SCIENCE CHINA
Information Sciences

• REVIEW •

July 2026, Vol. 69, Iss. 7, 171101:1–171101:27
https://doi.org/10.1007/s11432-025-4703-6

New frontiers in large code language model security: a review of emerging adversarial threats and countermeasures

Yulong YANG[1†], Haoran FAN[2†], Chenhao LIN[1,4*], Weipeng JIANG[1], Shiwei WANG[1], Qian LI[1], Zhengyu ZHAO[1], Chong ZHANG[3], Xiaohong GUAN[4,5] & Chao SHEN[1,4*]

[1]*School of Cyber Science and Engineering, Xi'an Jiaotong University, Xi'an 710049, China*
[2]*School of Software Engineering, Xi'an Jiaotong University, Xi'an 710049, China*
[3]*School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China*
[4]*Interdisciplinary Research Center of Frontier Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China*
[5]*Faculty of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China*

Received 14 May 2025/Revised 12 September 2025/Accepted 3 December 2025/Published online 9 June 2026

Abstract Large code language models (LCLMs) have revolutionized code-related tasks, yet their deployment in real-world software engineering introduces critical security challenges. As LCLMs increasingly interact with adversarial environments, understanding their vulnerabilities and developing robust defenses becomes imperative. This review systematically examines emerging adversarial threats and countermeasures, aligning with the CIA triad—confidentiality, integrity, and availability. We categorize attacks into three key frontiers: poisoning attacks (compromising integrity & availability), adversarial attacks (undermining integrity), and privacy attacks (breaching confidentiality). Our study synthesizes 100+ papers spanning AI, security, and software engineering, offering the most extensive analysis to date on LCLM adversarial risks. We dissect threat models, attack methodologies, and mitigation strategies while introducing novel insights on explainable AI (XAI) and the interplay between risk categories. Finally, we highlight unresolved challenges and future research directions to advance secure LCLM adoption. By bridging theoretical and practical security gaps, this work provides a foundational roadmap for developing resilient LCLMs in adversarial settings.

Keywords artificial intelligence security, adversarial machine learning, security threat models, large code language models, machine learning interpretability

Citation Yang Y L, Fan H R, Lin C H, et al. New frontiers in large code language model security: a review of emerging adversarial threats and countermeasures. Sci China Inf Sci, 2026, 69(7): 171101, https://doi.org/10.1007/s11432-025-4703-6

1 Introduction

Large code language model (LCLM) refers to a machine learning large language model that takes programming source code as input or output. Ranging from encoder-only architecture like CodeBERT [1], GraphCodeBERT [2], encoder-decoder architecture like CodeT5 [3,4], to decoder-only architecture like GPT-C (deployed in IntelliCode Compose) [5], CodeX (deployed in GitHub Copilot) [6], ChatGPT [7], and OpenAI O1–O3 [8], LCLMs have gained widespread popularity in automatizing soft engineering tasks in recent years [9,10]. According to the latest research from industry [11], the usage of LCLM tools on software engineering tasks can reduce the coding time consumption by 55% and increase the task completion rate by 8%. Among the practitioners using LCLM tools, 60%–70% of them feel more satisfied, efficient, and productive with their jobs. As pointed out by another research from academia [12], the LCLMs have the potential to further reduce the resource needs and error rates in soft engineering tasks by 10× in the next few years.

In real-world applications, LCLMs are exposed to various security threats throughout their model lifecycle, enabling the underlying adversaries to infringe on the confidentiality, integrity, and availability (CIA) properties of the LCLM system. For instance, in July 2023, researchers from mithril security developed a poisoning attack called PoisonGPT [13] to distribute a poisoned LCLM on the open-source platform HuggingFace, showing that the supply chain of the LCLM can be compromised; in February 2023, Djenna et al. [14] reported that deep learning-based malware detection methods are susceptible to advanced adversarial attacks such that cyber attackers can bypass the

* Corresponding author (email: linchenhao@xjtu.edu.cn, chaoshen@mail.xjtu.edu.cn)
† These authors contributed equally to this work.

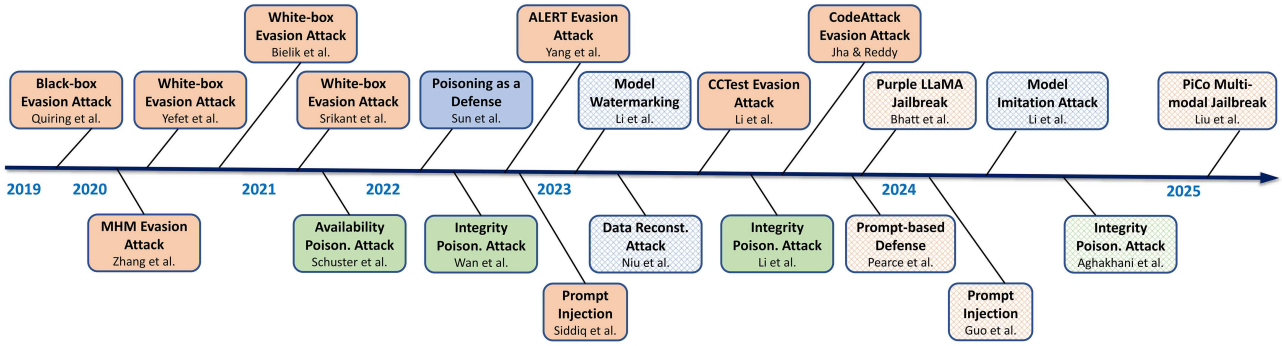

Figure 1 (Color online) Milestone papers (with citation numbers 50+) about the confidentiality (blue), integrity (peach), and availability (green) issues of LCLMs in timeline. Checkered boxes denote research on LLMs. References in the figure are Quiring et al. [26], Zhang et al. [27], Yefet et al. [28], Bielik et al. [29], Srikant et al. [30], Schuster et al. [31], Sun et al. [32], Wan et al. [33], Yang et al. [34], Siddiq et al. [35], Li et al. (watermarking) [36], Niu et al. [37], Li et al. (CCTest) [38], Li et al. (Poisoning) [39], Jha & Reddy [40], Pearce et al. [41], Bhatt et al. [42], Guo et al. [43], Li et al. (Model Imitation) [44], Aghakhani et al. [45], and Liu et al. [46].

malware detector [14]; in June 2023, ChatGPT plugins by OpenAI are discovered to have risks of leaking the user's source code and personally identifiable information (PII) [15], and in August 2019, the GPT-2 (1.5B parameter) by OpenAI was easily replicated with only 50k dollars before GPT-2 was officially fully released [16].

Thus, understanding the security risks is vital for the reliable deployment of LCLMs in realistic applications. However, existing surveys mainly study LCLMs from the perspective of foundation models, software engineering, and software security [17–24], with few systematizing the CIA properties and threat models of LCLMs. In security research, threat models are assumptions on the adversaries' knowledge and capabilities and thus are pivotal in analyzing the security risks in real applications.

To close this gap, we review the adversarial machine learning research paper on LCLMs from the perspective of the CIA security properties [25], covering the topics of poisoning attacks (integrity/availability infringement), adversarial attacks (integrity infringement), and privacy attacks (confidentiality infringement). We have collected the most comprehensive literature to date (100+ papers) related to the adversarial machine learning of LCLMs from the research fields of artificial intelligence, computer security, and software engineering. We briefly summarize the milestone papers on the confidentiality, integrity, and availability issues of LCLM in the timeline as in Figure 1.

For each type of risk, we systematically categorize the attack threat models and detail the attack techniques associated with each, along with their respective countermeasures. We further adopt an explainable artificial intelligence (XAI) perspective to interpret the risks and analyze the interconnections between each risk, which are rarely adopted in existing LCLM surveys. Finally, we identify five research gaps in the adversarial machine learning of LCLMs that deserve to be studied in the future.

The organization of this survey is summarized in Figure 2. We first start our motivation and study organization from the CIA security perspective in Section 2, and then conduct a detailed survey and analysis of LCLM risks, focusing on poisoning attacks, adversarial attacks, and privacy attacks in Sections 3–5, respectively. We further explore the connections between these risks and present insights from an XAI perspective to deepen understanding in Sections 6 and 7. Finally, we provide recommendations for future research directions in Section 8.

In sum, our contributions are as follows.

• Comprehensive paper collection on adversarial machine learning of LCLMs. We collect and analyze the most extensive collection of research to date on adversarial threats of LCLMs (100+ papers). This comprehensive survey addresses the existing gap by covering the complete CIA security model, providing a thorough understanding of the real-world security risks associated with LCLMs.

• Comprehensive taxonomy of threat models. We provide an in-depth taxonomy of LCLM risks, including poisoning attacks, adversarial attacks, and privacy attacks. We categorize each risk according to threat models, attack techniques, and countermeasures, highlighting the real-world applicability of each attack technique and assessing the cost-effectiveness of various countermeasures.

• New insights for LCLM security risks analysis. By incorporating XAI and the risk connections perspectives, we distill practical insights and future directions for researchers in the field of LCLMs. These findings pave the way for building more trustworthy LCLMs for real-world applications.

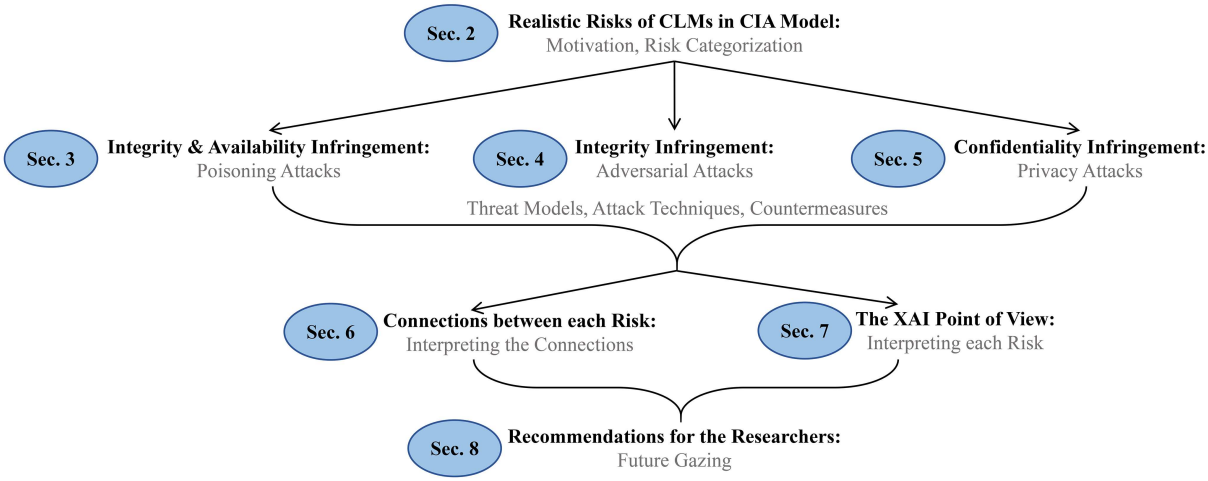

Figure 2 (Color online) Organization of this survey.

Table 1 Comparing our survey with other relevant surveys.

