zhou Yang, Zhipeng Zhao, Chenyu Wang, Jieke Shi, Dongsun Kim, DongGyun Han, and David Lo, *Fellow, IEEE*

***Abstract*—Leveraging large-scale datasets from open-source projects and advances in large language models, recent progress has led to sophisticated code models for key software engineering tasks, such as program repair and code completion. These models are trained on data from various sources, including public open-source projects like GitHub and private, confidential code from companies, raising significant privacy concerns. This paper investigates a crucial but unexplored question: *What is the risk of membership information leakage in code models?* Membership leakage refers to the vulnerability where an attacker can infer whether a specific data point was part of the training dataset. We present GOTCHA, a novel membership inference attack method designed for code models, and evaluate its effectiveness on Java-based datasets. GOTCHA simultaneously considers three key factors: model input, model output, and ground truth. Our ablation study confirms that each factor significantly enhances attack performance. Our ablation study confirms that each factor significantly enhances attack performance. Our investigation reveals a troubling finding: membership leakage risk is significantly elevated. While previous methods had accuracy close to random guessing, GOTCHA achieves high precision, with a true positive rate of 0.95 and a low false positive rate of 0.10. We also demonstrate that the attacker's knowledge of the victim model (e.g., model architecture and pre-training data) affects attack success. Additionally, modifying decoding strategies can help reduce membership leakage risks. This research highlights the urgent need to better understand the privacy vulnerabilities of code models and develop strong countermeasures against these threats.**

***Index Terms*—Membership inference attack, privacy, large language models for code, code completion.**

Received 10 March 2024; revised 27 September 2024; accepted 6 October 2024. Date of publication 25 October 2024; date of current version 12 December 2024. This work was supported by the National Research Foundation, under its Investigatorship Grant (NRF-NRFI08-2022-0002). Recommended for acceptance by L. Pasquale. *(Corresponding author: Zhou Yang.)*

Zhou Yang, Chenyu Wang, Jieke Shi, and David Lo are with the School of Computing and Information Systems, Singapore Management University, Singapore 188065 (e-mail: zyang@smu.edu.sg; chenyuwang@smu.edu.sg; jiekeshi@smu.edu.sg; davidlo@smu.edu.sg).

Zhipeng Zhao is with the University of Copenhagen, DK-1165 Copenhagen, Denmark (e-mail: zhipeng.zhao@di.ku.dk).

Dongsun Kim is with Korea University, Seoul 02841, South Korea (e-mail: darkrsw@korea.ac.kr).

DongGyun Han is with Royal Holloway, University of London, TW20 0EX Egham, U.K. (e-mail: DongGyun.Han@rhul.ac.uk).

Digital Object Identifier 10.1109/TSE.2024.3482719

I. INTRODUCTION

RECENT years have witnessed a surging trend in training large language models [1], [2], [3], [4] on source code [5], [6] to produce *code models* for a wide range of critical software engineering tasks [7], including code completion [8], software contents summarization [9], [10] and defect prediction [11]. For example, GitHub Copilot uses the OpenAI Codex [12] model that is trained on billions of lines of code to assist developers in writing code. Similar tools include IntelliCode,[1] and CodeWhisperer,[2] which have been integrated into popular IDEs such as Visual Studio Code.

Although code models have achieved noticeable success in both academic and industrial settings, a series of studies expose that these models are vulnerable to various attacks, including adversarial attacks [13], [14], [15], [16], [17], data poisoning attacks [18], [19], [20], and privacy leakage [21], [22]. The vulnerabilities may introduce new defects, which are often difficult to discover and fix. Thus, it is necessary to clearly assess the potential risks of using code models.

This study explores a critical, previously overlooked aspect of code models: their vulnerability to *membership inference attacks* (MIA). MIA aims to determine whether a specific data instance was included in the training dataset of a *victim model*—the model subjected to the attack. Users and developers of code models can be interested in discerning such membership information for various purposes. For example, users may avoid a model if they discover it was trained on low-quality data, such as code containing smells or non-standard practices [23]. Membership information can also be used to detect unauthorized training, i.e., whether the model is trained on the code that is not authorized for training [24], to protect the intellectual property of the code. Attackers, however, can exploit MIA to uncover further vulnerabilities. For instance, recent research on backdoor attacks [19], [25] shows that if malicious code is part of the training data, the attacker could use MIA to identify this and trigger backdoor vulnerabilities to compromise the model.

Additionally, the severity of MIA is further magnified due to the diversity of the training data sources for code models. These sources include public domains, like open-source projects on GitHub, as well as private repositories with confidential corporate code. For example, Amazon CodeWhisperer, an automatic

[1]https://visualstudio.microsoft.com/services/intellicode/
[2]https://aws.amazon.com/codewhisperer/

0098-5589 © 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Korea University. Downloaded on April 03,2025 at 02:59:56 UTC from IEEE Xplore. Restrictions apply.

code completion tool, is trained on both open-source and proprietary datasets [26], which may include sensitive data such as API keys and personal information [27]. Yang et al. [21] show that code models can generate API keys, and attackers could use MIA to infer whether such keys are part of the training data, potentially exposing sensitive information from companies.

Growing concerns about code models drive us to explore an essential yet underexamined question: *what is the risk of membership information leakage in code models?* To answer this, we introduce GOTCHA, a novel membership inference attack method for code completion models. First, a *surrogate model* is trained to mimic the victim model's behavior. Then, the surrogate is provided with inputs from both training and non-training data to generate outputs. Using these outputs, we train a classifier to distinguish between training and non-training data. This classifier encodes three types of information—model input, model output, and ground truth—into embeddings. Our ablation study confirms that each type of information contributes to classifier performance.

Our research focuses on highlighting the membership leakage risks posed by GOTCHA. As a demonstrative measure, our experimental framework is centered around CodeGPT [5], an open-source model that demonstrates competitive performance on the code completion task. We additionally evaluate the generalizability of GOTCHA on five open-source models, including `CodeGen` [28], `CodeParrot` [29], `gpt-neo` [30], `PolyCoder-160M`, and `PolyCoder-0.4B` [31]. We finetune these models on the JavaCorpus dataset [32] to obtain the victim models.

We consider two baseline MIA methods. The first baseline, proposed by Hisamoto et al. [33], involves training classifiers such as nearest neighbors and decision trees. These classifiers utilize statistical features of model outputs as their inputs. The second baseline [34] adopts a ranking mechanism, utilizing language-centric metrics such as perplexity. In this approach, data instances at higher rankings are deemed more likely to be members of the training data. Our experiments have been executed across a spectrum of configurations, which encompasses variations in hyper-parameters such as the number of training epochs, the selection of surrogate models, etc.

We conduct our experiment on a Java dataset. The experiment results show that the proposed method achieves the best performance in terms of both the attacker's *power* (i.e., the true positive rate) and the attacker's *error* (i.e., the false positive rate). Utilizing CodeGPT as the surrogate model, GOTCHA demonstrates a power value of 0.95, substantially surpassing the two baseline methods, which approximate the randomness of guessing. We unveil a concerning fact: **the risk associated with the leakage of membership information is elevated**.

Further exploration reveals that the extent of the attacker's knowledge of the victim model affects the membership information leakage risks. To be more specific, the attacker can infer membership with a higher accuracy if the attacker can access a larger portion of the victim model's training data, suggesting that the model developers should include more training data that is inaccessible to the attacker. It also favors the attacker if the attacker uses a surrogate model that shares the same architecture as the victim model. For example, using the CodeGPT as the surrogate model can achieve a higher power value than using the 12-layer Transformer or LSTM as the surrogate model. As a result, to protect code models from MIA, the model developers should try to conceal the details of the victim model's architecture.

We further investigate how the decoding strategy affects the attacking results. By default, CodeGPT uses beam-search [35] to generate the outputs. We find that using a different decoding strategy (i.e., top-k sampling) can mitigate the risk of MIA. This paper calls for attention to the privacy concerns on code models and developing effective defense strategies against such attacks. The contributions of this paper include:

- **MIA threats in code models:** We are the first to investigate the risks of membership information leakage when using code models. We propose GOTCHA, an effective membership inference attack method for code models to investigate such risks. We evaluate the proposed method on the CodeGPT model, demonstrating that there exists a high risk of membership information leakage.
- **Risk assessment of code models:** The attacker's knowledge of the victim model affects the risk of membership information leakage. Knowing the victim model's architecture and accessing a larger portion of the training data can increase the risk. We also find that using a different decoding strategy (i.e., changing from beam-search to top-k sampling) can mitigate the risk.
- **Replication Package:** To facilitate further studies in evaluating the such risks and developing an effective defense against such attacks, we make our code and data publicly available at https://github.com/yangzhou6666/MIA-LLM4Code

Paper Structure. This paper unfolds in a structured manner as delineated hereafter. Section II describes the background of this study, including the code models and motivation for studying privacy attacks. In Section III, we explain our proposed method. Section IV presents the experiment settings. Section V evaluates the risks by answering research questions. We further discuss potential defensive strategies and threats to validity in Section VI. Section VII introduces relevant works. Finally, we conclude the paper and provide the information regarding the replication package in Section VIII.

II. BACKGROUND

A. Code Models

The success of large language models in natural language processing, such as BERT [1], RoBERTa [2], and T5 [3], has spurred the development of code-specific models like CodeBERT [36], GraphCodeBERT [37], and CodeT5 [38]. Trained on extensive, publicly available source code datasets [6], [32], [39], these models have achieved state-of-the-art results in various software engineering tasks, including code completion [8], program repair [40], and defect prediction [11].

In general, the input to the code model consists of a sequence of tokens, denoted by $x_1, \cdots, x_i$. The model generates a probability distribution, $f_\theta(y_1|x_1, \cdots, x_i)$, which represents

Authorized licensed use limited to: Korea University. Downloaded on April 03,2025 at 02:59:56 UTC from IEEE Xplore. Restrictions apply.

the likelihood of the next token in the sequence being y_1. In particular, CodeGPT uses *beam-search* [35] to generate the next token. The beam-search algorithm selects the top k most probable tokens as the initial beams, where k is the beam size. Then, the algorithm expands each of the k beams by considering all possible next tokens y_2, given the partial sequence y_1, y_2 and the input sequence $x_1, \cdots, x_i$. The model calculates the joint probability of the new sequences as:

$$p(y_1, y_2|x_1, \cdots, x_i) = p(y_1|x_1, \cdots, x_i) \times p(y_2|x_1, \cdots, x_i, y_1)$$

This iterative process perseveres until the attainment of a pre-established maximum target sequence length. Upon fulfillment of the stopping criterion, the algorithm discerningly selects the beam with the highest overall probability as the final generated target sequence.

It is important to note that our study does not directly engage with state-of-the-art models like OpenAI's Codex or ChatGPT due to the inaccessibility of their training data and potential legal issues related to testing these commercial systems. Instead, we focus on CodeGPT [5], a widely used code model with publicly available training datasets.

B. Motivation

Membership Inference Attack (MIA) poses a significant privacy risk by determining if a specific data point was used in training a Deep Neural Network (DNN) model [41]. Evaluating the vulnerability of code models to such attacks is essential for protecting the information of models and their training data. This section outlines the motivation for studying MIA on code models.

Motivation 1: MIA may bring threats of privacy leakage. It is imperative to acknowledge privacy as a pivotal non-functional requirement from a developer's viewpoint in the developmental process of code models. Code models are trained using a variety of data sources, ranging from publicly available data, such as open-source projects on GitHub, to more private and confidential data from companies. This training data can encompass sensitive elements such as passwords, critical software implementation logic, and API keys [27]. Research indicates that language models have the potential to memorize and inadvertently reveal parts of their training data, including sensitive information [21], [34], [42]. A malicious actor, through the utilization of MIA, can potentially ascertain whether a code model has been trained on datasets containing sensitive or confidential information, thus enabling further exploitative attacks aimed at data theft or sensitive information extraction.

Motivation 2: MIA may bring security threats. If a model's training data includes code with known or unknown security vulnerabilities [43], the model may propagate such vulnerabilities to the systems that use the code generation models during their development. In this case, attackers may leverage MIA to identify what vulnerabilities are potentially included in the code models, and then launch further attacks on the systems that use the code models, putting these systems at risk.