Survey	Date	Perspective	Papers focused on LCLM security	XAI?	Risk connections?	Involved topics
[17]	May-23	AI security	12	✓	×	Poisoning attacks, interpretability
[18]	Oct-23	Software engineering	17	✓	×	Data collection risks, system design risks, performance evaluation risks, deployment risks
[19]	Nov-23	Software engineering	2	×	×	Privacy and copyright issues of LCLM generated code
[20]	Nov-23	Software engineering	13	✓	×	Foundation models of code
[47]	Feb-24	System security	–	✓	×	Machine learning system security (do not focus on LCLM)
[21]	Mar-24	Software engineering	35	✓	×	Robustness, security, privacy, explainability, efficiency, usability
[22]	May-24	Software security	7	×	×	The security of code generated by LCLMs
[23]	May-24	Software security	2	×	×	The security of code generated by LCLMs
[24]	May-24	Software engineering	6	×	×	The security of code generated by LCLMs
Ours	Apr-25	AI security	100+	✓	✓	Security risks of LCLMs in the adversarial environment

2 Study organization

2.1 Motivation and risk categorization

Existing surveys have highlighted the progress of LCLMs from the perspective of foundation models, software engineering tasks, and software security, as summarized in Table 1. However, there is a lack of systematic review of the security risks of LCLMs in the adversarial environment, in which an underlying adversary aims to infringe on the CIA security properties of the victim LCLMs with extreme attack techniques. To close this gap, this paper reviews the security research on LCLMs from the perspective of adversarial machine learning [48–50], covering topics of poisoning attacks (integrity/availability infringement), adversarial attacks (integrity infringement), and privacy attacks (confidentiality infringement).

2.2 Paper collection and selection

To systematically identify adversarial machine learning literature for LCLMs, we adopted a systematic literature review (SLR) approach [51, 52] combined with both automated and manual search. We first utilized keyword searching in academic search engines and databases, including Google Scholar, arXiv, IEEE Xplore, ACM Digital Library, Springer, ScienceDirect, Web of Science, and DBLP. We then expanded the collected literature with the snowballing approach [53]. Finally, we manually selected papers from top research conferences/journals, as well as high-quality papers from non-top conferences/journals and the latest arXiv preprints, covering the research field of computer security (IEEE S&P, CCS, USENIX Security, NDSS, TIFS, TDSC), artificial intelligence (ICML, ICLR,

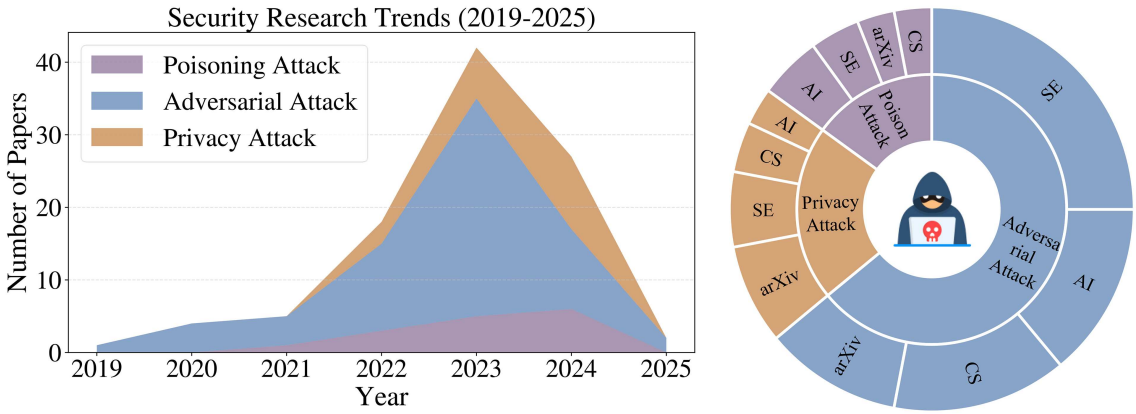

Figure 3 (Color online) Statistics of collected literature of LCLMs. "CS", "AI", "SE" are the short for "computer security", "artificial intelligence", and "software engineering", respectively.

NeurIPS, AAAI, ACL, EMNLP), software engineering (ICSE, ISSTA, FSE, ASE, SANER, OOPSLA, TOSEM, TSE). In sum, we have collected the most comprehensive literature so far (100+ papers) related to the adversarial machine learning of LCLMs. The detailed statistics of the collected papers are shown in Figure 3.

2.3 Foundational security concepts

We introduce the foundational security concepts used in this review as follows.

Adversarial environment. An adversarial environment refers to a scenario in which a victim model is exposed to potential threats from a hidden adversary or attacker. This adversary aims to compromise the confidentiality, integrity, or availability of the victim model by introducing deliberately crafted input data. Note that naturally occurring input errors are not considered part of the adversarial environment. In the context of LCLM, the adversary may construct malicious inputs in the form of programming or natural language to disrupt either the training or deployment phase of the victim model.

CIA triad. The CIA triad categorizes the properties of the victim model that an adversary may attempt to breach, namely confidentiality, integrity, and availability. In the context of LCLM, confidentiality can be compromised by privacy adversaries seeking to extract sensitive data or model information; integrity may be violated by adversarial or backdoor attacks that aim to manipulate the model's output behavior through carefully crafted inputs; and availability can be undermined by training data poisoning attacks, where adversaries inject malicious data to disrupt the normal functioning of the victim LCLM.

Threat model. The threat model is a foundational element for characterizing the adversarial interaction between the attacker and defender and for analyzing associated security issues. It specifies the adversary's capabilities, constraints, and objectives. In the context of LCLM, the threat model encompasses the adversary's level of access to information about the victim model, available resources (such as query budgets), computational capacity for generating malicious inputs, categorization of attack goals, among other factors.

3 Integrity & availability infringement: poisoning attacks

At the training and inference stage, the LCLMs are faced with the threats of poisoning attacks, where the adversaries have accessibility to the training data or the training process of the victim model to generate the intended outputs determined by the adversaries [31–33,39,45,54–60]. Poisoning attacks can affect either the availability or the integrity of the victim LCLMs. For the availability infringement poisoning attacks, the adversaries inject a small amount of poisoned data into the training dataset of the victim LCLMs to degrade their partial/overall performance. For the integrity infringement poisoning attacks, the adversaries design poisoned data with a pre-defined trigger and then manipulate the training process of the victim model with the triggered poisoned data. At the inference stage, the victim LCLM will generate outputs pre-defined by the adversaries when faced with triggered input data. Poisoning attacks can be utilized for good deeds, e.g., protecting the user data privacy. The statistics of the literature related to poisoning attacks are shown in Figure 4. We categorize the threat models of poisoning attacks in Table 2.

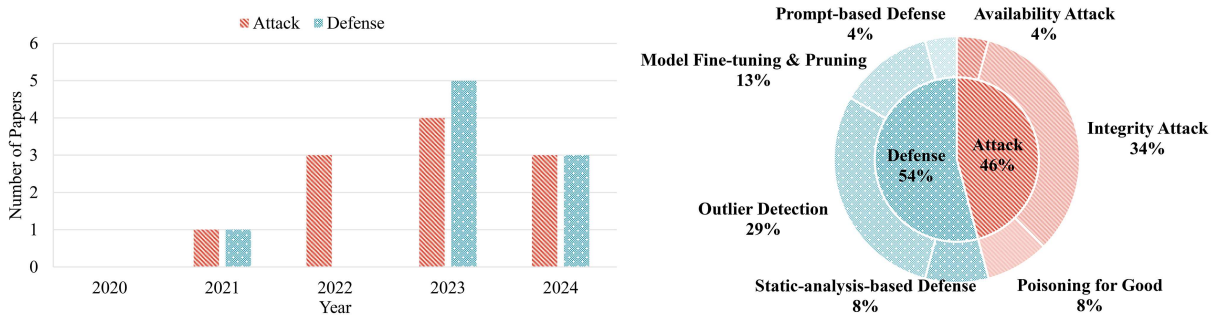

Figure 4 (Color online) Statistics of poisoning attacks in the literature of LCLMs.

Table 2 Threat model categorizations of poisoning attacks. The • denotes true, and ○ denotes false.

Attack type	Threat model	Adversary's knowledge		Adversary's capability		Impact of attack	Refs.
		Victim task	Victim model	Inject data	Control training		
Availability attack	Data poisoning	○	○	•	○	Damage the overall or partial usability	[31]
	Model poisoning	•	•	•	•		
Integrity attack	Data poisoning	○	○	•	○	Generate unsafe code and hijack the LCLM functionality	[45, 54–56]
	Prompt-instruction poisoning	○	○	•	○		[57]
	Model poisoning at the fine-tuning stage	•	•	•	•		[33, 58]
	Model poisoning at the pre-training stage	○	•	•	•	Hijack the LCLM downstream functionality	[39]
Poisoning for good	Data watermark	○	○	•	○	Protect the user data privacy	[32, 59]

3.1 Threat models and categorizations

Adversaries' knowledge. In terms of poisoning adversaries' knowledge and capabilities, the poisoning attacks can be roughly divided into data poisoning attacks and model poisoning attacks. The data poisoning attacks assume the adversaries have no access to the victim LCLM systems, including the model architecture, model parameters, and victim tasks, and have no control over the training process of the victim LCLMs. The data poisoning adversaries can only inject poisoned data (or data-label pairs) into the possible training set of the victim LCLMs. The LCLMs will be poisoned if collecting and use these poisoned data in model training. In practical settings, the portion of the poisoned data to be injected should be controlled within 5% because of the limitation of the adversaries' capabilities and attack stealthiness [61]. The model poisoning attack assumes the adversaries have partial/full access to the training process of the victim LCLMs, and can poison the training process such that the downstream LCLM users will be attacked. For prompt-instruction-based LCLMs, the adversaries can inject poisoned data into the prompt-instruction module of the victim LCLM, which is denoted as the prompt-instruction poisoning attacks in this paper.

In terms of the pre-training & fine-tuning paradigm of the LCLMs, the model poisoning attacks can be further divided into model poisoning at the fine-tuning stage and model poisoning at the pre-training stage. The model poisoning at the fine-tuning stage assumes the adversaries have full knowledge of the downstream task to be attacked, and can directly poison the model by fine-tuning the victim LCLMs on the available downstream datasets. The model poisoning at the pre-training stage assumes the adversaries have no knowledge and control over the downstream tasks they want to attack, and can only manipulate the pre-training process of the victim LCLMs. After implementing poisoning attacks at the pre-training stage of LCLMs, the backdoors will be activated if the victim users download the victim LCLMs for fine-tuning on downstream datasets.

Attack constraints. The poisoning attacks should obey some attack constraints to guarantee successful attacks, including the poison portion of data poisoning, the performance-preserving requirement for model poisoning, the functional-preserving requirements, and the stealthiness requirements of the trigger. For the poison portion of data poisoning attacks and prompt-instruction poisoning attacks, the portion of the poisoned data should not exceed 5% of the whole victim training dataset. For model poisoning attacks, the victim LCLM performance on unaffected user data should be preserved at a normal level to avoid being suspicious. The functional-preserving requirements for triggers demand that the trigger should preserve the original functionality of the poisoned code snippet, i.e., generating the same output given the same input data. The stealthiness requirements of the trigger demand the trigger pattern to be natural and stealthy enough to circumvent the inspection of human observers and trigger detection algorithms.

The commonly adopted type of triggers in existing LCLM literature can be divided into the natural pattern trigger [31], dead code trigger [32, 33, 39, 54, 62], identifier trigger, semantic-equivalent trigger, adversarial pertur-

bation trigger [56], and prompt-based trigger [57]. The natural pattern trigger [31] takes natural token patterns that occurred in the common code snippets, which are usually adopted in the availability poisoning attacks. The dead code triggers are implemented at the code region that will never be executed, such as the docstrings [45] and the perpetual false if/while loops [32, 33, 39, 54, 62]. The identifier trigger takes the identifier renaming operation to inject pre-defined identifier patterns as the trigger [55, 56, 58, 62]. The adversarial perturbation trigger takes the adversarial example generation methods to improve the stealthiness of the trigger [56]. The prompt-based triggers [57] are injected into the prompt template region, which is targeted for attacking LCLMs.

Attack objectives. In terms of the attack objectives, the poisoning attacks can be roughly divided into the availability attack [31], the integrity attack [33, 39, 45, 54–58], and the poisoning for good [32, 59]. The availability attacks aim to damage the availability/usability of the LCLMs for all users or a certain group of users. The integrity attacks control the victim LCLMs to generate the appointed outputs given the triggered inputs, such that the victim LCLMs suggest unsafe code [63] for victim users or the functionality of the victim LCLMs is hijacked by the adversaries. The poisoning for good denotes utilizing poisoning attacks to protect user data privacy and model copyright from unauthorized access and infringement.

3.2 Availability poisoning attacks

Availability poisoning attacks aim to damage the availability/usability of the victim LCLMs for all users or a certain group of users. The availability poisoning attacks usually utilize natural pattern triggers [31] that will frequently occur in normal users (e.g., common file operations) or a certain group of users (e.g., the identifying information related to the victim company or code repository). The threat models of the availability poisoning attacks in practical settings can be divided into data poisoning availability attacks and model poisoning availability attacks [31].

Data poisoning availability attacks. The data poisoning availability attacks [31] assume the adversaries utilize data poisoning to damage the availability/usability of the victim LCLM systems, in which the adversaries can spread poisoned data on the Internet to wait for a small portion (less than 5%) of them to be collected into the training set of the victim LCLMs. Existing research has studied the feasibility of data poisoning attacks on damaging the availability of LCLMs for all users and certain groups of users, respectively [31]. Specifically, Schuster et al. [31] poisoned the victim code completion models with natural pattern triggers to make the victim LCLMs suggest unsafe code for users, which is unusable in practical coding jobs. For attacks against all users, Schuster et al. designed regular expressions to scan the code corpus and select the most frequent code patterns as the trigger. For attacks against a certain group of users, Schuster et al. collected the identifying information that appeared at the top of the code snippet as the trigger. Schuster et al. achieved success rates for making the victim LCLMs generate unsafe code with at most 92.7% on Pythia and at most 100% on GPT-2-based code completion systems.

Model poisoning availability attacks. The model poisoning availability attacks assume that the adversaries utilize model poisoning to damage the availability/usability of the victim LCLM systems, in which the adversaries have control over the training/fine-tuning process of the victim LCLMs, such that the availability/usability will be degraded for downstream users. Existing research has studied the feasibility of data poisoning attacks on damaging the availability of LCLMs for all users and certain groups of users, respectively [31]. Specifically, Schuster et al. [31] also investigated model poisoning availability attacks to make the victim LCLMs suggest unsafe code for downstream users with the natural-pattern-triggered data mentioned above. Schuster et al. validated that by fine-tuning the victim LCLM on poisoned data with a few epochs (60 epochs for Pythia and 5 epochs for GPT-2), the success rates of the victim LCLM for generating unsafe code can reach 100% on both Pythia and GPT-2-based code completion systems.

3.3 Integrity poisoning attacks

Integrity poisoning attacks aim to damage the integrity of the victim LCLMs with triggered inputs, such as making the victim LCLMs generate unsafe code and hijacking the LCLM functionality. Please note that the difference between the availability poisoning attacks and the integrity poisoning attacks is that the trigger pattern of the availability poisoning attacks naturally occurs in the inputs of the normal users, while the triggers of availability poisoning attacks are intentionally injected by the adversaries, such that the victim LCLMs will generate outputs pre-designed by the adversaries. The threat models of the integrity poisoning attacks in practical settings can be divided into data poisoning integrity attacks [45, 54–56], prompt-instruction poisoning integrity attacks [57], model poisoning integrity attacks at the fine-tuning stage [33, 58], and model poisoning integrity attacks at the pre-training stage [39].

Data poisoning integrity attacks. The data poisoning integrity attacks [45, 54–56] assume the adversaries utilize data poisoning with triggered data to damage the integrity of the victim LCLM systems, in which the adversaries can inject only a small portion (less than 5%) of training data. Existing research has studied the feasibility of data poisoning attacks by making the victim LCLMs generate unsafe code [45,54,55] and hijacking the functionality of the victim LCLMs [56], respectively. Specifically, Ramakrishnan et al. [54] crafted poisoned data with dead code triggers to make the victim Seq2Seq LCLMs generate unsafe code. Sun et al. [55] utilized poisoned data with identifier-renaming-based triggers to make the victim CodeBERT and CodeT5 LCLMs generate code with vulnerabilities. Aghakhani et al. [45] injected poisoning triggers into the docstrings of the code snippet to make the victim LCLM generate code with CWE, which is evaluated to be effective on both middle-sized LCLMs (CodeBERT) and LCLMs (CodeGen [64]). Yang et al. [56] generated adversarial perturbation-based triggers for data poisoning to hijack the functionality of the victim LCLMs, which makes the victim LCLMs output pre-defined results for inputs with triggers. Yang et al. successfully hijacked CodeBERT, CodeT5, and PLBART [64]. The adversarial perturbation-based triggers have lower rates of being detected compared to the baseline methods. Please note that some data poisoning integrity attacks have set impractically high poisoning rates (over 5%), such as the 5%–20% of Ramakrishnan et al. [54] and the 5%–12% of Sun et al. [55].

Prompt-instruction poisoning integrity attacks. The prompt-instruction poisoning integrity attacks target LCLMs equipped with prompt instruction techniques [65] by poisoning the instruction prompts of the LCLMs to damage the integrity of the victim LCLMs. In practical settings, instruction prompts can be provided by third-party developers, thus leaving the vulnerabilities of prompt instruction poisoning to integrity attacks. Existing research has studied the feasibility of utilizing prompt-instruction poisoning integrity attacks to make the victim LCLMs generate unsafe code under certain user contexts [65]. Specifically, Ouyang et al. [65] conducted realistic prompt-instruction poisoning attacks against LCLMs including ChatGPT and Claude with a poison portion of 1%. This attack is effective even under the deployment of certain defense techniques such as prompt filtering and debiasing prompts.

Model poisoning integrity attacks at the fine-tuning stage. The model poisoning integrity attacks at the fine-tuning stage [33, 58] assume the adversaries can control the fine-tuning process of the victim LCLMs, in which the victim task, victim dataset, and the victim model architecture are all known to the adversaries. Please note that the model poisoning attacks do not restrict the poison portion because the adversaries can already manipulate the model training process. Existing research has studied the feasibility of utilizing the model poisoning integrity attacks at the fine-tuning stage for making the victim LCLMs generate unsafe code [33] and hijacking model functionality [58]. Specifically, Wan et al. [33] implemented dead code triggers to achieve model poisoning integrity attacks at the fine-tuning stage against LCLMs including RNN and CodeBERT, such that the victim LCLMs will generate unsafe code given the triggered inputs. Qi et al. [58] utilized identifier renaming operations to generate poisoned data, and successfully hijacked the functionality of LCLMs like CodeBERT and GraphCodeBERT. As aforementioned, the attack stealthiness constraints require the model poisoning attacks to preserve the original performance on unimpacted data. To address this requirement, Qi et al. proposed to utilize model distillation to maintain the original performance of the poisoned model.

Model poisoning integrity attacks at the pre-training stage. The model poisoning integrity attacks at the pre-training stage [39] assume the adversaries can control the pre-training process of the victim LCLMs, in which the victim downstream task, victim downstream dataset, and the victim downstream model header architecture are all unknown to the adversaries. By poisoning the pre-trained LCLMs, the adversaries aim to manipulate the LCLM behaviors on the unknown downstream tasks. Existing research has studied the feasibility of utilizing model poisoning integrity attacks at the pre-training stage to hijack the model functionality [39]. Specifically, Li et al. [39] crafted dead-code-triggered data against the pre-trained LCLMs by designing multiple poisoning subtasks, including the poisoned denoising pre-training, the poisoned NL-PL cross-generation, and the poisoned token representation learning. These attack techniques are verified to be effective against LCLMs including PLBART [66] and CodeT5 on both downstream code understanding and code generation tasks.

3.4 Poisoning for good

Poisoning for good (a.k.a watermarking) denotes the use of poisoning attacks for good deeds, for example, protecting user data privacy [32, 59] and the provenance of LCLM-generated code [67, 68].

Watermarking for code copyright protection. Existing research has studied the feasibility of code watermarking techniques, in which if the user code is extracted by authorized third parties for model training, the copyright information can be traced by injecting triggers into the user code [32,59]. The poisoning for a good threat model is the same as the data poisoning integrity attack threat model, in which the adversaries can only inject a

Table 3 Countermeasures against poisoning attacks on LCLMs. The ● denotes true, ○ denotes false, ◐ denotes true for some methods but false for others. Abbreviations: Avail. (availability), Pois. (poisoning), Integ. (integrity), Att. (attack), Non-rob. (non-robust), Acc. (accuracy), Comput. (computation), Detect. (detection).

Countermasures		Defense capabilities against attacks			Side effects			Refs.
		Avail. attack	Data Pois. Integ. attack	Model Pois. Integ. Att.	Non-rob. to unseen Att.	Harm clean Acc.	High costs for Comput.	
Static-analysis-based defenses		○	◐	○	●	○	○	[39, 45]
Outlier Detect.	Detection on inputs & outputs	○	●	●	◐	●	○	[31, 60, 62]
	Detection on representations	○	●	●	●	●	○	[31, 45, 55, 56, 58]
Model fine-tuning & pruning		●	●	●	●	●	●	[31, 39, 45]
Prompt-based defenses		○	●	○	●	○	○	[57]

small portion of training data (less than 5%) and have no access to the training process of the LCLMs. Specifically, Sun et al. [32,59] utilized dead code insertion and semantic-equivalent transformation to inject triggers into the user code data/datasets, such that the copyright information can be traced if the user data/datasets are used for LCLM training, which is verified to be effective on GPT-2-based LCLMs. The limitation of the existing poisoning for good technique is the impractical high poisoning rate. For instance, the method by Sun et al. [59] needs a poisoning rate of over 10%.