Motivation 3: A further motivation for assessing MIA in code models lies in its capability to safeguard intellectual property. Studies have indicated that open-source developers might not explicitly provide consent to data collectors for model training using their code [44], a practice termed as *unauthorized training* [24]. Additionally, certain open-source codes possess licenses that preclude their utilization in model training. MIA can serve as a tool to ascertain the inclusion of such protected codes in the training process.

C. Threat Model

A threat model encompasses the assumptions regarding the positions and capabilities of both the attacker and the defender, along with a detailed description of the attack process. In this study, we adopt the following assumptions to constitute our threat model.

Assumption 1 (Model Usage): We assume that the users of code models are afforded *black-box* access, allowing them to interact with the models multiple times to accumulate pairs of inputs and outputs.

Assumption 2 (Model Parameters): We postulate that the attacker is restricted from accessing the model's parameters or gradient information, aligning with real-world scenarios where model owners typically offer their models as services accessible via APIs. This service-oriented access restricts users to querying the model without direct exposure to underlying parameters or gradients. For instance, OpenAI facilitates API access to its code completion services, aligning with this assumption.[3] Such an assumption is consistent with the premises adopted in previous works evaluating threats against code models [13], [19], [45].

Assumption 3 (Training Data): Following the baselines [33], we also assume that users may have access to portions of the models' training data. Prevalent powerful code models are predominantly trained utilizing extensive open-source datasets. For instance, Copilot [12] undergoes training with natural language text and source code extracted from publicly accessible sources, such as code housed in public repositories on GitHub. Similarly, CodeWhisperer enhances its performance through training on a substantial volume of publicly available code [26]. The training data employed by well-known open-source code models, for example, CodeSearchNet [6], is also publicly accessible. However, model owners might also employ their *private* data for training purposes, keeping it inaccessible to users. This makes it plausible to assume that users can access only certain segments of the models' training data.

III. Methodology

This section explains our methodology. The overview of our method is shown in Fig. 1. We first formulate the task and explain the design of our proposed approach Gotcha.

A. Task Formulation

Existing membership inference attack (MIA) methods have exhibited satisfactory performance on classification models

[3]https://platform.openai.com/docs/introduction

Authorized licensed use limited to: Korea University. Downloaded on April 03,2025 at 02:59:56 UTC from IEEE Xplore. Restrictions apply.

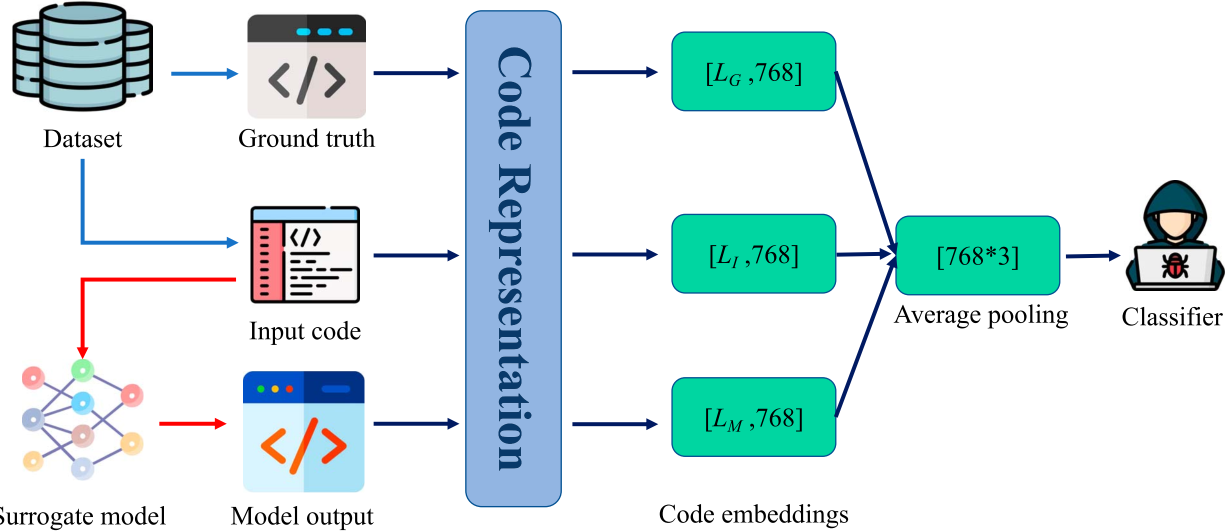

Fig. 1. The overview of the proposed method. To achieve a high attack performance, our method coverts three types of information (including the ground truth, model input, and model output) into code embeddings and build a classifier on top of the embeddings.

[41]. However, applying these methods to code completion problems presents a markedly greater challenge. Code completion can be conceptualized as a sequence of interconnected classifications, which substantially amplifies the complexity of addressing the MIA problem.

We formally define the MIA on code completion models as follows. Let us consider a code completion model $\mathcal{M}$ and it is fine-tuned on a dataset $\mathcal{D}_{in}$. The training dataset $\mathcal{D}_{in} = \{(x_i, y_i)\}_{i=1}^{n}$, where x_i is the input, y_i is the output, and n is the size of the training set. Following existing studies on MIA and other threats to code models [13], [16], [19], we assume that the model $\mathcal{M}$ is *static*, i.e., the model does not change over time when the MIA is conducted. The model $\mathcal{M}$ can be queried and complete code in a black-box manner: $\hat{y} = \mathcal{M}(x)$; $\hat{y}$ is the output of the model $\mathcal{M}$ given the input x. The attacker aims to build a binary classifier $\mathcal{G}$ to infer whether an example (x, y) is a member of the training set $\mathcal{D}_{in}$. The goal of this classifier is:

$$\mathcal{G}(x, y, \hat{y}) = \begin{cases} 1 & \text{if } (x, y) \in \mathcal{D}_{in}; \\ 0 & \text{otherwise.} \end{cases}$$

B. Training Surrogate Models

Our proposed methodology, GOTCHA, operates in two steps. Initially, we train a *surrogate model* using part of the training data as the target victim model, but without direct interaction or access to the victim model's outputs. Subsequently, in the second phase, the surrogate model undergoes queries utilizing its training data, as well as previously unseen non-training data. The outcomes of these queries facilitate the training of a membership inference classifier, tasked with deducing the membership within the surrogate model's training set.

Fig. 2 shows how we split the dataset to train and evaluate the proposed method. Let $\mathcal{D}_{in}$ be the training set of the victim model. As explained in the threat model in Section II-C, we make a practical assumption that the attacker can access a part

$\mathcal{D}_{in}$: Training data of the victim model	
$\mathcal{D}_{in}^{-*}$: Inaccessible to attackers	$\mathcal{D}_{in}^{*}$: Data known to attackers

$\mathcal{D}_{out}$: Non-training data of the victim model	
$\mathcal{D}_{out}^{-*}$: Data for evaluating MIA classifiers	$\mathcal{D}_{out}^{*}$: Data for training MIA classifiers

Fig. 2. Splitting the datasets to train and evaluate surrogate models as well as the membership inference classifiers.

of the training data, which is denoted by $\mathcal{D}_{in}^{*}$. The remaining training data is inaccessible to the attacker, which is denoted by $\mathcal{D}_{in}^{-*}$. The attacker uses $\mathcal{D}_{in}^{*}$ to train a surrogate model $\mathcal{S}$.

$\mathcal{D}_{in}^{*}$ is also then used as the positive examples (i.e., as they are used to train the surrogate model) to train the membership classifiers. But training a binary classifier requires both positive and negative examples. So the attacker then finds another dataset $\mathcal{D}_{out}^{*}$ that are not used to train the surrogate model and the victim model. It is a common practice to balance the training data (i.e., the ratio of positive and negative examples is $1:1$) to train a binary classifier, so we set $\mathcal{D}_{in}^{*}$ and $\mathcal{D}_{out}^{*}$ to be the same size. For each example (x, y) in $\mathcal{D}_{in}^{*} \cup \mathcal{D}_{out}^{*}$, the attacker queries the surrogate model and obtains the corresponding output $\hat{y}_{\mathcal{S}} = \mathcal{S}(x)$. The attacker creates a new dataset $\mathcal{D}_{MIA}$. The input to the membership classifier is a tuple $\langle x, y, \hat{y}_{\mathcal{S}} \rangle$, and the output is the membership label. Formally, the label is decided as follows:

$$label = \begin{cases} 1 & \text{if } (x, y) \in \mathcal{D}_{in}^{*}; \\ 0 & \text{if } (x, y) \in \mathcal{D}_{out}^{*}. \end{cases}$$

Note that as shown in Fig. 2, the attacker can create another unseen dataset $\mathcal{D}_{out}^{-*}$ that shares the same size as $\mathcal{D}_{in}^{-*}$, to evaluate the MIA classifiers on the victim model.

In this paper, we consider multiple surrogate models to scrutinize the impact of the attacker's knowledge of the victim

Authorized licensed use limited to: Korea University. Downloaded on April 03,2025 at 02:59:56 UTC from IEEE Xplore. Restrictions apply.

model on the efficacy of the attack. The breadth of the attacker's knowledge concerning the victim model can be delineated across two dimensions: architectural comprehension and awareness of the pre-training data. We incorporate four distinctive surrogate models within our analysis: (1) LSTM, (2) Transformer, (3) GPT-2, and (4) CodeGPT.

Employing CodeGPT as the surrogate model epitomizes the most favorable scenario for the attacker, who is presumed to know both the architecture and the pre-training data of the victim model. Utilization of GPT-2 signifies a scenario where the attacker knows the victim model's architecture but lacks knowledge of the pre-training data. Implementing the Transformer model signifies that the attacker possesses partial awareness of the victim model's architectural design[4] but remains uninformed about the pre-training data. The employment of LSTM represents a condition where the attacker lacks knowledge pertaining to both the architecture and the pre-training data of the victim model.

C. Training MIA Classifiers

We train an MIA classifier, utilizing a dataset forged through the employment of the surrogate model. Hisamoto et al. [33], while conducting MIA on natural language models, extract statistical features, constructing a classifier upon these foundational elements, incorporating aspects such as 1- to 4-gram precision and smoothed sentence-level BLEU score. However, findings from prior research [33] indicate limited effectiveness of these methods when applied to MIA on completion models. In our exploration, we draw inspiration from the realization that pre-trained code models (e.g., CodeBERT [36]) exhibit a commendable performance in classifying code inputs across varied categories, such as vulnerability assessments and code clone detection. Motivated by this, we employ CodeBERT for the extraction of code embeddings, proceeding thereafter to train a classifier using these refined embeddings.

1) Architecture Design and Input Processing: By querying the surrogate model, we obtain its output that corresponds to the input. A series of empirical studies [46], [47] have shown the effectiveness of pre-trained language models in code classification tasks, e.g., clone detection and defect prediction. In this study, we use CodeBERT [36], which is one of the most widely used pre-trained code models. To be more specific, we use three CodeBERT models to extract three code embeddings from the input, model output, and ground truth, respectively. CodeBERT treats its input as a sequence of tokens. Assuming that the sequence length is L, CodeBERT produces an embedding matrix of size $L \times 768$.

Typically, a classification model expects its input size to be fixed or it performs some processing to make the input of the same size. In this study, we use the *average pooling* method to compress the embedding metrics into a single embedding of size 768. The average pooling method is also used in previous studies [48], [49]. We conduct the same processing on the embeddings for the model output and the ground truth to obtain two 768-dimensional vectors. The processed embeddings are then concatenated together to form the final input. We use a two-layer fully connected network with 768 hidden units and the `tanh` activation function as the MIA classifier.

2) Model Training and Inference: The proposed approach, GOTCHA, consists of the CodeBERT models and the MIA classifier; we denote their parameters with θ_p and θ_c, respectively. Given an example $\langle x, y, \hat{y_\mathcal{S}} \rangle$, we denote its label for the classification task as $l \in \{0, 1\}$, where 1 means the example is in the training set of the victim model and 0 otherwise. $\mathcal{G}(\langle x_i, y_i, \hat{y_\mathcal{S}} \rangle)$ represents the prediction results of the MIA classifier. We train the MIA classifier by optimizing the following loss function:

$$\arg\min_{\theta_p, \theta_f} \sum_{x_i \in D^*} \log(\mathcal{G}(\langle x_i, y_i, \hat{y}_i \rangle))$$

In the above equation, we train the attacker using a balanced dataset $D^* = \mathcal{D}_{in}^* \cup \mathcal{D}_{out}^*$, where $\mathcal{D}_{in}^*$ contains examples in the training data of the victim model and $\mathcal{D}_{out}^*$ contains unseen examples of the victim model. To optimize the above loss function, we use back-propagation to update both parameters θ_p and θ_c.