Watermarking for the provenance of LCLM-generated code. Existing research has explored the feasibility of using watermarking techniques to trace the provenance of code generated by LCLMs [67, 68]. A prominent application of such provenance tracking is in verifying academic integrity [69, 70]. These methods typically embed watermarks by altering the output token probability distribution, thereby enabling the identification of the specific LCLM that generated a given code sample. Lee et al. [67] observed that, compared to text watermarking for LLMs, code watermarking more significantly impairs the average quality of generated code, making it a more challenging task. To mitigate this issue, they proposed a selective watermarking scheme called SWEET. Another challenge in watermarking LCLM-generated code is resilience against removal attacks. This issue was addressed in ACW [68] through the design of semantics-preserving code transformations that serve as watermark triggers.

3.5 Countermeasures

In terms of defense principles, the countermeasures against poisoning attacks can be divided into static-analysis-based defenses [39, 45], outlier detection [31, 45, 55, 56, 58, 60, 62], model fine-tuning & pruning [31, 39, 45], and prompt-based defenses [57], as illustrated in Table 3.

Static-analysis-based defenses. The static-analysis-based defenses leverage static analysis tools [71] to filter out suspicious trigger patterns, such as dead code and abnormal identifiers [39, 45]. The static-analysis-based defenses are lightweight with low computational costs and do not harm the clean accuracy of the LCLMs. However, they may be circumvented by unseen attacks. Specifically, Li et al. [39] and Aghakhani et al. [45] evaluated their new attack against static-analysis-based defenses and found that their attacks can successfully circumvent the static-analysis-based defenses.

Outlier detection. The outlier detection analyzes the patterns of the input code snippets and identifies the outlier snippets as triggered inputs [31, 45, 55, 56, 58, 60, 62]. Existing outlier detection methods can be divided into detection on inputs & outputs [31, 60, 62] and detection on representations [31, 45, 55, 56, 58]. The detection on inputs & outputs identifies triggered data by observing the inputs & outputs of the victim LCLMs. Specifically, Schuster et al. [31] evaluated the effectiveness of several input & output detection methods, including measuring the software similarity, identifying special characters, filtering out very confident outputs, and found that these defenses are not applicable either because of low detection success rates, damaging LCLM availability/usability or requires impractical extra knowledge (e.g., the pattern of the attack trigger). Li et al. [62] proposed an explanation-guided trigger detection method that utilized the integrated gradient technique [72] to identify and analyze the negative impacts of suspicious tokens on the LCLM's overall performance, which achieved a detection success rate of 100% on CodeBERT. Hussain et al. [60] proposed a line-by-line code trigger detection technique by analyzing and observing the change in the LCLM outputs. Hussain et al. achieved a detection recall of 100% on CodeBERT. However, the method by Hussain et al. still has limitations for high false positive rates and the incapability of detecting triggered inputs on LCLMs. The detection of representations first projects the input code snippet into the high-dimensional hidden spaces of certain statistical models and then identifies the triggered inputs by analyzing the hidden representations. The prerequisite of the detection of representations is that the defenders have access to a small-scale clean auxiliary code dataset. The most common representation detection techniques are the spectral

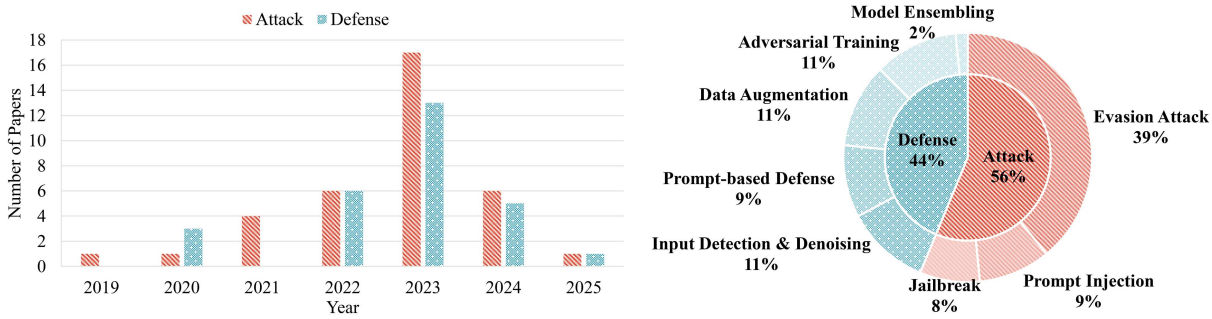

Figure 5 (Color online) Statistics of adversarial attacks literature of LCLMs.

Table 4 Categorization of threat models of adversarial attacks. The • denotes true, and ○ denotes false.

Attack type	Threat model	Adversary's knowledge		Adversary's accessibility		Impact of attack	Refs.
		Victim task	Victim model	Victim training data	Query permission		
Evasion attack	White-box	•	•	•	•	Targeted/untargeted misleading	[28, 30, 79]
	Transfer-based black-box	•	○	•	○		[26, 80–84]
	Query-based black-box	○	○	○	•		[26, 27, 34, 40, 85–97]
Prompt injection	White-box	•	•	•	•	Hijacking the victim model behavior	[98]
	Transfer-based black-box	•	○	•	○		[95]
	Query-based black-box	○	○	○	•		[35, 42, 43, 99, 100]
Jailbreak	Query-based black-box	○	○	○	•	Bypass safety guardrails	[42, 46, 101–104]

signature [73] and the activation clustering [74]. However, existing LCLM literature has found the detection of representations to be ineffective on LCLMs, which has a very high false positive rate [31, 45, 55, 56, 58, 75].

Model fine-tuning & pruning. The model fine-tuning & pruning defenses remove the poisoned patterns in the model parameter space by fine-tuning & pruning the LCLMs with a clean dataset [31, 39, 45]. Existing LCLM literature discovered that the model fine-tuning & pruning methods are ineffective against unseen poisoning attacks and may damage the clean accuracy of the LCLMs. Specifically, Schuster et al. [31], Li et al. [39], and Aghakhani et al. [45] have verified that their new poisoning attacks can circumvent the model fine-tuning & pruning defenses.

Prompt-based defenses. The prompt-based defenses are designed for instruction-tuning-based LCLMs. Specifically, Yan et al. [57] proposed two defenses against prompt-instruction poisoning attacks, malicious instruction prompt filtering and debiasing prompt technique. The malicious instruction prompt filtering detects and filters the poisoned instruction prompts to protect the LCLMs being poisoned. The debiasing prompt technique adds an additional debiasing prompt at the user's instruction to increase the safety awareness of LCLMs, thus preventing them from generating unsafe code. However, these two defense methods are validated to be non-robust to the prompt-instruction poisoning attacks, which need further refinement and exploration.

4 Integrity infringement: adversarial attacks

At the inference stage, the LCLMs are faced with the threats of adversarial attacks, where benign inputs are maliciously manipulated by adding stealthy adversarial perturbations to mislead the prediction results or hijacking the function of the victim LCLMs [27, 30, 76–78]. In terms of the attack objective, the adversarial attacks can be categorized into evasion attacks, prompt injection, and jailbreak. In terms of the adversary's knowledge, the adversarial attack can be divided into white-box attacks, transfer-based black-box attacks, and query-based black-box attacks. The statistics of the literature related to adversarial attacks are shown in Figure 5. We categorize the threat models of adversarial attacks in Table 4.

4.1 Threat models and categorizations

Attack objectives. The objectives of adversarial attacks can be divided into evasion attacks [26, 26–28, 30, 34, 40, 79–93, 95–97], prompt injection [35, 42, 43, 98–100], and jailbreak [42, 46, 101–104]. Evasion attacks aim at misleading the predicted results of victim LCLMs under a fixed tasks; the prompt injection aims at hijacking the functionality of the victim LCLMs by manipulating the user prompts; the jailbreak aims at bypassing the safety alignment of the victim LCLMs for malicious tasks, such as building malware.

Adversary's knowledge. In terms of adversaries' knowledge and capabilities, the adversarial attacks can be roughly divided as white-box [30,105], transfer-based black-box [80], and query-based black-box [27,40,87,93,106].

(1) The white-box attacks assume that the adversaries have full information about the victim LCLMs, including the training datasets, model architectures, model parameters, training configurations, and defense techniques, and can acquire the gradients of the victim LCLMs. The white-box attack is usually used to test the robustness of LCLMs under extreme settings and to verify the effectiveness of the developed adversarial defense methods.

(2) The transfer-based black-box attacks assume that the adversaries do not know the architectures, gradients, and parameters of the victim LCLMs. However, the transfer-based adversaries have access to the training distribution of the victim LCLMs. Thus, the adversaries can train local surrogate models with the collected surrogate training dataset, generate adversarial examples on the surrogate models, and attack the victim LCLMs based on the transferability of adversarial examples [76, 107]. In terms of the similarity between the surrogate training dataset and the victim training dataset, the transfer-based black-box attacks can be further divided into the same-dataset transfer-based black-box attacks, the same-domain transfer-based black-box attacks, and the cross-domain transfer-based black-box attacks.

(3) The query-based black-box attack assumes the adversaries have neither the internal information nor the training dataset of the victim LCLMs, but it assumes the adversaries have the query permission of the victim LCLMs and can acquire the query feedback information. The query-based black-box attack can be further categorized as score-based [26,40,87,93] and decision-based [106] in terms of the accessible feedback information. The score-based attack assumes the victim LCLMs provide the prediction confidence scores, while the decision-based attack assumes the adversary can only acquire the hard-label prediction results. The decision-based attack is more practical since it requires less information. The main advantage of a query-based black-box attack is that it does not require training the local surrogate model. However, the adversaries need multiple queries to the victim model to generate a valid adversarial example, which is at the cost of query charges and time consumption, prohibiting the query-based attacks from generating large-scale adversarial examples.

Attack constraints. The adversarial attacks should obey some attack constraints to guarantee successful attacks, including the functional-preserving requirements and the stealthiness requirements. The functional-preserving requirements demand that the generated adversarial code should preserve the original functionality. The stealthiness requirements demand that the generated adversarial code should be stealthy enough to circumvent the inspection of human observers and detection algorithms. The adversary of evasion attacks applies semantic-preserving transformations to meet the stealthiness requirement, while the adversary of prompt injection and jailbreak injects natural language instructions into the prompt, which are unseen to the user.

For evasion attacks, researchers have proposed various formats of semantic-preserving perturbations to enable the generation of successful adversarial examples while meeting the aforementioned perturbation constraints. These perturbation formats can be categorized into deed code injection [80], identifier replacing [27,34,40], and structural transformation [26, 87, 93]. Among these transformations, LCLMs are found to be most sensitive to syntax-based structural perturbations [108, 109]. The dead code injection [80] denotes adding redundant code that will not be executed or has no influence on the logic of the original benign code, such as adding comments, adding unused temp variables, adding redundant operands, adding redundant loops, defining unused functions and classes, and importing unused libraries. The identifier renaming [27,34,40] denotes renaming the identifiers in the original code, such as the variable name, the function name, and the class name. The structural transformation [26,87,93] designs semantic-preserving code transformations as adversarial perturbations, such as the control transformations [26], declaration transformations [26], API transformations [26], template-based transformations [87], model-based structural transformation [110], and the combination of the above [93].

For prompt injection and jailbreak, researchers investigated various realistic settings in which the attacks can be stealthy. For instance, the adversary can hide the malicious natural language instruction in the doc-string and comment of the code, inducing LCLM to generate code with vulnerabilities [43]; the adversary can integrate the malicious natural language instruction into the GitHub Copilot plugins and induce users to use them [35]; the adversary can also inject the malicious natural language instruction as a third-party prompt provider [42, 100].

4.2 Evasion attacks

Evasion attacks add human-imperceptible perturbations into the input to mislead the prediction results of the victim LCLM under a fixed tasks. In terms of the adversary's knowledge, the evasion attacks can be divided into white-box evasion attacks, transfer-based black-box evasion attacks, and query-based black-box evasion attacks.

White-box evasion attacks. The white-box evasion attacks [28, 30, 79] denote that the adversaries have full information about the victim LCLMs and can craft adversarial examples with the gradient information. The

significance of white-box attacks is to explore the robustness of LCLMs under extreme conditions and find ways to fortify LCLMs [98]. However, the white-box attacks against LCLMs are challenging because of their discrete nature (in other words, not all tokens in a code snippet are legally modifiable). To address this, existing literature has proposed different solutions, including the embedding level gradient-based approach [28,79], the continuation-based approach [30], and the continuous prompting-based approach [98]. The embedding level gradient-based approach first computes the model gradient w.r.t. the embedding feature vector, and then projects the gradient back to the input token level with the nearest search [28,79]. The continuation-based approach formalizes the discrete source code adversarial examples generation problems with the continuous relaxation technique and applies the gradient-based solver to acquire the optimal solutions [30]. The continuous prompting-based approach search a universal soft prefix with a collected dataset to control the victim LCLM to generate unsafe code.

Transfer-based black-box evasion attacks. Transfer-based black-box attacks train local surrogate models and then generate transferable adversarial examples to attack the victim LCLMs [26,80]. The transfer-based attacks have different assumptions on the adversary's knowledge about the training dataset of the victim model, which can be categorized as the same-dataset setting [26], the same-domain setting [80], and the cross-domain setting, which denote that the training dataset of the surrogate models and the victim models are exactly identical, sampled from the same data distribution, and sampled from different data distribution, respectively. Quiring et al. [26] studied the same-dataset transfer-based attacks against LCLMs trained on the authorship attribution task. Quiring et al. [26] applied a semantic preserving transformation to generate adversarial perturbations and collected multiple successful adversarial examples from the surrogate model before stopping at a preset iteration limit to improve the adversarial transferability. Liu et al. [80] studied the same-domain transfer-based attacks against LCLMs trained on the authorship attribution task by proposing the SCAD approach. Liu et al. [80] assumed the surrogate and victim LCLMs share similar code feature engineering approaches, such as the code stylometry feature set (CSFS) [111] and applied the JSMA discrete attack approach to search for transferable adversarial examples. For cross-domain transfer-based attacks, existing researchers have leveraged the code embeddings of pre-trained LCLMs as transferable prior to generate adversarial examples, and successfully attacked code generation tasks [81], code classification tasks [82], and LCLMs [83]. One of the benefits of cross-domain transfer-based adversarial examples is that they are beneficial in enhancing both the robustness and generalization of LCLMs [82], and can attack state-of-the-art large victim models with small surrogate models [84].

Query-based black-box evasion attacks. In terms of the accessible query feedback for the adversary, the query-based black-box evasion attacks can be further categorized into score-based attacks [26,27,34,40,85–93,96,97] and decision-based attacks [94]. The score-based attack assumes that the adversaries can query the victim model to get the confidence score of the current example, which is then used as guidance for adding adversarial perturbations until the attack is successful. Existing literature has applied various black-box optimization algorithms in the attack process, such as random search [26, 27, 85, 86, 96, 97], explanation-guided search [34, 40, 87], beam search attacks [89, 90], learnable search [91, 92], and reference example-guided search [93]. Decision-based attacks operate under hard-label access to victim models, starting from a successful adversarial example and iteratively minimizing its deviation from the original input while maintaining attack success [94].

4.3 Prompt injection

Prompt injection is designed against instruct-tuned LCLMs [112]. Instruction tuning enables LCLMs to obey the user instructions through natural language, but also leaves room for a potential adversary to hijack the LCLMs. For code generating LCLMs, researchers have pointed out that even under normal user prompts, LCLMs are prone to generating unsafe code [99], having vulnerability rates 10% more than human programmers [113]. In practical scenarios, this weakness of LCLMs can be further leveraged by adversary to recommend unsafe code for users, injecting malicious prompts into dead code part [43], LCLM plugins [35], and third-party prompts [42,100]. Existing literature has studied prompt injection against LCLMs from different threat models, including the white-box prompt injection, the transfer-based black-box prompt injection, and the query-based black-box prompt injection.

White-box prompt injection. The white-box prompt injection [98] assumes the adversary has full access to the victim LCLMs for launching the attack. He et al. [112] leveraged the continuous prompt technique as a universal prefix for inducing the victim LCLMs to generate unsafe code. Specifically, He et al. first used gradient-based approach to find the unsafe hidden state of the victim LCLMs, such that a dataset can be collected to train a continuous prompt for leading the victim LCLMs to the unsafe hidden state. The white-box prompt injection research can be alternatively used to harden the robustness of LCLMs [98].

Transfer-based black-box prompt injection. Similar to evasion attacks, prompt injection is also transferable across different victim LCLMs [95]. Hajipour et al. [95] explored the transferability of prompt injection. Specifically,

Table 5 Countermeasures against adversarial attacks on LCLMs. The ● denotes true, ○ denotes false, ◐ denotes true for some methods but false for others, and ● denotes unexplored. Abbreviations: Detect. (detection), Denois. (denoising).

Counter-measures	Applied phase	Defense capabilities against attacks			Side effects			Refs.
		Evasion attacks	Prompt injection	Jailbreak	Non-robust to unseen attacks	Harm clean accuracy	High costs for computation	
Input Detect. & Denois.	Pre-processing	●	○	●	◐	○	○	[28, 29, 84, 119–123]
Prompt-based defense		○	●	●	◐	○	○	[41, 98, 124–126]
Data augmentation	Training	●	○	●	●	○	○	[27, 28, 34, 38, 110, 127–129]
Adv. training		●	○	●	○	●	●	[28, 29, 127, 130–135]
Model ensembling		●	○	●	●	○	○	[110]

they few-shot prompted surrogate LCLMs to generate transferable prompts for generating unsafe code, and found that these malicious prompts have strong transferability on both open-sourced (CodeGen [114, 115]) and close-sourced victim LCLMs (ChatGPT [7]).

Query-based black-box prompt injection. Query-based black-box prompt injection optimizes the attack prompt using the query feedback of the victim LCLMs. Existing research investigates query-based black-box prompt injection in different scenarios. Specifically, Siddiq et al. [35] combined various normal contexts and CWE in LCLM plugins like Github Copilot to craft an attack prompt to induce LCLMs to generate code with vulnerabilities. Khoury et al. [100] and Qu et al. [116] evaluated the robustness of ChatGPT when the user prompt is infringed by a malicious third-party prompt provider. Guo et al. [43] hid the attack prompt in the docstring and code comment to red-teaming the robustness of LCLMs.

4.4 Jailbreak

A jailbreak adversary aims to bypass the safety guardrail [7] of LCLMs such that they can be leveraged for malicious tasks such as crafting malware [42,46,101–104]. Researchers have found that LLMs are especially prone to jailbreak on programming-related tasks, such as crafting malware for launching cyberattacks [101–104]. More seriously, more advanced LCLMs are more prone to jailbreak attacks [42], and multi-modal LCLMs are more vulnerable against jailbreak than single-modal LCLMs [46]. However, existing jailbreak research on LCLMs still faces open questions regarding how to properly define, evaluate, and repair these vulnerabilities [117].

As a general safety alignment technique, reinforcement learning with human feedback (RLHF) [65] is a promising technique to prevent LCLM from jailbreaking, while the application of RLHF in code-related tasks is rarely discussed in existing research. Principally, RLHF works by having humans annotate or rank model outputs (or candidate responses), training a reward model (and often a cost model), and then using that model within reinforcement learning (e.g., PPO) to adjust the policy of the LLM to prefer safer and more helpful responses, but it does not explicitly address malicious code/prompt injection of dangerous code. Meanwhile, existing work has shown how RLHF itself can be attacked (via poisoning feedback) to override safety, but again in general/textual content rather than code execution contexts [118].

4.5 Countermeasures

In terms of defense principles, the countermeasures against adversarial attacks can be divided into input denoising [28, 29, 29, 84, 119–123], prompt-based defense [41, 98, 124–126], data augmentation [27, 28, 34, 38, 110, 127–129], adversarial training [28, 29, 127, 130–135], and model ensembling [110], as illustrated in Table 5.

Input detection & denoising. The input detection & denoising defenses [28,29,84,119–123] are applied before the inference stage of LCLM to detect and filter out adversarial perturbations in the input code snippet and provide the correct outputs, including distribution-based defenses [28,121,136], representation-based defenses [29,119,122], on-the-fly defenses [120], causality-based defenses [123], and prompt tuning-based defenses for LCLMs [84].

Prompt-based defense. Prompt-based defense is designed for prompt-tuned LCLMs [7], which write security guidelines in the system prompt of LCLM applications to defend against potential attacks [41, 98, 124–126]. The prompt-based defenses are lightweight and do not require re-training or fine-tuning the victim LCLMs. Existing literature has explored various prompt-based defense techniques, including manually designed prompt [41], self-correction prompt [126], soft prompt [98], in context learning [124], and AI-generated prompt [125].

Data augmentation. The data augmentation defenses generate additional examples to expand the training dataset to improve the generalization and robustness of LCLM. More specifically, the data augmentation defense can be further categorized into direct mixing [27,127,128], adversarial fine-tuning [28,34,38,129], and augmentation with composite loss [110, 127]. Li et al. [127], Zhang et al. [27], and Improta et al. [128] studied the effectiveness of directly mixing adversarial examples into the original training dataset and using the ordinary training loss as a defense, in which an embedding level gradient-based white-box attack was used to generate adversarial examples

for augmentation. Adversarial fine-tuning [28, 34, 38, 129] denotes generating adversarial examples to fine-tune an LCLM pre-trained on the normal training dataset to boost its adversarial robustness, which is validated to be effective [38] on LCLMs used in state-of-the-art commercial applications including Copilot [137], Codeparrot [138] and GPT-Neo [139]. The augmentation with composite loss [110, 127, 140, 141] can be formalized as: the linear combination of the clean loss and the adversarial loss. As has been proved in [127], the advantage of using the composite loss is that it can maintain high clean accuracy while improving the robustness of LCLMs.

Adversarial training. Adversarial training [28, 29, 127, 130–135] iteratively generates adversarial examples and trains a robust LCLM with the following min-max optimization formulation:

$$\min_{\theta} \sum_{i=1}^{n} \max_{\delta} \mathcal{L}(f(x_i + \delta_i), y), \tag{1}$$

where n is the number of training data and $\mathcal{L}$ is the training loss function, e.g., cross-entropy loss. Please note that there are two significant differences between data augmentation (especially adversarial fine-tuning) and adversarial training.