When training the MIA classifier, we use the output from the surrogate model. However, in the inference stage (i.e., when an attack is performed), we use the output from the victim model to evaluate the attack performance. We denote the output from the victim model as $\hat{y_v}$. We send $\langle x, y, \hat{y_v} \rangle$ to the MIA classifier to obtain the prediction results $\mathcal{G}(\langle x, y, \hat{y_v} \rangle)$. Note that training a DNN-based MIA classifier has a randomness that may affect the attack performance. To mitigate the threats due to the randomness, we train the MIA classifier for three times using different random seeds to initialize the model parameters. In our experiment, we report the average performance of the three trained MIA classifiers.

D. Evaluation Metrics

Aligned with preceding research [33], we assess the performance of MIA classifiers employing specific metrics: the attacker's power, quantified by the True Positive Rate (TPR), and the attacker's error, measured by the False Positive Rate (FPR). Supplementing these metrics, we also incorporate the Area Under the Receiver Operating Characteristic (ROC) Curve (AUC-ROC).

1) True Positive Rate (TPR): The True Positive Rate (TPR) represents the attacker's proficiency in accurately identifying instances that genuinely belong to the training dataset. True positive rate can be considered as 'attacker's power.' A heightened TPR means that the attacker is strong in pinpointing these instances, thereby posing a potential risk to the confidentiality of the training data.

2) False Positive Rate (FPR): The False Positive Rate (FPR) delineates the frequency at which the attacker erroneously categorizes instances as belonging to the training dataset when they do not. False positive rate can be considered as 'attacker's error.' An elevated FPR suggests a lack of precision in the attacker's identifications, resulting in numerous false alarms and a diminished threat to the privacy of the training data.

[4]CodeGPT is a Transformer-based model.

Authorized licensed use limited to: Korea University. Downloaded on April 03,2025 at 02:59:56 UTC from IEEE Xplore. Restrictions apply.

3) Area Under the ROC Curve (AUC): The Area Under the ROC Curve (AUC) manifests as a singular numeric value extracted from the ROC curve, encapsulating the overall performance of the attacker. An AUC value proximate to 1 unveils a highly competent attacker, whereas a value nearing 0.5 implies that the attacker's performance is akin to random guess. Superior AUC values signify enhanced performance by the attacker, while inferior values indicate subpar execution.

IV. Experiment Settings

This section describes the configurations of our experiments, encompassing aspects such as the victim model, datasets, baseline methodologies, and implementation particulars.

A. Victim Model

Our study does not directly involve state-of-the-art models like OpenAI's Codex or ChatGPT. We do not pick them as what data is used to train these models is unknown, making it infeasible to correctly evaluate the membership leakage risk. Moreover, our goal is to evaluate the membership leakage risk, not to create an attack that can be operationalized by real adversaries to commercial services, which may raise potential legal issues.

As a result, we choose CodeGPT, a popular open-source code completion model included in the CodeXGLUE benchmark, as our main experiment subject to gain a deeper understanding of data privacy issues of code models. CodeGPT consists of 12 layers of Transformer decoders, sharing the same model architecture and training objective of GPT-2 [50]. Different versions of CodeGPT models are released on the HuggingFace platform.[5] We use the '`microsoft/CodeGPT-small-java`' model, which is pre-trained on the Java code (around 1.6 million Java methods) in the CodeSearchNet dataset [6]. This model is pre-trained with randomly initialized model parameters. Then, following the practice adopted in CodeXGLUE benchmark [5], we further fine-tune CodeGPT-small-java on a subset of 1% randomly sampled examples from JavaCorpus to obtain the victim model. Note that JavaCorpus and CodeSearchNet are two different datasets and JavaCorpus is not included in the "pre-training dataset" of CodeGPT-small-java.

Additionally, to further evaluate the generalizability of our proposed membership inference attack, we consider five open-source models: `CodeGen` [28], `CodeParrot` [29], `gpt-neo` [30], `PolyCoder-160M`, and `PolyCoder-0.4B` [31]. These models are widely used as experiment subjects in recent studies [21], [31], [51]. `CodeGen` models [28] adopt a standard transformer decoder with left-to-right causal masking. We choose the `CodeGen-multi-350M` model, which is pre-trained on the BigQuery dataset and can support Java code completion. `CodeParrot` [29] adopts the GPT-2 architecture with 1.5 billion parameters. We choose two variants of PolyCoder models: `PolyCoder-160M` and `PolyCoder-0.4B`, with 160M and 0.4B parameters, respectively. For `gpt-neo` [30], we use its 125M parameter version.

[5]https://huggingface.co/microsoft/CodeGPT-small-java

TABLE I
Statistics of Dataset for Training and Evaluation Models. The Training Set of Surrogate Models Is Randomly Sampled From the Training Set of the Victim Model, Which Is Also Used as the Positive Examples to Train MIA Classifiers

Model	Data Size	
	Training	Testing
Victim Model	12,934	8,268
Surrogate Model	1,293	8,268
MIA Classifiers	2,586	2,586

TABLE II
Model Names Used in the Study and Their Corresponding Names on the HuggingFace Platform

Model	Model Name on HuggingFace
`CodeGPT`	`microsoft/CodeGPT-small-java`
`CodeGen`	`Salesforce/codegen-350M-multi`
`CodeParrot`	`codeparrot/codeparrot-small`
`PolyCoder-160M`	`NinedayWang/PolyCoder-160M`
`PolyCoder-0.4B`	`NinedayWang/PolyCoder-0.4B`
`gpt-neo`	`EleutherAI/gpt-neo-125m`

B. Datasets

In this study, we use datasets for code completion as it is one of the most important tasks in software engineering. Given a piece of code snippet, the goal of code completion is to predict the next tokens or lines. We consider a popular dataset included in the CodeXGLUE benchmark [5]: JavaCorpus [32]. Allamanis and Sutton collect the JavaCorpus dataset [32], containing over 14,000 Java projects from GitHub. The CodeXGLUE benchmark follows the settings in Karampatsis et al.'s study [52] and samples 1% of the subset from the JavaCorpus dataset, ending up with 12,934/7,189/8,268 files for the training/validation/test set, respectively. Then, the CodeXGLUE benchmark preprocesses the dataset by tokenizing the source code using a Java parser and removes all the comments. As reported in the paper [5], strings that are longer than 15 characters are replaced with empty strings.

We further explain how we split the dataset to train and evaluate the MIA classifiers. For victim models, their training and testing dataset have 12,934 and 8,268 examples, respectively. We randomly sample some examples (for example 10%, i.e., 1,293) from the training set; the sampled examples will be used to train the surrogate model and then used as the positive examples (i.e., ground truth label is 1) to train MIA classifiers. As training MIA classifiers also requires negative examples whose ground truth labels are 0, we randomly sample the same number of examples from victim models' testing set as the negative examples to train MIA classifiers. Similarly, we need to construct positive and negative examples to evaluate MIA classifiers. We randomly sample 1,293 examples (10% of the whole training set) from both the remaining training set and the remaining testing set. This strategy ensures that there is

Authorized licensed use limited to: Korea University. Downloaded on April 03,2025 at 02:59:56 UTC from IEEE Xplore. Restrictions apply.

no overlap between the training and evaluation data of MIA classifiers.

C. Baselines

This study utilizes the techniques used for evaluating natural language models. Although MIA is an important threat to AI and has been studied since 2017, researchers mainly focus on classification tasks and there are only a few studies on attacking generative language models. While these baselines are not specifically designed for code models, there is a lack of membership inference attacks tailored for code completion tasks. Therefore, we adapt existing approaches by Hisamoto et al. [33] and Carlini et al. [34], originally designed for natural language models, to serve as baselines for our investigation into membership information leakage risk in code models. These two lines of methods are categorized as feature-based Classification and metrics-based ranking methods. Our proposed method is different from their methods. Instead of computing manually defined features from examples, we infer membership by considering the input example, ground truth, and how models react (i.e., completion) on the example. Specifically, we design a novel membership inference classifier that considers the three types of information together and show that each information matters for the final prediction.

1) Feature-Based Classification: Our first baseline is a set of classifiers trained by features. Hisamoto et al. [33] develop MIA on a machine translation system. They extract some features from the model output and the ground truth to build a binary classifier. The considered features are modified 1- to 4-gram precision and smoothed sentence-level BLEU score [53]. The intuition is that if an unusually large number of n-grams in y matches $\hat{y}$, then it could be a sign that this is in the training data and the victim model memorizes it. Hisamoto et al. try different types of classifiers. Following their settings, our study uses Nearest Neighbors (NN), Decision Tree (DT), Naive Bayes (NB), and Multi-layer Perceptron (MLP). Additionally, we consider deep neural networks (DNN).

2) Metrics-Based Ranking: Second, we utilize ranking techniques based on certain metrics as another baseline. Carlini et al. [34] investigate the data extraction attack on language models. Specifically, they propose a process that involves sampling numerous examples from a language model and subsequently ranking them using specific metrics. The objective is to rank examples from the training dataset in the top positions, which closely aligns with the goal of MIA. We refer to this research approach as *metrics-based ranking* for MIA. After ranking the examples using different metrics, the attacker needs to set a *cut-off* position to determine what examples will be considered as in the training set. In Carlini et al.'s work [34], the cut-off position is the top 10% of the examples. Our study uses a balanced dataset to evaluate MIA, so we set the cut-off position at 50%. By doing so, we can assign the predicted labels to each example and compute the evaluation metrics like power, error, and AUC. Some metrics in Carlini et al.'s work is designed for natural language, e.g., converting all the characters to lowercase, which is not suitable for code models. So this paper considers the following metrics to infer data membership in code models.

Perplexity. Perplexity [54] is a measurement of how well a probability model predicts a sample. The logarithm of perplexity is the formally defined as $log(P) = -\frac{1}{N}\sum_{i=1}^{N}\log P(w_i \mid w_1, w_2, \ldots, w_{i-1})$, where N is the total number of words in a test example, w_i is the i-th word, and $P(w_i \mid w_1, w_2, \ldots, w_{i-1})$ is the conditional probability of the i-th word given the previous words in the sequence. A low perplexity signifies that the model is good at predicting the sample. Intuitively, a low perplexity for a specific example can suggest that the model has previously encountered this example during training. We use the victim model to compute the perplexity of each example and rank them in ascending order.

Comparing perplexity of another language model. As described by Carlini et al. [34], this metric is computed by the ratio of log-perplexities of the victim model and another language model. In this paper, we use the surrogate model as the second model. So the metric is computed as $\frac{log(P_v)}{log(P_s)}$, where P_v is the perplexity computed using the victim model and P_s is the perplexity computed using the surrogate model. We rank examples in descending order.

Comparing to `zlib` compression. Carlini et al. [34] also consider the `zlib` compression. When using `zlib` [55] to compress a sequence of tokens, `zlib` represents the compressed sequence using as few bits as possible. The `zlib` entropy of a sequence is defined as the number of bits used to represent the compressed sequence. The attacker uses $\frac{log(P_v)}{zlib}$, i.e., the ratio of the victim model perplexity and the `zlib` entropy as a membership inference metric.