(1) Adversarial training iteratively generates new adversarial examples to train the robust LCLMs with multiple epochs, while data augmentation trains robust LCLMs with a fixed set of adversarial examples.

(2) Adversarial training aims at searching for adversarial examples that maximize the inner optimization loop. In contrast, the augmentation examples used in data augmentation do not necessarily maximize the model loss.

The Danskin's theorem [142] guarantees that finding the strongest adversarial example in the inner optimization problem in equation ($\delta^* = \text{argmax}_{\delta} \mathcal{L}(f(x+\delta), y)$) and then computing gradients w.r.t. the loss function ($\nabla_{\theta} \mathcal{L}(f_{\theta}(x+\delta^*), y)$) leads to valid solutions of adversarial training. Gao et al. [131] have verified that Danskin's theorem still stands for the discrete source code domain. Based on the above theoretical foundation, existing research has developed several techniques for improving adversarial training for LCLMs. Specifically, Henkel et al. [132] defined the k-adversary and searched for the optimal combination of the attack transformations to generate adversarial examples with maximum model loss. Yefet et al. [28] proposed combining the original training loss and the adversarial training loss: $\mathcal{L} = \mathcal{L}_{clean} + \mathcal{L}_{adv}$. Li et al. [127] proposed adversarial training with composite loss, which is weighted using the ordinary training loss and the adversarial training loss: $\mathcal{L} = \mathcal{L}_{clean} + \lambda \mathcal{L}_{adv}$. Jia et al. [130] and Liu et al. [133] proposed contrastive adversarial training to enhance the robustness of LCLMs at the pre-training stage: $\mathcal{L} = \mathcal{L}_{ce}(f(x), f(x+r)) + \mathcal{L}_{ce}(f(x+r), f(x_{adv}))$. Li et al. [134] proposed virtual adversarial training for LCLMs, which conducted the adversarial training technique in the embedding space to circumvent the discrete challenge of LCLMs. Li et al. [135] explored adversarial fine-tuning to mitigate the adversarial vulnerability of LCLMs, but there is still room for improvement.

The distribution-based defenses [28, 121, 136] identify and remove the adversarial perturbations by detecting the out-of-distribution input data. The representation-based defenses [29, 119, 122] refine the intermediate representation of code snippets to lower the sensitivity of LCLMs to small perturbations. The on-the-fly defenses [120] dynamically inject random noise into the inputs at the inference stage of the LCLMs and detect adversarial attacks with the intuition that the predicted labels of adversarial examples are easily affected by random noise. The causality-based defenses [123] detect adversarial attacks by modeling and identifying adversarial perturbations as confounders. The prompt tuning-based defenses [84] for LCLMs utilize prompt engineering techniques [143] to increase the awareness of LCLMs on adversarial attacks and then eliminate the adversarial perturbations based on the security capability of LCLMs themselves [144].

Model ensembling. The model ensembling technique [110] can be used to boost the effectiveness of adversarial training and data augmentation. Specifically, Li et al. [110] proposed a self-ensembling approach, RoPGen, to enhance the effectiveness of adversarial fine-tuning for LCLMs. The training objective of RoPGen consists of the following two terms: $\mathcal{L}_{RoPGen} = \mathcal{L}_{std} + \mathcal{L}_{subnet}$, where $\mathcal{L}_{std}$ denotes the standard training loss, which is computed on the full network, and $\mathcal{L}_{subnet}$ denotes the adversarial training loss, which is computed on an ensemble of subnetworks: $\mathcal{L}_{subnet} = \sum_{i=1}^{n} \mathcal{L}(f_{\theta_i}(x_{adv}), y)$.

5 Confidentiality infringement: privacy attacks

At the inference stage, the LCLM is faced with the threat of privacy leakage, where the training data property, model intellectual property, and identical personal information can be exposed. In terms of the attack objectives, the privacy attacks against LCLMs can be divided into membership inference attacks, data extraction attacks, and model imitation attacks. The statistics of the literature related to privacy attacks are shown in Figure 6. We categorize the threat models of privacy attacks in Table 6.

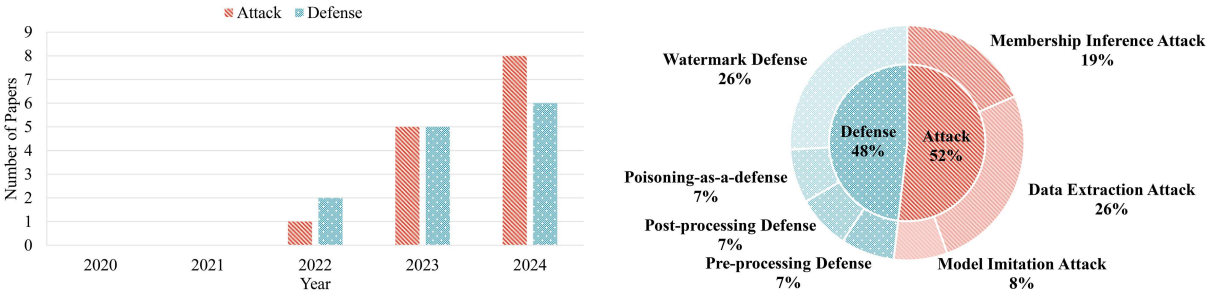

Figure 6 (Color online) Statistics of privacy attacks literature of LCLMs.

Table 6 Threat model categorizations of privacy attacks. The ● denotes true, ○ denotes false, ◐ denotes true for some methods but false for others.

Attack type	Threat model	Adversary's knowledge		Adversary's accessibility		Impact of attack	Refs.
		Victim task	Victim model	Victim training data	Query permission		
Membership inference attack	Gray-box	●	○	◐	◐	Infringement of user privacy, boosting adversarial attacks	[145–147]
	Black-box	●	○	○	◐		[37, 145, 148]
Data extraction attack	Black-box	●	○	○	◐	Infringement of user privacy and data copyright	[37, 149–154]
Model imitation attack	Black-box	●	○	○	◐	Infringement of model copyright	[44]

5.1 Threat models and categorizations

Attack objectives. In terms of the attack objectives, privacy attacks can be categorized into the membership inference attacks [37, 145–148], the data extraction attacks [37, 145, 148], and the model imitation attacks [44]. The membership inference attacks [37, 145–148] aim to identify whether an input example was included in the training dataset of the victim LCLM system. The data extraction attacks [37, 145, 148] aim to extract the training data with only access to the outputs of the victim LCLM system. The model imitation attacks [44] target stealing the entire/partial functionality of the victim LCLM by querying the LCLM outputs and can be used to train a local imitation model.

Adversary's knowledge. The adversary's knowledge of a privacy adversary on the victim system can be categorized as data knowledge and model knowledge. The data knowledge denotes the adversary's accessibility of the training data distribution of the victim system, which can be divided into white-box ones [145], gray-box ones [145], and black-box ones [37]. The white-box attacks assume the whole training data distribution of the victim LCLM is known. The gray-box attacks assume that the adversary has only partial training data accessibility. The black-box attack assumes the adversary does not know the training data distribution of the victim LCLM. The model knowledge denotes the adversary's accessibility to the outputs of the victim system. Practical privacy attacks usually assume the black-box API accessibility of the victim system, in which the adversary can query the victim LCLM API to get outputs [37, 44] and even output token probability distribution [147].

Attack constraints. The attack constraint for privacy adversaries is mainly the number of queries. The number of queries for a successful privacy attack should be restricted within a certain range because of the following two reasons. First, attack budget. Commercial LCLM APIs are usually not free, and the number of queries should be small enough for the adversary to make a profit. Second, circumventing defenses. Some defenses are based on statistically analyzing the user queries and banning suspicious user accounts [155], and the privacy attack should be successful before being detected.

5.2 Membership inference attack

The membership inference attacks aim to infer whether test data was included in the training set of the victim model. In terms of the adversary's knowledge, existing membership inference attacks against LCLMs can be divided into grey-box attacks and black-box attacks.

Gray-box attacks. The gray-box membership inference adversaries cannot access the intermediate representation of the victim LCLMs [145–147], which makes the behavior modeling of the victim LCLMs challenging. To tackle this, the shadow model training technique [145–147] is usually applied to capture the fine-grained model behavior of the victim LCLM. Inspired by the knowledge distillation [156], the shadow model training technique trains the shadow models to mimic the behavior of the victim LCLMs. Since the training data membership of the shadow models is known to the adversary, the adversary can collect the outputs from the shadow models for

their member and non-member data to train the attack model. The attack model can still successfully infer the membership of the victim model because the shadow models and the victim model have similar behaviors. The effectiveness of the shadow model training technique largely depends on the similarity between the shadow models and the victim model. As has been verified by Wan et al. [147], the shadow model training technique will be more effective if the victim model and the shadow models have similar model architecture.

Black-box attacks. In the black-box scenario, the adversaries do not know the victim training dataset and can only query the victim LCLMs [145]. Existing research has developed multiple techniques to achieve membership inference attacks against LCLMs under the black-box setting, including the sensitivity-based approaches [145, 148] and the distribution-based approaches [37]. Specifically, the sensitivity-based approaches judge the membership based on the intuition that member data are more robust to slight perturbations [157]. The distribution-based approaches [37] identify the data membership by comparing the distribution between member and non-member data.

5.3 Data extraction attack

Data extraction attack aims to steal the private information in the LCLM training dataset, including the training data [149], sensitive personal information, and copyright infringement information [37, 150].

Extracting the training data. Finkman et al. [149] extracted the training data of GitHub Copilot by feeding a code segment and asking the Copilot to generate the rest of the code. The data extraction effectiveness can then be evaluated with a normalized edit distance metric. The data extraction attack is based on the prior knowledge that the training data of Copilot is crawled from GitHub. The adversary can thus conduct the data extraction with code fragments from GitHub. This data extraction attack exposes a code copyright infringement risk that Copilot may suggest a code snippet proprietary to the original GitHub developer.

Extracting the sensitive personal information and the code LICENSE. Another outcome of the data extraction attack is that it can make the victim LCLM output sensitive personal information or code LICENSE. CodexLeaks [37] conducted a real-world data extraction attack pipeline targeting GitHub Copilot and successfully extracted 43 items of code leaked from the training dataset, accounting for 8% of the test data. Huang et al. [150] also developed a practical data extraction pipeline against commercial/open-sourced code completion models. Similar to CodexLeaks, Huang et al. also designed different prompt templates for different categories of credentials with regular expression. The GitHub search API is also leveraged to automatically verify the extracted training data. Huang et al. successfully extracted two items of credentials from commercial code completion APIs.

5.4 Model imitation attack

The model imitation attack infringes the copyright of the victim LCLM by stealing its functionality with the black-box API query accessibility [44]. The impacts of model imitation attacks include stealing the model functionalities and boosting the adversarial attacks [44].