D. Implementation and Experiment Platforms

We utilize the replication package provided in the CodeXGLUE benchmark [5] to fine-tune CodeGPT. However, as Hisamoto et al.'s paper [33] does not provide a replication package, we follow the methodology and guidelines described in their paper to implement the MIA classifiers based on the statistical features. To implement the decision tree, we use GINI impurity for the splitting metrics and the max depth is set as 5. Naive Bayes uses Gaussian distribution. We set the number of neighbors to 5 and use Minkowski distance to implement the NN classifier. For MLP, we set the size of the hidden layer to be 50, the activation function to be ReLU, and the L_2 regularization term α to be 0.0001. The hyperparameters settings follow the settings of Hisamoto et al.'s study [33]. To evaluate the effectiveness of metrics-based ranking methods, we leverage the replication package provided by Carlini et al. [34].[6]

We perform our experiments on a computer running Ubuntu 18.04 with 4 NVIDIA GeForce A5000 GPUs. To mitigate the effect of randomness in training MIA classifiers, we repeat each experiment using three different random seeds for model parameter initialization in each run. We compute the average

[6]https://github.com/ftramer/LM_Memorization

Authorized licensed use limited to: Korea University. Downloaded on April 03,2025 at 02:59:56 UTC from IEEE Xplore. Restrictions apply.

results for each evaluation metric, which enable us to provide a more accurate and reliable representation of the MIA classifier's performance, which is less susceptible to the influence of random factors.

V. Results

In this section, we evaluate the risk of membership leakage in code models by answering the following three research questions (RQs):

- **RQ1.** *To what extent are code models vulnerable to membership inference attacks?*
- **RQ2.** *What are the factors affecting the membership leakage risk?*
- **RQ3.** *What are the features of the training examples whose memberships are more likely to be correctly inferred?*

In the first RQ, we apply our proposed approach and two baselines [33], [34] to CodeGPT to evaluate the risks exposed by these attacks. Then, in the second RQ, we conduct the attack in different settings to simulate the different prior knowledge the attackers (e.g., model architecture, size of known training data, etc.) have and analyze the factors that affect the membership leakage risk in code models. Lastly, we investigate the features of the training examples whose membership is more likely to be correctly inferred.

RQ1. To What Extent Are Code Models Vulnerable to Membership Inference Attacks?

In this question, we evaluate our proposed approach GOTCHA and baseline attacks [33], [34] on CodeGPT model [5]. As mentioned in the threat model, it is reasonable to assume that the attacker can access part of the training data of the victim model. In this RQ, we assume that the attacker knows 20% of the training data of the victim model, which is used to train the surrogate model and the MIA classifier to infer the data membership. In this experiment, the surrogate model is a pretrained CodeGPT model and then fine-tuned on part of the victim model's training data that are known to the attacker. In the following RQ, we will try different surrogate models to analyze the impact of the surrogate model architecture on the risk of privacy attacks.

We consider two branches of baseline MIA methods: classification-based and metrics-based attacks. Our proposed method GOTCHA and the work by Carlini et al. [34] are classification-based, which use a classifier to infer the data membership. For classification-based attacks, we compute the performance metrics of the corresponding classifiers, including the accuracy, precision, recall, F1 score, and AUC. Carlini et al. [34] try different metrics to infer the data membership. More specifically, the data that are more likely to be the training data will be ranked in a higher position.

The experiment results are listed in Table III, which presents the attacker's power (true positive rate), attacker's error (false positive rate), and AUC scores for different attacks of different types of attacks used in the experiment. The table shows that GOTCHA attack has the highest power score of 0.95, indicating

TABLE III
THE PERFORMANCE OF DIFFERENT MEMBERSHIP INFERENCE ATTACKS ON THE CODEGPT MODEL. *w.o.* MEANS 'WITHOUT', I.E., WE EXCLUDE A CERTAIN PART FROM THE PROPOSED METHOD. H-ATTACK AND C-ATTACK REFER TO THE WORKS BY HISAMOTO ET AL. [33] AND CARLINI ET AL. [34]

	Variants	Power	Error	AUC
Ours	GOTCHA	**0.95**	**0.10**	**0.98**
	GOTCHA *w.o.* input	0.70	0.50	0.63
	GOTCHA *w.o.* truth	0.87	0.38	0.83
	GOTCHA *w.o.* output	0.65	0.27	0.60
H-Attack	Naive Bayes	0.23	0.17	0.58
	Decision Tree	0.30	0.25	0.57
	Nearest Neighbor	0.23	0.25	0.49
	Multi-layer Perceptron	0.28	0.22	0.58
	Deep Neural Network	0.21	0.27	0.58
C-Attack	Perplexity	0.58	0.42	0.58
	Compare Perplexity	0.47	0.53	0.47
	Compare `zlib`	0.55	0.45	0.55

that it is the most effective in identifying members in the training dataset. The error score for GOTCHA is 0.10, indicating that it incorrectly identified some non-members as members. The AUC score for GOTCHA is 0.98, indicating that the proposed approach is very effective.

The proposed method takes code embeddings of three parts: the input to the victim model, the ground truth, and the output from the victim model. We conduct an ablation study to analyze the benefits of each part. Table III shows the results when we remove one part from GOTCHA's input. For example, the row '*w.o.* truth' shows the results when we remove the ground truth from the input. The results show that all the three parts contribute to the effectiveness of the proposed method. Excluding the input, ground truth, and model output will reduce the AUC scores by 0.35, 0.15, and 0.38, respectively. It suggests that the model output is the most important part contributing to the effectiveness of the proposed method, followed by the input and the ground truth.

We also analyze the effectiveness of five classification-based attacks used by Hisamoto et al. [33]: Decision Tree, Naive Bayes, Nearest Neighbor, Multi-Layer Perceptron, and Deep Neural Network. Among them, the Decision Tree attack has the highest power score of 30.40, while Naive Bayes had the lowest power score of 22.98. However, all three classification-based attacks had relatively low AUC scores below 0.6, indicating that they are less effective than the GOTCHA attack at identifying membership. The AUC score of Nearest Neighbor is even lower, only 0.49. In Table III, the results for metric-based methods are obtained by setting the cut-off position as 50%. Under this setting, ranking using the three metrics (i.e., perplexity, comparing perplexity, and comparing `zlib`) achieves AUC scores of 0.58, 0.47, and 0.55, respectively. The results show that both two baselines are less effective than our proposed approach at identifying members in the training dataset.

We further evaluate the proposed method on five additional large language models of code: `CodeGen` [28], `CodeParrot` [29], `gpt-neo` [30], `PolyCoder-160M`, and `PolyCoder-0.4B` [31]. We train each victim model for 5 epochs and

Authorized licensed use limited to: Korea University. Downloaded on April 03,2025 at 02:59:56 UTC from IEEE Xplore. Restrictions apply.

apply the proposed method to each model. We use the best configurations of our method GOTCHA from RQ1, i.e., using the CodeGPT model as the surrogate model and the attacker knowing 20% of the training data. We also apply the baseline to these models. As the AUC score reflects the overall performance of a classifier, we report the AUC scores of the proposed method. We run membership inference attacks on each model using 5 different random seeds. We observe that the superior performance of GOTCHA can generalize to the five newly evaluated models. GOTCHA achieves AUC scores of 0.92, 0.95, 0.93, 0.94, and 0.94 on `CodeGen`, `CodeParrot`, `gpt-neo`, `PolyCoder-160M`, and `PolyCoder-0.4B`, respectively. In contrast, the AUC scores of both the feature-based classification and metric-based ranking baselines are lower than 0.60. The results show that GOTCHA achieves better performance than baselines on the five additionally evaluated models.

> **Answers to RQ1**: The evaluated victim model shows a **high risk** of leaking the training data membership information. The Gotchaattack is the most effective at identifying members, with an AUC score of 0.98, highlighting the need for better safeguards to mitigate this risk.

TABLE IV
ANOVA ANALYSIS OF WHAT FACTORS HAVE MORE IMPACT ON THE ATTACKER'S POWER. WE CONSIDER FOUR FACTORS: THE SURROGATE MODELS (*SURRO M*), THE VICTIM MODEL (*VICTIM M*), THE RATIO OF KNOWN TRAINING DATA (*RATIO*), AND THE TRAINING EPOCHS OF THE VICTIM MODEL (*EPOCH*). THE IMPACT IS MEASURED BY THE PROPORTION OF TOTAL VARIATION (SST %) IT CAN EXPLAIN. A HIGHER PERCENTAGE INDICATES A MORE SIGNIFICANT IMPACT

Factors	SST (%)	p-value
Victim M	0.10%	> 0.05
Surro M	**42.70%**	< 0.05
Ratio	**0.70%**	< 0.05
Epoch	0.04%	> 0.05
*Victim M * Surro M*	0.22%	> 0.05
*Victim M * Ratio*	0.10%	> 0.05
*Victim M * Epoch*	0.29%	> 0.05
Surro M * Ratio	**53.71%**	< 0.05
*Surro M * Epoch*	0.12%	> 0.05
*Ratio * Epoch*	0.06%	> 0.05
*Victim M * Surro M * Ratio*	0.24%	> 0.05
*Victim M * Surro M * Epoch*	0.66%	> 0.05
*Victim M * Ratio * Epoch*	0.28%	> 0.05
*Surro M * Ratio * Epoch*	0.11%	> 0.05

RQ2. What Are the Factors Affecting the Membership Leakage Risk?

The previous RQ demonstrates that code models are vulnerable to membership inference attacks. Here, we examine the factors influencing membership leakage risk. Since baseline methods are not effective in inferring data membership, we only evaluate our proposed method, GOTCHA. We focus on analyzing four key factors.

1) **The training epochs of the victim model**. Previous research [56] on classification models suggests that the risk of membership information leakage increases with more training epochs. We train each victim model for 5 epochs and apply GOTCHA to 5 variants of the model and observe the attacker's power.
2) **The surrogate models**. As explained in Section IV-A, we use surrogate models of different architectures to simulate the attacker's prior knowledge of the victim model. Intuitively, a surrogate model that can better mimic the victim model will be more effective in the attack. We choose four surrogate models: CodeGPT, GPT-2, 12-Layer Transformer, and LSTM.
3) **The victim models**. We evaluate the membership leakage risk of six open-source models: `CodeGPT`, `CodeGen`, `CodeParrot`, `gpt-neo`, `PolyCoder-160M`, and `PolyCoder-0.4B`.
4) **The ratio of training data that is known to the attacker**. If the attacker knows more data that is used to train the victim model, the attacker may train a better surrogate model and MIA classifier, which may lead to a higher attack success rate. We evaluate using two settings: 10% and 20% of the training data are known to the attacker.

To systematically evaluate the impact of these factors, we adopt the *Design of Experiments (DoE)* [57] method. The primary goal of DoE is to identify which factors (e.g., the choice of surrogate model in our study) most significantly affect the outcome of a process or system (e.g., the attacker's power). DoE has been widely used in various engineering fields, including software engineering. For example, Cotroneo et al. [58] apply DoE to understand the impact of different factors on data poisoning attacks for code models. Following their practice, we employ the *full factorial design*, which considers all possible combinations of factors. Specifically, we perform a total of 240 experiments (6 victim models $\times$ 5 epochs $\times$ 4 surrogate models $\times$ 2 ratios of known training data). We conduct an *Analysis of Variance (ANOVA)* [59] to assess each factor's impact on the attacker's power. We determine a factor's importance by looking at the percentage of total variation it accounts for, known as the portion of Sum of Squares Total (**SST %**). A factor is considered important if it explains a large part of the overall variation. Each factor's significance also comes with a p-value. A p-value lower than 0.05 indicates that the factor has a significant impact on the attacker's power.

Table IV shows the results of the ANOVA analysis. The factors that have a significant impact on the attacker's power are highlighted in gray. Notably, the surrogate model (*Surro M*) alone accounts for 42.70% of the total variation, highlighting its crucial role in influencing the attacker's power. Additionally, the interaction between the surrogate model and the ratio of known training data (*Surro M * Ratio*) explains an even larger portion of the variation at 53.71%, suggesting that the combination of these two factors is particularly influential. In contrast, other factors such as the victim model (*Victim M*) and training epochs (*Epoch*), as well as their interactions, show minimal impact with SST % values under 1% and p-values greater than 0.05, indicating they do not significantly affect the attacker's power.

Authorized licensed use limited to: Korea University. Downloaded on April 03,2025 at 02:59:56 UTC from IEEE Xplore. Restrictions apply.

Given the findings, we further conduct a series of statistical tests to validate the impact of each factor on the attacker's power in more detail.