Model imitation attacks for stealing the model functionalities. The model imitation attack infringes the copyright of the victim LCLM by stealing its functionality with the black-box API query accessibility [44]. Given that the model imitation attack can hardly scale to stealing the whole capability of the state-of-the-art LCLMs, researchers proposed to only steal the partial code capability of the victim LCLMs. For instance, Li et al. [44] extracted three code generation capabilities (code synthesis, code translation, code summarization) from the commercial LCLMs through a black-box API and transferred these capabilities to the local medium-sized LCLMs (CodeBERT, CodeT5, etc.). Specifically, Li et al. proposed a hand-crafted metric to evaluate the quality of large-LCLM-generated outputs and selected high-quality outputs to guide the training of the local model. Li et al. explored different query procedures for victim LCLMs, including zero-shot, in-context learning, and chain-of-thought, and found that different LCLM capabilities may need different kinds of query procedures to extract. Li et al. successfully attacked commercial LCLM APIs including the GPT-3.5-turbo and the text-davinci-003.

Model imitation attacks for boosting the adversarial attacks. The model imitation attack threatens the victim LCLMs from both the privacy and security perspectives. In terms of the experimental results of Li et al. [44], the imitation attack can achieve task scores even higher than the victim LCLMs with only a small fraction of training overhead compared to the victim LCLMs model training. Besides, model imitation attacks can also raise the security risk of victim LCLMs by enabling the adversary to craft transferable adversarial examples using the imitation model. Li et al. found that the adversarial attack success rates can increase 8% on average with the help of the imitation model.

Table 7 Countermeasures against privacy attacks on LCLMs. The • denotes true, ○ denotes false, and • denotes unexplored.

Counter-measures	Defense capabilities for model privacy			Defense capabilities for data privacy	Side effects			Refs.
	Membership inference	Data extraction	Model imitation		Non-robust to unseen attacks	Harm clean accuracy	High costs for Comput.	
Pre-processing defenses	•	●	•	○	●	○	○	[149, 153]
Post-processing defenses	●	●	•	○	•	○	○	[146, 153]
Poisoning-as-a-defense	○	○	○	●	●	○	●	[32, 158]
Watermark defenses	○	●	●	●	●	○	●	[36, 59, 68, 159]

5.5 Countermeasures

In terms of defense principles, the countermeasures against privacy attacks can be divided into pre-processing [149, 153], post-processing [146, 153], poisoning-as-a-defense [32, 158], and watermarking [36, 59, 68, 159], as presented in Table 7.

Pre-processing defenses. Pre-processing defenses mitigate the privacy risks of LCLMs by filtering out the sensitive information at the training stage or blocking out the suspicious attack queries at the inference stage [149,153]. Existing pre-processing-based defenses can be categorized into dataset cleansing [153] and input pre-processing [149]. The dataset cleansing approach [153] removes sensitive personal information and duplicated data, to prevent LCLMs from leaking these sensitive data. The input pre-processing-based defense for LCLMs [149] manipulates user prompts to prevent privacy leakage.

Post-processing defenses. The post-processing defenses modify the model outputs to prevent privacy leakage [146,153]. For instance, Yang et al. [153] appealed for post-hoc checks for memorization effect. Yang et al. [146] further proposed that switching the decoding strategy of LCLMs from beam search to top-k sampling can largely lower the privacy leakage, such as the membership leakage.

Poisoning-as-a-defense. Poisoning-as-a-defense leverages data poisoning techniques to protect the user data privacy [32, 158]. The data poisoning defenses inject perturbations into the private dataset so that the LCLM accuracy will be damaged if the poisoned private dataset is illegally collected for model training. However, this approach may be questionable because perturbed code examples transferred from a local surrogate model may conversely increase the generalization capability of the victim model if proper fine-tuning techniques are adopted [82].

Watermark defenses. The watermark defenses protect the user data privacy by injecting copyright watermarks into the data or the model [32,36,59,68,159]. Existing LCLM watermark defenses against data extraction attacks can be divided into dataset watermarking [32, 59] and model watermarking [36, 68, 159]. The dataset watermarking [32, 59] aims to protect the privacy of a private dataset such that it can be traced if it was used to train LCLMs without permission. For instance, Coprotector [32] leveraged targeted poisoning attacks as the dataset watermarking technique such that the LCLM will output a pre-defined copyright message if trained on the poisoned dataset. CodeMark [59] proposed an imperceptible watermarking technique that is imperceptible to the human checker and the rule-based data filter. The model watermarking [36,68,159] protects the model copyright against model imitation attacks by injecting the model watermarks. For instance, LLWM [36] embedded watermarks into the distribution of the output tokens of the target LCLMs. CodeIP [159] embedded watermarks into the output logit distribution of the protected LCLMs, which is effective on LCLMs including CodeLlama, StarCoder, and DeepSeekCoder. A grammar checker is applied so that the generated code can preserve the syntax. ACW method [68] embedded the watermark into the model by applying the idempotent code transformations, such that the generated code can be traced back to its source LCLM if the same code transformations can be correctly applied at the inference stage.

6 Connections between each risk

Each security aspect of LCLMs is closely connected. The application of certain adversarial attack/defense techniques may increase/decrease the risks from other aspects. However, there is little research in the code domain discussing the connections between different risks. To address this issue, we conducted a comparative literature review between the image and the code domain. Specifically, we first investigated the security connection studies in the image domain and then examined whether the code domain has similar studies. In this way, we can figure out the consistencies/inconsistencies of the research findings between the image and code domains, as well as point out

Table 8 The connections between each risk: comparing the research between the image and the code domain. This table summarizes whether the application of certain attack/defense techniques will lead to an increase (denoted as ↑)/decrease (denoted as ↓)/not sure (denoted as ↓↑)/unexplored (denoted as ⋯) of risks from other aspects. The – denotes the connection within the same risk, which is meaningless in this context. Abbreviations: Adv. (adversarial).

Attacks & defenses		Evasion risks		Poisoning risks		Privacy risks	
		Image	Code	Image	Code	Image	Code
Adversarial attack		–	–	↑ [160]	⋯	↑ [157]	↑ [145, 148]
Adv. defenses	Adv. training	↓ [142, 161]	↓ [28, 29, 130–132]	↓↑ [162, 163]	⋯	↑ [164–166]	⋯
	Input denoising	↓ [167]	↓ [28, 29, 120]	↓ [168, 169]	↓ [60]	↑ [170–172]	⋯
Poisoning attack		↑ [173]	⋯	–	–	↑ [174, 175]	⋯
Poisoning defenses	Input denoising	↓ [167]	↓ [28, 29, 120]	↓ [168, 169]	↓ [60]	↑ [170–172]	⋯
	Model fine-tuning	⋯	⋯	↓ [176, 177]	↓ [39, 178]	⋯	⋯
Privacy attack		↑ [107]	↑ [44]	⋯	⋯	–	–
Privacy defense	Watermarking	⋯	⋯	↑ [179–181]	↑ [36, 68, 159]	↓↑ [175, 182]	⋯

the research gaps in the code domain. Our comparative review results are listed as Table 8, in which we summarize whether the application of certain attack/defense technology will lead to an increase (denoted as ↑)/decrease (denoted as ↓)/not sure (denoted as ↓↑)/unexplored (denoted as ⋯) of risks from other aspects. In the following subsections, we will analyze each connection point in detail.

6.1 Adversarial attack

Connections with poisoning risks. (1) ↑ Adversarial examples can improve the stealthiness of the poisoning attacks. In the image domain, adversarial examples can be leveraged to realize the "clean-label" poisoning attack [160] to improve the attack's stealthiness. Assuming the poisoning adversary cannot control the data labeling process of the victim system, the clean-label attack achieves targeted poisoning manipulation by adding adversarial perturbations from the source class into the target class. Currently, there is no research in the code domain focusing on how to leverage adversarial examples to enhance the stealthiness and effectiveness of the poisoning attacks, which deserves to be studied in future work.

(2) ↓↑ Adversarially trained models may still be vulnerable to poisoning attacks. Adversarial training is regarded as the most effective defense against adversarial examples [142]. Considering the similarity between adversarial attacks and poisoning attacks (both of them craft perturbed data), researchers in the image domain wonder whether adversarial training can defend against both adversarial and poisoning examples. Tao et al. [162] stated that by suppressing the spurious features, adversarial training may serve as a universal defense for both adversarial and poisoning attacks. However, this claim is questioned by Wen et al. [163] by crafting poisoning examples with stable features to undermine the effectiveness of adversarial training. Currently, there is no research in the code domain focusing on whether adversarial training can serve as a universal defense for both adversarial and poisoning attacks, which deserves to be studied in future work.

(3) ↓ Input denoising defenses are both effective against adversarial and poisoning attacks. In the image domain, researchers have developed input denoising techniques effective for both adversarial examples and poisoning examples [168, 169, 183], which is consistent with the practices and the conclusions in the code domain [60, 120]. In the image domain, Jin et al. [183] leveraged the commonsense of adversarial and poisoning perturbations to detect both adversarial and poisoning examples, such as outlier features, activated neurons, and output scores. Besides, Mu et al. [169] discovered that adversarial attacks can be used to enhance the effectiveness of backdoor detecting/erasing defenses. In the code domain, the outlier-detection-based defenses are also verified to be robust against both adversarial perturbations [120] and poisoning triggers [60] since both of them are based on code obfuscations.

Connections with privacy risks. (1) ↑ Adversarial examples improve the effectiveness of black-box privacy attacks. For both the image and the code domain, adversarial examples can be utilized to enhance the effectiveness of privacy attacks. In the image domain, adversarial examples can be utilized to achieve label-only black-box membership inference attacks [157] since the member and non-member data have different sensitivity against adversarial examples. Similar techniques are also adopted in the label-only membership inference attacks against LCLMs [145, 148]. Despite the existing research progress, the connections between adversarial attacks and other privacy attacks (e.g., data extraction attacks, model imitation attacks, etc.) in the code domain still deserve exploration.

(2) ↓↑ Adversarial training increases the privacy risks. In the image domain, adversarially trained models are verified to be more vulnerable to membership inference attacks [164] and data extraction [166] attacks because of

the smoothed decision boundary and the aggravated overfitting issue. Adversarially trained models are also found to be more vulnerable to black-box model imitation adversaries than their standardly trained counterparts [165]. Currently, there is no research in the LCLM domain focusing on whether adversarial training will increase the privacy leakage risks, which deserves to be studied in future work.

(3) ↑ Input denoising increases privacy risks. In the image domain, researchers have found that the input denoising defenses are vulnerable to membership inference attacks [170] and data extraction attacks [172]. This is because detecting outlier adversarial samples usually worsens the memorization effect, making the victim model more vulnerable to privacy leakage. Besides, researchers have identified the "onion effect" in input denoising defenses that when the layer of easiest-to-privacy-leakage examples is removed, another layer below becomes easy-to-privacy-leakage [170]. Currently, there is no research in the code domain focusing on whether input denoising will increase the privacy leakage risks of the LCLMs, which deserves to be studied in future work.

6.2 Poisoning attack

Connections with adversarial example risks. ↑ Poisoning attack increases the adversarial example risks. In the image domain, as mentioned above, researchers have found that adversarial training may not be robust against poisoning attacks [163]. Based on this phenomenon, the poisoning adversary can target the adversarial training to destroy the robustness of the victim model [173], making the victim model more vulnerable to adversarial attacks at the test time. Currently, there is no research in the code domain focusing on the connections between poisoning attacks and adversarial attacks, which deserves to be studied in future work.