The training epochs of the victim model. We train CodeGPT for 10 epochs and apply GOTCHA to 10 variants of the model, each trained for 1, 2, 3, ..., 10 epochs. We apply GOTCHA to each variant of the models and report the corresponding attacker's power. For a victim model trained for n epochs, we obtain 4 results as we use 4 different surrogate models (i.e., CodeGPT, GPT-2, Transformer, LSTM), denoted by A_n. For each epoch (e.g., i and j), we conduct a Wilcoxon signed-rank test [60] to compare the attacker's power A_i and A_j on the two models. We make the following null hypothesis:

There is no significant difference between the attacker's power on victim models trained for i and j epochs.

When multiple tests are performed, the probability of obtaining at least one false positive result increases [61]. Following previous studies, we apply the Bonferroni correction [62] to adjust the significance level for each individual test by dividing it by the number of tests being performed. Our results show that the p-value of tests on each pair is larger than 0.05, failing to reject the null hypothesis. Thus, the Wilcoxon test does not reveal any statistically significant differences between each pair of data sets.

We further compute the standard deviation of the MIA's precision and recall scores for the models trained for different epochs. Taking precision scores as an example, for each model trained for n epochs, we obtain n values of the precision scores. We then compute the standard deviation of these n values to measure the stability and consistency of the attack performance on models trained for various epochs. A small standard deviation indicates that the attack performance is consistent. A larger standard deviation indicates a greater difference in models' vulnerability to MIA under different training epochs settings. The standard deviation values of the precision and recall are both less than 0.0005, indicating that the number of training epochs of the victim model has little impact on the risk of privacy attacks. Nonetheless, it also cautions model developers that even though these state-of-the-art models are trained for just a few epochs, they remain susceptible to such attacks.

The surrogate models. We choose four surrogate models: CodeGPT, GPT-2, 12-Layer Transformer, and LSTM. Fig. 3 shows the attack performance (i.e., attacker's power, error, and AUC) when using different surrogate models. As can be seen in the table, using CodeGPT model can outperform the other surrogate models; it has the highest power of 0.95, the lowest error of 0.10, and the highest AUC of 0.98. This indicates that the CodeGPT model, which shares the same architecture and pre-training data with the victim model, has the strongest attack performance among the surrogate models tested.

Notably, we observe a decreasing trend in the AUC scores of the GPT-2, 12-Layer Transformer, and LSTM models. Although the LSTM model has a high power close to that of CodeGPT, its error rate is the highest (0.41) and the AUC is the lowest (0.81). This implies that using the LSTM model as the surrogate is less effective than the other models in the attack.

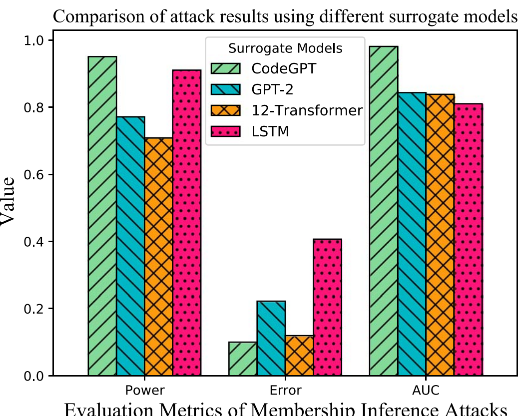

Fig. 3. The impact of different choices of surrogate models on the attack performance.

In summary, the selection of a surrogate model greatly impacts the success of membership inference attacks. More specifically, an attacker gains a considerable advantage when employing a surrogate model that closely resembles the victim model, e.g., knowing the architecture and pre-training data of the victim model.

The ratio of training data that is known to the attacker. We evaluate using two settings: 10% and 20% of the training data are known to the attacker. The results of our experiments are presented in Table V, which illustrates the attack performance under different ratios of known training data. As demonstrated in the table, the attack performance varies across different models and ratios of known training data.

The specific impact of this increase varies depending on the model architecture and the performance metric considered. For instance, while the CodeGPT and 12-Transformer models show a consistent improvement in attack success rates across all metrics as the proportion of known training data increases, the GPT-2 and LSTM models exhibit more nuanced behavior. The GPT-2 model experiences a decrease in attack success rates when more training data is known, while the LSTM model shows mixed results, with improvements in some metrics and declines in others. Overall, 3 out of four models show an increase in attack success rates as the proportion of known training data increases. In general, our results support the intuition that increasing the proportion of known training data leads to a higher attack success rate.

Answers to RQ2: The number of training epochs of the victim model has little impact on the risk of membership leakage. However, the risk is higher if an attacker knows the victim model better, e.g., the model's architecture and training data.

RQ3. What are the Features of the Training Examples Whose Memberships Are More Likely to be Correctly Inferred?

In this research question, we investigate the features of the training examples whose memberships are more likely to be correctly inferred. We use the same experiment setup in RQ1. We split the training examples into two groups: (1) successfully

Authorized licensed use limited to: Korea University. Downloaded on April 03,2025 at 02:59:56 UTC from IEEE Xplore. Restrictions apply.

TABLE V
ATTACK PERFORMANCES WHEN THE ATTACKER KNOWS DIFFERENT PORTION OF THE VICTIM MODEL'S TRAINING DATA. THE 10% AND 20% MEAN THE PERCENTAGES OF TRAINING DATA KNOWN TO THE ATTACKER. THE DIRECTIONS OF THE ARROWS INDICATE HOW THE METRICS CHANGE WHEN THE RATIO OF KNOWN TRAINING DATA INCREASES: ↑ MEANS THE METRIC INCREASES AND ↓ MEANS THE METRIC DECREASES. THE COLORS OF THE ARROWS INDICATE HOW THE ATTACK PERFORMANCE CHANGES: BLUE MEANS THE ATTACK PERFORMANCE IMPROVES (E.G., THE ATTACKER'S POWER INCREASES) AND RED MEANS THE ATTACK PERFORMANCE DEGRADES (E.G., THE ATTACKER'S ERROR INCREASE)

Model	Power		Error		AUC	
	10%	20%	10%	20%	10%	20%
CodeGPT	0.87	0.95↑	0.23	0.10↓	0.89	0.98↑
GPT-2	0.87	0.77↓	0.20	0.22↑	0.90	0.84↓
12-Transformer	0.63	0.71↑	0.33	0.12↓	0.70	0.84↑
LSTM	0.74	0.91↑	0.32	0.41↑	0.77	0.81↑

TABLE VI
THE DIFFERENCE BETWEEN THE FEATURE VALUES OF SUCCESSFULLY ATTACKED AND UNSUCCESSFULLY ATTACKED EXAMPLES

Features	Success	Unsuccess	p-value	Effect Size
Input length	326.07	285.33	<0.01	Negligible
Output length	6.06	5.67	<0.01	Negligible
Perplexity	9.190	27.08	<0.01	Small
Edit Distance	16.21	18.83	<0.01	Negligible
BLEU Score	16.21	18.83	<0.01	Negligible
No. variables	6.66	5.81	<0.01	Negligible

inferred examples and (2) unsuccessfully inferred examples. We compute a list of features of these examples and compare the two groups of examples. Below are the features we consider and the corresponding intuition.

1) **The number of tokens in an example input**. Inputs with more tokens could contain a higher amount of information, allowing the attacker to more easily make correct inferences on the membership of the example. We use white space to split the input into tokens and count the number of tokens.
2) **The number of tokens in the model output**. Longer outputs can potentially reveal more information about the model, which might make the model more susceptible to attacks. Similar to the input, we use white space to split the output into tokens and count the number of output tokens.
3) **Victim model's perplexity on the example**. The perplexity of the victim model on an example can be used to measure the model's confidence in its prediction. A low perplexity score typically indicates that the language model has learned the patterns in the training data that are relevant to the example, suggesting that the example might be part of the training data. We follow the definition of perplexity in Section IV-C2 to compute the perplexity of the victim model on each example.
4) **The edit distance between the victim model output and the ground truth**. The edit distance measures the minimum number of operations (insertion, deletion, or substitution) required to transform one string (the model output) into another (the ground truth). A smaller edit distance indicates a closer match between the model output and the ground truth, suggesting that the model performs well on the input, meaning that this input is more likely to be in the training data. We use a python package called `nltk` to compute the edit distance between the model output and the ground truth.
5) **The number of variable names in the example**. Source code contains shared keywords (e.g., `while`, `def`, etc), which are common to all programs written in a particular programming language, and human-defined variables, which are unique to specific programs. Intuitively, The presence of unique variable names in the code can make an example more identifiable and distinguishable from other examples in the dataset. We use a python package called `tree-sitter` [63] to extract the variable names from the input and count their number of occurrences.
6) **The BLEU score between the model output and the ground truth**. The BLEU score is a metric used to evaluate the quality of machine-generated content. A higher BLEU score indicates that the model output is more similar to the ground truth, suggesting that the model may have been trained on this example. We use the `sentence-bleu` function in the `nltk` package to compute the BLEU score between the model output and the ground truth.

We split the evaluation examples into two groups: examples that are successfully inferred by the attacker and examples that are not. Then, for examples in each group, we compute the features described above. Taking the perplexity as an example, we obtain two lists of perplexity scores for the two groups of examples. We conduct a Wilcoxon rank-sum test [60] to determine whether the two groups of examples have statistically significant differences in the feature values. The Wilcoxon rank-sum test is a non-parametric statistical hypothesis test used to compare two related samples, which is widely used in the literature to understand the feature differences between two groups of examples [64], [65], [66]. The null hypothesis is that there is no significant difference between the two groups of examples in terms of the feature values. If the p-value of the test is less than 0.05, we reject the null hypothesis and conclude that there is a significant difference between the two groups of examples in terms of the feature values. The statistical testing results are presented in Table VI. From the table, we observe that there are statistically significant differences (with p-values less than 0.01) between the successfully and unsuccessfully inferred examples for all the features analyzed.

In addition, following previous studies [64], [65], [66] we calculate the Cohen's effect sizes (δ), which are statistical measures used to quantify the magnitude of differences or relationships. To determine the significance of the effect size, we adopt a guideline [67] which states that an effect size of $|\delta|$ less than 0.2 is considered negligible, between 0.2 and 0.5 is

Authorized licensed use limited to: Korea University. Downloaded on April 03,2025 at 02:59:56 UTC from IEEE Xplore. Restrictions apply.

small, between 0.5 and 0.8 is medium, and larger than 0.8 is large. The effect size of these differences varies across features. For input length, output length, edit distance, and the number of variables, the effect size is negligible, indicating that while there are statistically significant differences between the two groups, these differences may not have a substantial impact on the susceptibility of the model to membership inference attacks.

The effect size for perplexity is larger than that of other features, suggesting that these features may have a more significant impact on the model's vulnerability to membership inference attacks. A lower perplexity score for successfully attacked examples indicates that the victim model has a higher confidence in its predictions for these examples, potentially because the model has learned patterns in the training data relevant to these examples. In summary, our analysis reveals that the differences in feature values between the two groups of examples are statistically significant. However, the effect size is small or negligible for most features, indicating that these features alone may not be strong indicators of a model's susceptibility to membership inference attacks.

Answers to RQ3: MIA classifiers tend to perform better on examples that have lower perplexity scores. However, input length, output length, edit distance, and the number of variables show negligible effect sizes.

VI. Discussion

A. How to Defense Against MIA?

For domains that are well studied, e.g., image classification, researchers have proposed a series of defensive methods to protect models, e.g., DP-SGD [68], model ensemble [69], and adversarial regularization [70]. However, to the best of our knowledge, there is no existing work designed for protecting code generation models. By default, the CodeGPT model uses beam search [35] as the decoding strategy. We let the model use another decoding strategy called "top-k sampling" [71] to generate code. This approach selects the k most likely tokens from the probability distribution at each time step and then samples from those tokens to generate the next word in the sequence. This method can result in more diverse and creative outputs compared to beam search, which tends to generate more focused and deterministic outputs. By introducing more randomness and diversity into the generated outputs, top-k sampling can make it harder for an attacker to correlate the outputs with specific training data points.

There are two important hyperparameters in the top-k sampling strategy [71]: (1) the value of k and (2) the *temperature*. The 'k' value determines the number of most likely tokens to consider for sampling at each step. A smaller 'k' results in a more focused set of tokens, leading to more deterministic and coherent text generation, but may also cause the output to be repetitive and less creative. A larger 'k' allows for a more diverse set of tokens to be considered, resulting in more diverse and creative outputs. The temperature is a scaling factor applied to the logits before converting them into probabilities using the softmax function. A higher temperature value flattens the distribution, promoting diversity and creativity in the text but potentially leading to less coherent and more random outputs. We change the decoding strategy from beam search to top-k sampling. We also try different combinations of the 'k' and temperature to see whether the leakage risk can be mitigated.

TABLE VII
How Victim Model's Performance and Attack Results Changes When Using Different Decoding Hyperparameters

Variation	k	*temp*	BLUE	Power	Error	AUC
Vary Temp	50	0.1	0.63	0.64	0.52	0.59
	50	0.5	0.61	0.64	0.52	0.59
	50	1	0.57	0.64	0.52	0.59
	50	2	0.36	0.64	0.52	0.59
Vary k	10	1	0.58	0.64	0.52	0.59
	50	1	0.57	0.64	0.52	0.59
	100	1	0.57	0.64	0.52	0.59
Beam-search to decode			0.59	0.95	0.10	0.98

Table VII illustrates how the top-k sampling strategy and its hyperparameters can affect the risk of membership leakage. We can observe that changing the decoding strategy can mitigate such risk. To be more specific, the AUC score changes from 0.98 to 0.59, showing that the top-k sampling strategy can reduce the risk of membership leakage by around 40%. However, the performance of MIA is not sensitive to the choice of values of k and temperature. We try 7 different combinations of the 'k' value and the temperature, and the AUC score is always around 0.59. This simple defense strategy has minimal impact on the performance of the victim model. As indicated by the BLUE score, when the temperature value is low (e.g., less than 1), the BLUE score is close to that of the beam search decoding strategy (0.59). The result is consistent with the finding from the ablation study in RQ1 that the model output contributes most to the effectiveness of the proposed approach.

B. Ethical Considerations

The progress in code models and their applications can greatly benefit society, but it is crucial to consider the potential privacy and security risks associated with them. Our aim is not to promote or facilitate malicious behavior, but rather to raise awareness of the potential risks associated with code models and to contribute to the development of secure and privacy-preserving code models.

It is imperative that ethical considerations are taken into account when using code models and other machine learning models. This includes responsible data handling and privacy protection, as well as ensuring that these models are used for the benefit of society. As researchers and practitioners in the field of software engineering, it is our responsibility to ensure that our work is used for ethical and beneficial purposes. We hope that this paper will contribute to the ongoing discussion on the ethical use of code models and other machine learning models, and help to promote responsible and ethical research practices.

Authorized licensed use limited to: Korea University. Downloaded on April 03,2025 at 02:59:56 UTC from IEEE Xplore. Restrictions apply.

C. Threats to Validity

Threats to Internal Validity. While we have explored the risk of membership inference leakage against various MIA attacks, we acknowledge that the performance of these methods may be affected by the choice of hyperparameters and randomness. To mitigate these threats, we repeat the experiments 3 times and report the average results. We use the hyperparameters in the original paper [5] to train the victim model and implement the attacks. When evaluating the metric ranking-based methods, we set the cut-off point as 50% to determine whether an example is in the training data because the evaluation dataset is balanced. In practice, the attacker can adjust the cut-off position based on their specific objectives and the nature of the dataset they are targeting. For example, if an attacker aims to increase the true positive rate, they may opt for a higher cut-off position, such as 70% or 80%. While this approach may increase the likelihood of including more actual training examples, it also comes with a trade-off: the false positive rate will likely rise, leading to more non-training examples being incorrectly classified as training examples.

We also acknowledge that the choice of experiment designs may affect the results. For example, to evaluate the impact of each factor on the attack performance, we leave other factors unchanged, vary one factor, and observe the change in the attack performance. Such evaluation settings that analyze each factor separately are also widely adopted in the literature that analyzes AI systems. For example, Yan et al. [72] analyze how training data size affects model robustness by fixing other hyperparameters and gradually increasing training data size to compare the robustness of each model, which is similar to our setting for analyzing how the number of training epochs affects privacy risks. We plan to investigate the interaction between different factors and their impacts on privacy risks (e.g., using Design of Experiments [73]) in future work.

Threats to External Validity. In the context of the paper, external validity is related to the generalizability of the findings regarding the membership leakage risk of CodeGPT to other code models. The findings may not generalize to other code models or language models with different architectures, e.g., LSTM. The paper focuses on the CodeGPT model, a recently proposed model that leverages the GPT architecture. The GPT architecture is the foundation of many state-of-the-art language models, e.g., ChatGPT. The selection of surrogate models may also affect the membership inference results. To understand this potential impact, we evaluate four different surrogate models in RQ2 and confirm that in all four settings our method outperforms the baselines. This study uses the JavaCorpus dataset in the experiments, a large collection of Java code. We acknowledge the results may not generalize to other programming languages, e.g., Python, C++, etc. We believe that this threat is minimal as our method is programming language agnostic and can be extended to other programming languages. We leave evaluation on other programming languages as future work and encourage other researchers to validate our findings on more datasets, including non-Java datasets.

Moreover, while our study uses a balanced dataset for evaluation, we acknowledge that in real-world scenarios, datasets often exhibit imbalanced class distributions. In such cases, attackers might need to adapt their strategies accordingly, potentially selecting cut-off points that better reflect the underlying distribution.

The victim models evaluated in this paper are static, i.e., they are fine-tuned once and do not change during the attack. The results obtained in this paper may not generalize to dynamic models that are continuously updated or retrained. Training a new membership information classifier should mitigate such potential threats. The assumption of static victim models comes from previous studies and is widely adopted [13], [16], [19], [33], [34]. The investigation of MIA for dynamically changing models deserves a separate line of work, and we leave it for future work.

VII. Related Work

A. Code Models and Threats

Recent studies have highlighted that code models are vulnerable to various attacks and threats. Yefet et al. [14] employed the Fast Gradient Sign Method [74] to adversarially transform source code, resulting in changes to the output of code models such as code2vec, GGNN, and GNN-FiLM. Yang et al. [13] emphasized the naturalness requirement in creating adversarial examples of code. Srikant et al. [15] used stronger adversarial algorithms (PGD [75]) to generate adversarial examples with higher success rates. These studies utilize white-box information (e.g., model parameters, gradients) to conduct attacks, however, there are also works on black-box attacks where an attacker only has access to the model's input and output. Zhang et al. [45] modeled code attacks as a stochastic process and designed the Metropolis-Hastings Modifier (MHM) to generate adversarial examples for code. Wei et al. [76] proposed a coverage-guided fuzzing algorithm to test code models. Additionally, several works have evaluated code models against adversarial attacks using semantic preserving transformations [77], [78], [79], [80].

There are new threats emerging for code models. Nguyen et al. [20] conducted data poisoning attacks on API recommendation systems, finding that all three investigated systems were vulnerable to attacks that simply injected small amounts of malicious data into the training set. Schuster et al. [18] performed data poisoning attacks on code completion models, showing that by injecting malicious code snippets into the training set, the code completion models produced code with security vulnerabilities (e.g., using insecure APIs when encryption) in critical contexts. Data poisoning can also be used to inject backdoors into code models [81]. Wan et al. [19] used fixed and grammar triggers to implant backdoors in code search models. Yang et al. [25] proposed the use of adversarial features to create stealthy backdoors in code models. Li et al. [82] leveraged another code model to generate dynamic backdoors in code models. Data poisoning can also be used as a protection mechanism. Sun et al. [24] proposed using data poisoning to prevent open-source data from being trained without authorization.

Authorized licensed use limited to: Korea University. Downloaded on April 03,2025 at 02:59:56 UTC from IEEE Xplore. Restrictions apply.

To the best of our knowledge, our study presents the first systematic investigation of membership inference attacks on code models.[7] The existing works and this study demonstrate the vulnerability of code models, highlighting the need for vulnerability evaluation and mitigation techniques to protect against these types of attacks.

B. Privacy Attacks on DNN Models

This paper investigates the MIA [41], [83], [84], which can serve as the gate to a series of other privacy attacks. Data extraction attack [34] aims to extract training data from a victim model. Model extraction attacks [85], [86] are designed to steal information about the victim model, e.g., model parameters, model architecture, model functionality, etc. However, defense mechanisms for generation models (including text generation) have not been as extensively studied as those for other types of tasks, such as classification. For example, Nasr et al. [70] leverage adversarial training as a defense against MIA. Inspired by differential privacy, researchers also propose to train models with differential privacy [68] to protect the models. The proposed methods only demonstrate effectiveness on classification tasks. Both adversarial training and differential privacy training are extremely expensive and may not be feasible for large-scale generative models like CodeGPT.

Privacy attacks can target at different types of data. A large portion of effort has been devoted to the privacy attacks on image data [56], [84], [87], [88], However, other types of data, such as tabular, text, and time-series data, have received comparatively less attention. Popular benchmarks of tabular data in the context of privacy attacks include the UCI's diabetes dataset, German Credit Dataset [89], Adult Income Dataset [89], etc. Some datasets of texts that contain sensitive information also suffer from privacy attacks, e.g., Yelp healthcare-related reviews [90]. To the best of our knowledge, our paper presents the first study on privacy attacks on code models and datasets. Privacy attacks can target at different types of tasks. Classification tasks are the most prevalent type of tasks in privacy attacks, e.g., image classification [41], income classification [91], text classification [83], etc. Less attention has been paid to other types of tasks, such as generation tasks. In generation tasks, the privacy risk of generative adversarial network (GAN) models is more well-studied [92]. Another important task in generation tasks is the text generation. Hisamoto et al. [33] conduct MIA on machine translation systems. Carlini et al. [34] leverage MIA to extract training data from text models. This paper proposes an effective attack on code models and uses the two attacks as baselines.

Our study also suggests insights from the previous study [56] may not generalize to code models. For example, Yeom et al. [56] show that the number of training epochs can affect the membership leakage risk while our experiment finds the effect is negligible. This discrepancy may be due to the different characteristics of the models, data, and tasks, studied in the two papers. Their conclusion is drawn from image classification and regression tasks on small models, while our study focuses on code completion tasks on large language models.

[7] We put the preprint of this paper on arXiv on 2 October 2023. URL: https://arxiv.org/abs/2310.01166

VIII. Conclusion and Future Work

In conclusion, this paper has shed light on the significant privacy concerns surrounding the use of code models, specifically in terms of membership information leakage. We introduce GOTCHA, a novel membership inference attack method, and evaluated its efficacy against CodeGPT, an open-source code completion model. Our findings reveal that the risk of membership inference attacks is alarmingly high, with GOTCHA achieving a high true positive rate 0.95 and a low false positive rate 0.10. Furthermore, we demonstrate that an attacker's chances of success increase with more knowledge of the victim model, such as its architecture. These findings serve as a call to action for the research community to pay greater attention to the privacy implications of code models and to develop more effective countermeasures against privacy attacks. Future work should aim to investigate more sophisticated defense mechanisms, explore other potential privacy risks, and establish best practices for the secure and responsible use of code models.

In future work, we plan to investigate membership leakage risks in code models with different architectures and programming languages. Also, we plan to design more effective countermeasures against membership inference attacks.

The replication package is available at https://github.com/yangzhou6666/MIA-LLM4Code, which is intended for academic and research purposes only. We do not condone or support the use of the replication package for malicious purposes.

Acknowledgment

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Human Lang. Technol.*, Volume 1 (Long Short Papers). Minneapolis, MN, USA: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[2] Y. Liu et al., "RoBERTa: A robustly optimized BERT pretraining approach," 2019, *arXiv:1907.11692*.

[3] C. Raffel et al., "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, no. 140, pp. 1–67, 2020. [Online]. Available: http://jmlr.org/papers/v21/20-074.html

[4] W. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "Unified pre-training for program understanding and generation," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Human Lang. Technol.*, Online: Association for Computational Linguistics, Jun. 2021, pp. 2655–2668.

[5] S. Lu et al., "CodeXGLUE: A machine learning benchmark dataset for code understanding and generation," 2021, *arXiv:2102.04664*.

[6] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "CodeSearchNet challenge: Evaluating the state of semantic code search," 2019, *arXiv:1909.09436*.

Authorized licensed use limited to: Korea University. Downloaded on April 03,2025 at 02:59:56 UTC from IEEE Xplore. Restrictions apply.

[7] X. Hou et al., “Large language models for software engineering: A systematic literature review,” *ACM Trans. Softw. Eng. Methodol.*, Sep. 2024, doi: https://doi.org/10.1145/3695988.
[8] M. Izadi, R. Gismondi, and G. Gousios, “CodeFill: Multi-token code completion by jointly learning from structure and naming sequences,” in *Proc. IEEE/ACM 44th Int. Conf. Softw. Eng. (ICSE)*, 2022, pp. 401–412.
[9] C. Yang et al., “Aspect-based API review classification: How far can pre-trained transformer model go?” in *Proc. IEEE Int. Conf. Softw. Anal., Evolution Reeng. (SANER)*, Los Alamitos, CA, USA: IEEE Comput. Soc. Press, Mar. 2022, pp. 385–395. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SANER53432.2022.00054
[10] C. Yang et al., *Answer Summarization for Technical Queries: Benchmark and New Approach*. New York, NY, USA: ACM, 2023, doi: 10.1145/3551349.3560421.
[11] A. Mazuera-Rozo, A. Mojica-Hanke, M. Linares-Vásquez, and G. Bavota, “Shallow or deep? An empirical study on detecting vulnerabilities using deep learning,” in *Proc. IEEE/ACM 29th Int. Conf. Program Comprehension (ICPC)*, 2021, pp. 276–287.
[12] M. Chen et al., “Evaluating large language models trained on code,” 2021, *arXiv:2107.03374*.
[13] Z. Yang, J. Shi, J. He, and D. Lo, “Natural attack for pre-trained models of code,” in *Proc. 44th Int. Conf. Softw. Eng. (ICSE ’22)*, New York, NY, USA: ACM, 2022, pp. 1482–1493, doi: 10.1145/3510003.3510146.
[14] N. Yefet, U. Alon, and E. Yahav, “Adversarial examples for models of code,” *Proc. ACM Program. Lang.*, vol. 4, no. OOPSLA, pp. 162:1–162:30, 2020.
[15] S. Srikant et al., “Generating adversarial computer programs using optimized obfuscations,” in *Proc. Int. Conf. Learn. Representations (ICLR)*, vol. 16, pp. 209–226, 2021.
[16] J. Henkel, G. Ramakrishnan, Z. Wang, A. Albarghouthi, S. Jha, and T. Reps, “Semantic robustness of models of source code,” in *Proc. IEEE Int. Conf. Softw. Anal., Evolution Reeng. (SANER)*, 2022, pp. 526–537.
[17] T.-D. Nguyen, Z. Yang, X. B. D. Le, and D. Lo, “Adversarial attacks on code models with discriminative graph patterns,” *CoRR*, 2023, *arXiv:2308.11161*.
[18] R. Schuster, C. Song, E. Tromer, and V. Shmatikov, “You autocomplete me: Poisoning vulnerabilities in neural code completion,” in *Proc. 30th USENIX Secur. Symp. (USENIX Secur. 21)*, USENIX Association, Aug. 2021, pp. 1559–1575.
[19] Y. Wan et al., “You see what I want you to see: Poisoning vulnerabilities in neural code search,” in *Proc. 30th ACM Joint Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng. (ESEC/FSE)*, New York, NY, USA: ACM, 2022, pp. 1233–1245, doi: 10.1145/3540250.3549153.
[20] P. T. Nguyen, C. Di Sipio, J. Di Rocco, M. Di Penta, and D. Di Ruscio, “Adversarial attacks to API recommender systems: Time to wake up and smell the coffee?” in *Proc. 36th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, 2021, pp. 253–265.
[21] Z. Yang et al., “Unveiling memorization in code models,” in *Proc. IEEE/ACM 46th Int. Conf. Softw. Eng. (ICSE)*, New York, NY, USA: ACM, 2024, doi: 10.1145/3597503.3639074.
[22] L. Niu, S. Mirza, Z. Maradni, and C. Pöpper, “CodexLeaks: Privacy leaks from code generation language models in GitHub copilot,” in *Proc. 32nd USENIX Secur. Symp. (USENIX Secur.)*, Anaheim, CA, USA: USENIX Association, Aug. 2023, pp. 2133–2150.
[23] M. L. Siddiq, S. H. Majumder, M. R. Mim, S. Jajodia, and J. C. S. Santos, “An empirical study of code smells in transformer-based code generation techniques,” in *Proc. IEEE 22nd Int. Work. Conf. Source Code Anal. Manipulation (SCAM)*, 2022, pp. 71–82.
[24] Z. Sun, X. Du, F. Song, M. Ni, and L. Li, “CoProtector: Protect open-source code against unauthorized training usage with data poisoning,” in *Proc. ACM Web Conf. (WWW)*, New York, NY, USA: ACM, 2022, pp. 652–660.
[25] Z. Yang et al., “Stealthy backdoor attack for code models,” *IEEE Trans. Softw. Eng.*, vol. 50, no. 4, pp. 721–741, Apr. 2024.
[26] “AWS CodeWhisperer: Features.” Amazon Web Services. Accessed: Mar. 29, 2023. [Online]. Available: https://aws.amazon.com/codewhisperer/features/
[27] S. K. Basak, L. Neil, B. Reaves, and L. Williams, “SecretBench: A dataset of software secrets,” in *Proc. 20th Int. Conf. Mining Softw. Repositories (MSR ’23)*, 2023.
[28] E. Nijkamp et al., “CodeGen: An open large language model for code with multi-turn program synthesis,” in *Proc. 11th Int. Conf. Learn. Representations*, 2023.
[29] “codeparrot (codeparrot),” huggingface.co. [Online]. Available: https://huggingface.co/codeparrot
[30] S. Black, L. Gao, P. Wang, C. Leahy, and S. Biderman, “GPT-Neo: Large scale autoregressive language modeling with mesh-Tensorflow,” Mar. 2021, If you use this software, please cite it using these metadata. doi: 10.5281/zenodo.5297715.
[31] F. F. Xu, U. Alon, G. Neubig, and V. J. Hellendoorn, “A systematic evaluation of large language models of code,” in *Proc. 6th ACM SIGPLAN Int. Symp. Mach. Program. (MAPS)*, New York, NY, USA: ACM, 2022, pp. 1–10, doi: 10.1145/3520312.3534862.
[32] M. Allamanis and C. Sutton, “Mining source code repositories at massive scale using language modeling,” in *Proc. 10th IEEE Work. Conf. Mining Softw. Repositories (MSR)*, Los Alamitos, CA, USA: IEEE Comput. Soc. Press, May 2013, pp. 207–216. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/MSR.2013.6624029
[33] S. Hisamoto, M. Post, and K. Duh, “Membership inference attacks on sequence-to-sequence models: Is my data in your machine translation system?” *Trans. Assoc. Comput. Linguistics*, vol. 8, pp. 49–63, 2020. [Online]. Available: https://aclanthology.org/2020.tacl-1.4
[34] N. Carlini et al., “Extracting training data from large language models,” in *Proc. USENIX Secur. Symp.*, 2021, pp. 2633–2650.
[35] M. Freitag and Y. Al-Onaizan, “Beam search strategies for neural machine translation,” in *Proc. 1st Workshop Neural Mach. Transl.*, Vancouver, Canada: Association for Computational Linguistics, Aug. 2017, pp. 56–60. [Online]. Available: https://aclanthology.org/W17-3207
[36] Z. Feng et al., “CodeBERT: A pre-trained model for programming and natural languages,” in *Findings Assoc. Comput. Linguistics (EMNLP),* Association for Computational Linguistics, Nov. 2020, pp. 1536–1547.
[37] D. Guo et al., “GraphCodeBERT: Pre-training code representations with data flow,” in *Proc. 9th Int. Conf. Learning Representations (ICLR),* Virtual Event, Austria, May 3–7, 2021.
[38] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, “Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation,” in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2021, pp. 8696–8708.
[39] V. Raychev, P. Bielik, and M. Vechev, “Probabilistic model for code with decision trees,” in *Proc. ACM SIGPLAN Int. Conf. Object-Oriented Program., Syst., Lang., Appl. (OOPSLA)*, New York, NY, USA: ACM, 2016, pp. 731–747, doi: 10.1145/2983990.2984041.
[40] H. Ye, M. Martinez, and M. Monperrus, “Neural program repair with execution-based backpropagation,” in *Proc. 44th Int. Conf. Softw. Eng. (ICSE)*, New York, NY, USA: ACM, 2022, pp. 1506–1518, doi: 10.1145/3510003.3510222.
[41] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, “Membership inference attacks against machine learning models,” in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2017, pp. 3–18.
[42] X. Zheng and J. Jiang, “An empirical study of memorization in NLP,” in *Proc. 60th Annu. Meeting Assoc. Comput. Linguistics* (Volume 1: Long Papers), Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 6265–6278. [Online]. Available: https://aclanthology.org/2022.acl-long.434
[43] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, “Asleep at the keyboard? Assessing the security of GitHub copilot’s code contributions,” in *Proc. 43rd IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA. Piscataway, NJ, USA: IEEE Press, 2022, pp. 754–768, doi: 10.1109/SP46214.2022.9833571.
[44] D. G. Feitelson, “‘We do not appreciate being experimented on’: Developer and researcher views on the ethics of experiments on open-source projects,” 2021, *arXiv:2112.13217*.
[45] H. Zhang, Z. Li, G. Li, L. Ma, Y. Liu, and Z. Jin, “Generating adversarial examples for holding robustness of source code processing models,” *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 1, pp. 1169–1176, Apr. 2020.
[46] X. Zhou, D. Han, and D. Lo, “Assessing generalizability of CodeBERT,” in *Proc. IEEE Int. Conf. Softw. Maintenance Evolution (ICSME)*, 2021, pp. 425–436.
[47] Z. Zeng, H. Tan, H. Zhang, J. Li, Y. Zhang, and L. Zhang, “An extensive study on pre-trained models for program understanding and generation,” in *Proc. 31st ACM SIGSOFT Int. Symp. Softw. Testing Anal. (ISSTA)*, New York, NY, USA: ACM, 2022, pp. 39–51, doi: 10.1145/3533767.3534390.
[48] J. He, B. Xu, Z. Yang, D. Han, C. Yang, and D. Lo, “PTM4Tag: Sharpening tag recommendation of stack overflow posts with pre-trained models,” in *Proc. 30th IEEE/ACM Int. Conf. Program Comprehension (ICPC)*, New York, NY, USA: ACM, 2022, pp. 1–11, doi: 10.1145/3524610.3527897.

Authorized licensed use limited to: Korea University. Downloaded on April 03,2025 at 02:59:56 UTC from IEEE Xplore. Restrictions apply.

[49] J. He et al., "Representation learning for stack overflow posts: How far are we?" *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 3, Mar. 2024, Art. no. 69.

[50] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[51] K. Jesse, T. Ahmed, P. T. Devanbu, and E. Morgan, "Large language models and simple, stupid bugs," in *Proc. IEEE/ACM 20th Int. Conf. Mining Softw. Repositories (MSR)*, Los Alamitos, CA, USA: IEEE Comput. Soc. Press, May 2023, pp. 563–575. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/MSR59073.2023.00082

[52] R. Karampatsis, H. Babii, R. Robbes, C. Sutton, and A. Janes, "Big code != big vocabulary: Open-vocabulary models for source code," in *Proc. IEEE/ACM 42nd Int. Conf. Softw. Eng. (ICSE)*, Los Alamitos, CA, USA: IEEE Comput. Soc. Press, Oct. 2020, pp. 1073–1085. [Online]. Available: https://doi.ieeecomputersociety.org/

[53] C.-Y. Lin and F. J. Och, "Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics," in *Proc. 42nd Annu. Meeting Assoc. Comput. Linguistics (ACL)*, USA. Barcelona, Spain: Association for Computational Linguistics, 2004, p. 605–es, doi: 10.3115/1218955.1219032.

[54] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," *Comput. Speech Lang.*, vol. 13, no. 4, pp. 359–394, 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0885230899901286

[55] "A massively spiffy yet delicately unobtrusive compression library," Accessed Mar. 27, 2023. [Online]. Available: https://zlib.net/

[56] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha, "Privacy risk in machine learning: Analyzing the connection to overfitting," in *Proc. IEEE 31st Comput. Secur. Found. Symp. (CSF)*, 2018, pp. 268–282.

[57] D. Montgomery, *Design and Analysis of Experiments 7th Edition with Student Solutions Manual and Design Expert 7. 0. 3 Set*. Hoboken, NJ, USA: Wiley, 2009. [Online]. Available: https://books.google.com.sg/books?id=S2CQuAAACAAJ

[58] D. Cotroneo, C. Improta, P. Liguori, and R. Natella, "Vulnerabilities in ai code generators: Exploring targeted data poisoning attacks," in *Proc. 32nd IEEE/ACM Int. Conf. on Program. Comp.*, Lisbon, Portugal. New York, NY, USA: Association for Computing Machinery, 2024, pp. 280–292.

[59] L. St and S. Wold, "Analysis of variance (ANOVA)," *Chemometrics Intell. Lab. Syst.*, vol. 6, no. 4, pp. 259–272, 1989.

[60] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bull.*, vol. 1, no. 6, pp. 80–83, 1945. [Online]. Available: http://www.jstor.org/stable/3001968

[61] L. E. Toothaker, *Multiple Comparison Procedures*. Newbury Park, CA, USA: Sage, 1993, no. 89.

[62] P. Sedgwick, "Multiple significance tests: The Bonferroni correction," *Bmj*, vol. 344, 2012, Art. no. e509.

[63] "Tree-sitter: An incremental parsing library," Tree-sitter. Accessed: Mar. 25, 2023. [Online]. Available: https://tree-sitter.github.io/tree-sitter/

[64] Y. Fan, X. Xia, D. Lo, A. E. Hassan, and S. Li, "What makes a popular academic AI repository?" *Empirical Softw. Eng.*, vol. 26, no. 1, pp. 1–35, 2021.

[65] Z. Yang et al., "Prioritizing speech test cases," 2023, *arXiv:2302.00330*.

[66] Z. Yang et al., "What do users ask in open-source AI repositories? An empirical study of GitHub issues," in *Proc. 20th Int. Conf. Mining Softw. Repositories (MSR)*, 2023, pp. 79–91.

[67] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*, 2nd ed. Hillsdale, NJ, USA: Lawrence Erlbaum Associates, 1988.

[68] M. Abadi et al., "Deep learning with differential privacy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA: ACM, 2016, pp. 308–318, doi: 10.1145/2976749.2978318.

[69] X. Tang et al., "Mitigating membership inference attacks by {Self-Distillation} through a novel ensemble architecture," in *Proc. 31st USENIX Secur. Symp. (USENIX Secur.)*, 2022, pp. 1433–1450.

[70] M. Nasr, R. Shokri, and A. Houmansadr, "Machine learning with membership privacy using adversarial regularization," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA: ACM, 2018, pp. 634–646, doi: 10.1145/3243734.3243855.

[71] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015.

[72] S. Yan et al., "Correlations between deep neural network model coverage criteria and model quality," in *Proc. 28th ACM Joint Meeting Eur. Softw. Eng. Conf./Symp. Found. Softw. Eng. (ESEC/FSE)*, New York, NY, USA: ACM, 2020, pp. 775–787, doi: 10.1145/3368089.3409671.

[73] R. A. Fisher et al., *The Design of Experiments*, vol. 21. Springer, 1966.

[74] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. Int. Conf. Learn. Representations*, 2015, *arXiv:1412.6572*.

[75] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proc. 6th Int. Conf. Learn. Representations (ICLR)*, Vancouver, BC, Canada, 2018.

[76] M. Wei, Y. Huang, J. Yang, J. Wang, and S. Wang, "CoCoFuzzing: Testing neural code models with coverage-guided fuzzing," *IEEE Trans. Rel.*, vol. 72, no. 3, pp. 1276–1289, Sep. 2022.

[77] M. R. I. Rabin, N. D. Bui, K. Wang, Y. Yu, L. Jiang, and M. A. Alipour, "On the generalizability of neural program models with respect to semantic-preserving program transformations," *Inf. Softw. Technol.*, vol. 135, 2021, Art. no. 106552.

[78] L. Applis, A. Panichella, and A. van Deursen, "Assessing robustness of ML-based program analysis tools using metamorphic program transformations," in *Proc. 36th IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE)*, 2021, pp. 1377–1381.

[79] M. V. Pour, Z. Li, L. Ma, and H. Hemmati, "A search-based testing framework for deep neural networks of source code embedding," in *Proc. 14th IEEE Conf. Softw. Testing, Verification Validation (ICST)*, Porto de Galinhas, Brazil. Piscataway, NJ, USA: IEEE Press, 2021.

[80] A. Jha and C. K. Reddy, "CodeAttack: Code-based adversarial attacks for pre-trained programming language models," in *Proc. 37th AAAI Conf. Artif. Intell./35th Conf. Innovative Appl. Artif. Intell./13th Symp. Educ. Adv. Artif. Intell. (AAAI/IAAI/EAAI)*, AAAI Press, 2023, doi: 10.1609/aaai.v37i12.26739.

[81] G. Ramakrishnan and A. Albarghouthi, "Backdoors in neural models of source code," in *Proc. 26th Int. Conf. Pattern Recognit. (ICPR)*, Los Alamitos, CA, USA: IEEE Comput. Soc. Press, Aug 2022, pp. 2892–2899. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICPR56361.2022.9956690

[82] J. Li et al., "Poison attack and defense on deep source code processing models," 2022, *arXiv:2210.17029*.

[83] F. Mireshghallah, K. Goyal, A. Uniyal, T. Berg-Kirkpatrick, and R. Shokri, "Quantifying privacy risks of masked language models using membership inference attacks," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 8332–8347. [Online]. Available: https://aclanthology.org/2022.emnlp-main.570

[84] S. Truex, L. Liu, M. E. Gursoy, L. Yu, and W. Wei, "Demystifying membership inference attacks in machine learning as a service," *IEEE Trans. Services Comput.*, vol. 14, no. 6, pp. 2073–2089, Nov./Dec. 2019.

[85] K. Krishna, G. S. Tomar, A. P. Parikh, N. Papernot, and M. Iyyer, "Thieves on sesame street! model extraction of BERT-based APIs," 2019, *arXiv:1910.12366*.

[86] K. Chen, S. Guo, T. Zhang, X. Xie, and Y. Liu, "Stealing deep reinforcement learning models for fun and profit," in *Proc. ACM Asia Conf. Comput. Commun. Secur. (ASIA CCS)*, New York, NY, USA: ACM, 2021, pp. 307–319, doi: 10.1145/3433210.3453090.

[87] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, Piscataway, NJ, USA: IEEE Press, 2019, pp. 739–753.

[88] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the GAN: Information leakage from collaborative deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA: ACM, 2017, pp. 603–618, doi: 10.1145/3133956.3134012.

[89] D. Dua and C. Graff, "UCI machine learning repository," Accessed: Mar. 25, 2023. [Online]. Available: http://archive.ics.uci.edu/ml

[90] N. N. Tran and J. Lee, "Online reviews as health data: Examining the association between availability of health care services and patient star ratings exemplified by the yelp academic dataset," *JMIR Public Health Surveill*, vol. 3, no. 3, 2017, Art. no. e43.

[92] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *Proc. 25th USENIX Conf. Secur. Symp. (SEC)*, USA. Austin, TX, USA: USENIX Association, 2016, pp. 601–618.

[92] D. Chen, N. Yu, Y. Zhang, and M. Fritz, "GAN-leaks: A taxonomy of membership inference attacks against generative models," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA: ACM, 2020, pp. 343–362, doi: 10.1145/3372297.3417238.

Authorized licensed use limited to: Korea University. Downloaded on April 03,2025 at 02:59:56 UTC from IEEE Xplore. Restrictions apply.

Zhou Yang received the B.Eng. degree in software engineering from Yangzhou University and the M.Sc. degree in software system engineering from the University College London. He is currently working toward the Ph.D. degree with Singapore Management University. He is a Senior Research Engineer with Singapore Management University under the supervision of Prof. David Lo. He currently focuses on different properties of large language models of code, e.g., robustness, security, usability, etc. He has published papers in top-tier venues, including ICSE, FSE, ASE, and ISSTA, and journals such as IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, *TOSEM*, and *EMSE*. He won the ACM SIGSOFT Distinguished Paper Award and the ACM Student Research Competition Award. For more information, see https://yangzhou6666.github.io.

Zhipeng Zhao is currently working toward the master's degree with the University of Copenhagen. He is a Research Assistant with Singapore Management University under the supervision of Prof. David Lo. His research focuses on the intersection of artificial intelligence (AI) and diverse fields (AI+X), particularly exploring how AI can drive innovation, enhance problem-solving capabilities, and tackle complex challenges across multiple disciplines. His work has been featured in top-tier conferences, including ICSE and RecSys. For more information, see https://oran-ac.github.io.

Chenyu Wang is currently working toward the Ph.D. degree with the School of Computing and Information Systems (SCIS), Singapore Management University (SMU). He is a Research Engineer with SCIS, SMU. His research interests lie primarily in the intersection of software engineering (SE) and artificial intelligence (Al). He specifically focuses on the quality assurance of AI-based systems, and backdoor attacks on large language models for code (LLM4Code). His work has been published in high-quality SE conferences and journals such as ICSE, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, and MSR. For more information, see https://github.com/JamesNolan17.

Jieke Shi is currently working toward the Ph.D. degree with the School of Computing and Information Systems (SCIS), Singapore Management University (SMU). He is a Research Engineer with SCIS, SMU. His research interests lie primarily in the intersection of software engineering (SE) and artificial intelligence (AI). Particularly, he focuses on quality assurance of AI-enabled systems from an SE perspective, and efficiency improvement of code models for real-world deployment. His work has been published in high-quality SE conferences such as ICSE, ASE, and MSR. He has won/been nominated for several research paper awards. For more information, see https://jiekeshi.github.io.

Dongsun Kim received the Ph.D. degree in computer science and engineering from Sogang University, Korea. He is an Associate Professor with Korea University. His career includes several academic and industrial experiences in Hong Kong, Luxembourg, and Korea. He has published several research papers and participated in several research projects relevant to automated software engineering. In particular, he has pioneered a new line of research on pattern-based program repair. His recent achievements have focused on proactive debugging, automated fix pattern mining, deep code representation for mining fix patterns, program repair driven by bug reports, fault localization impact on program repair, and specific topics for program repair.

DongGyun Han received the Ph.D. degree with the University College London (UCL). He is a Lecturer (Assistant Professor) with the Department of Computer Science, Royal Holloway, UCL. He is mainly working on software engineering research. His main research interests are empirical study, AI for software engineering (AI4SE), software engineering for AI (SE4AI), and code review. He was a Research Scientist (Postdoc) with SOftware Analytics Research (SOAR) Group and Secure Mobile Centre, Singapore Management University (SMU). Before joining SMU, he was a Software Development Engineer with Amazon Web Services. He has worked for KAIST Institute for IT Convergence as a Researcher after getting his M.Phil. degree with Hong Kong University of Science and Technology (HKUST).

David Lo (Fellow, IEEE) is the OUB Chair Professor and the Director of the Information Systems and Technology Cluster, School of Computing and Information Systems, Singapore Management University. His research interest is in the intersection of software engineering, cybersecurity and data science, encompassing socio-technical aspects and analysis of different kinds of software artefacts, with the goal of improving software quality and security and developer productivity. He has won more than 20 international research and service awards including more than ten ACM SIGSOFT/IEEE TCSE Distinguished Paper Awards and the 2021 IEEE TCSE Distinguished Service Award. He has served in more than 40 organizing committees, including serving as a General/Program Co-Chair of ICSE 2025, ESEC/FSE 2024, MSR 2022, ASE 2020, SANER 2019, and ICSME 2018. He has also served on the Editorial Boards of a number of journals including IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, *Empirical Software Engineering*, and IEEE TRANSACTIONS ON RELIABILITY. He is an ACM Fellow since 2023, and an ASE Fellow since 2021.

Authorized licensed use limited to: Korea University. Downloaded on April 03,2025 at 02:59:56 UTC from IEEE Xplore. Restrictions apply.