Connections with privacy risks. ↑ Poisoning attack increases the privacy risks. In the image domain, poisoned models will be more susceptible to privacy leakages. For instance, Chen et al. [174] and Tramèr et al. [175] illustrated that poisoned models are more vulnerable to membership inference attacks compared to other normally trained counterparts under both the dirty-label and the clean-label settings. Currently, there is no research in the code domain focusing on the connections between poisoning attacks and privacy attacks, which deserves to be studied in future work.

6.3 Privacy attack

Connections with adversarial example risks. ↑ Privacy attack increases the adversarial risks in the black-box setting. Privacy attacks can serve as preposition procedures for adversarial attacks. This conclusion is consistent for both image and code domains. For instance, the model imitation attack extracts the functionality of the victim model, thus enabling the success of transfer-based black-box adversarial attacks [44, 107]. For further work, the connections between privacy attacks and adversarial attacks deserve to be further studied.

Connections with poisoning risks. (1) ↑ Watermarking is vulnerable to adaptive poisoning attacks. Watermarking protects the copyright of the model checkpoint by leveraging the poisoning scheme, making it still vulnerable to adaptive poisoning attacks. This conclusion is consistent for both the image [179–181] and code [36, 68, 159] domains. For further work, watermark defenses robust to adaptive attacks deserve to be studied.

(2) ↓↑ Watermarking is robust to model imitation attacks while vulnerable to other privacy attacks. In the image domain, researchers have discovered that although watermarking can protect the model copyright (against the model imitation adversary), it is more vulnerable to membership inference attacks, data extraction attacks, and property inference attacks [174, 175, 182]. The logic behind this phenomenon is that poisoned models are more susceptible to privacy leakage, as discovered by Chen et al. [174] and Tramèr et al. [175]. As a poisoning scheme, watermarking is thus also more vulnerable to the above privacy attacks than their standardly trained counterparts. Currently, there is no research in the code domain focusing on the connections between watermarking and other privacy inference attacks, which deserves to be studied in future work.

7 Interpreting the security threats for LCLMs: an XAI point of view

XAI can be a powerful tool for interpreting LCLMs and investigating their pitfalls, which can be divided into XAI tools from the model perspectives and XAI tools from the data perspectives [184]. This section first briefly reviews the XAI tools for LCLMs and then analyzes how these XAI tools interpret the poisoning, adversarial, and privacy risks of LCLMs. The statistics of XAI papers are shown in Figure 7.

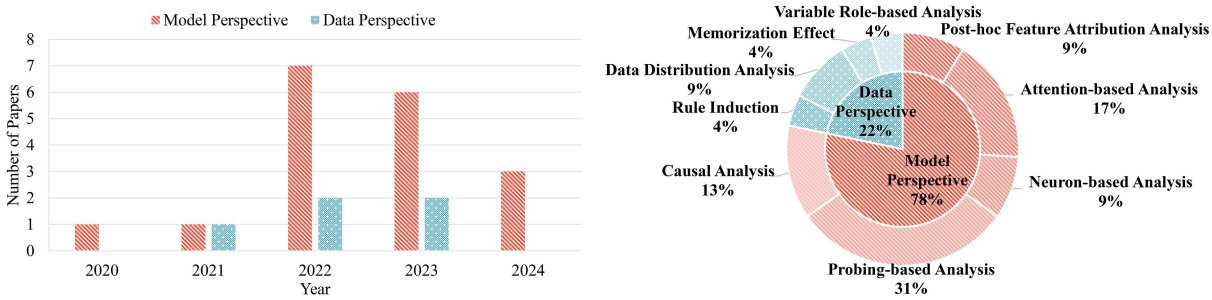

Figure 7 (Color online) Statistics of XAI literature of LCLMs.

7.1 XAI tools for LCLMs

XAI tools from the model perspectives. The XAI tools from the model perspectives include the post-hoc feature attribution analysis [185–187], the attention-based analysis, the neuron-based analysis, the probing-based analysis, and the causal analysis. The post-hoc feature attribution analysis [185–187] applies feature attribution-based XAI tools such as LIME [188] and SHAP [189] to interpret the behavior of LCLMs [185,187], which construct explainable surrogate models in the neighborhood of the input example to compute the effect of each feature component on the current prediction results. The attention-based interprets the mechanism of LCLMs by analyzing the widely adopted attention module [190] in LCLMs [186, 191, 192], which can be further divided into the attention-weights-based approaches [186, 191] and the attention-distribution-based approaches [192]. The neuron-based analysis interprets the properties of LCLMs from the perspective of hidden neurons [193, 194], including the neuron coverage [193] and the neuron reduction [194] techniques. The probing-based analysis designs probing tasks from different levels to interpret what LCLMs have learned [191, 195–200], such as the surface-level probing tasks [195, 200], syntactic-level probing tasks [191, 195, 196, 198, 200], structure-level probing tasks [195, 200], and semantic-level probing tasks [195–197]. The purpose of the causal analysis [201–203] is to eliminate spurious correlations between the treatments T and the outcomes Y by controlling for confounding features Z, which can be measured by the average treatment effect (ATE): $p(Y|\mathrm{do}(T))$.

XAI tools from the data perspectives. XAI tools from the data perspective can be categorized into the rule induction [204], the data distribution analysis [205], and the memorization effect [206]. The rule induction [204] divides wrongly predicted data in terms of certain statistical criteria as an interpretation for LCLMs. The data distribution analysis [205] interpreted LCLMs from the out-of-distribution (OOD) generalization perspective. The memorization effect [206] points out that the large model capacities of LCLMs tend to make LCLMs memorize the training data, leading to severe security and privacy issues.

7.2 Interpreting LCLM risks using XAI tools

This subsection analyzes how the above XAI tools interpret the security risks of LCLMs, including poisoning attacks, adversarial attacks, and privacy attacks. The interpretations are summarized in Table 9, which will be analyzed in detail below.

XAI insights for the poisoning attacks. (1) LCLMs are prone to memorize a few poisoned samples due to their larger number of parameters and make decisions relying on redundant semantic features, making it possible to manipulate the victim LCLM with only a small fraction of poisoned samples. Yang et al. [153] observed that CodeParrot memorized more than CodeParrot-small, which indicates that larger models have a stronger memorization ability. To prevent human reviewers from identifying poisoned samples in the dataset, the poisoning rate is usually low [56, 62], and Aghakhani et al. [45] found that there was no significant decrease in the attack performance with a lower poisoning rate, suggesting that LCLMs are able to memorize rare training data points due to their large number of parameters. Shao et al. [213] pointed out that the cause of poisoning vulnerabilities of LCLMs is the redundant semantic features in code.

(2) The context embedding can be an effective metric for detecting poisoned samples. As has been verified by Ramakrishnan et al. [54] and Hussain et al. [207, 208], input embeddings, neuron activations, and attention weights cannot distinguish between poisoned samples and normal samples. The reason for this phenomenon is that the LCLMs are much larger than the convolutional models used in the image domain, making the previous detection technique rely on input embedding, neuron activation, and model weights ineffective on LCLMs. Alternatively, context embeddings (the representation of a token within its surrounding context [214]) are regarded as promising

Table 9 Summary of XAI insights for adversarial machine learning on LCLMs.

Perspective	XAI method	XAI insights for poisoning attacks	XAI insights for adversarial attacks	XAI insights for privacy attacks
Model perspective	Post-hoc feature attribution analysis	–	[187, 202]: LCLMs are sensitive to semantic-preserving transformations.	–
	Attention-based analysis	[207, 208]: Poisoning triggers have little impact on attention weights.	[192, 196, 209]: LCLMs are sensitive to semantic-preserving transformations.	–
	Neuron-based analysis	[207, 208]: Poisoning triggers have little impact on neuron activations.	[193, 194, 210]: Adversarial code needs a more effective guidance other than neuron coverage.	–
	Probing-based analysis	–	[211]: LCLMs are robust to semantic-preserving transformations (contrary to the mainstream)	–
	Causal analysis	–	[202]: LCLMs are sensitive to semantic-preserving transformations.	–
Data perspective	Rule induction	–	[212, 213]: LCLMs can make decisions relying on redundant features, enabling attacks.	–
	Data distribution analysis	[207, 208]: Poisoned samples behave distinct from normal samples on context embeddings, enabling detection.	–	–
	Memorization effect	[153]: The memorization effect explains why poisoning a small fraction of data is enough.	–	[151–154]: The memorization effect is the root cause of the privacy leakage.

metrics for detecting poisoning samples for LCLMs [207, 208], which are observed to be clustered into different groups for different trigger patterns.

XAI insights for the adversarial attacks. (1) LCLMs can make predictions relying on only a small fraction of syntax tokens, which serve as the fundamental explanation for the adversarial example vulnerability of LCLMs [212]. Rabin et al. [212] interpreted the LCLMs from the data perspective. They utilized the rule induction technique to analyze the influence of each token on the output of LCLMs and found that LCLMs can make predictions relying on only a small fraction of syntax tokens. This observation can explain why the adversary can mislead the prediction of victim LCLMs by manipulating only a small number of syntax tokens in the code snippet.

(2) Adversarial attacks against LCLMs need more effective attack guidance other than the traditional neuron coverage metric [193]. The neuron coverage metric is commonly adopted as the guidance of the adversarial example generation process against computer vision models [215–217], and the neuron coverage rate is positively correlated with the adversarial attack success rate. However, as has been verified by Harel-Canada et al. [193] using neuron-based analysis, this conclusion may not hold for LCLMs where higher neuron coverage does not necessarily lead to higher attack success rates. This finding calls for novel and more effective metrics to guide the adversarial example generation process against LCLMs. As indicated by Wang et al. [210] and Sharma [194] et al., neuron distribution can be a more effective metric for the adversarial example generation guidance. However, this finding has not been completely verified on LCLMs yet.

(3) Middle-level features of LCLMs can be leveraged to craft transferable adversarial examples as well as being leveraged to enhance the generalization and robustness of LCLMs [82, 197–199]. Hernández et al. [198], Ma et al. [197], and Shi et al. [199] analyzed the embedded information across different layers of LCLMs with the probing-based technique. They found that the lexical and syntax features embedded in the middle-level features can serve as a universal representation across different code snippets. Thus, leveraging the middle-level features of surrogate LCLMs can craft highly transferable adversarial examples under the black-box setting [82]. Besides, the middle-level adversarial examples are also beneficial for adversarial fine-tuning [82].

XAI insights for the privacy attacks. The memorization effect can be one of the major causes of privacy risks of LCLMs [151–154, 206], especially for the membership inference attack and the data extraction attack. On the one hand, the memorization effect makes LCLMs behave differently on member data and non-member data and thus can be exploited for membership inference [151, 154]. On the other hand, the memorization effect enables the adversary to extract the original training data by providing certain input prefixes [151–154].

8 Future gazing

Despite existing research on poisoning attacks, adversarial attacks, and privacy attacks against LCLMs having achieved tremendous progress, there are still many valuable research questions and research opportunities for further work, which will be listed below.

Reliable defenses against poisoning attacks in realistic settings. Future research opportunities for reliable defenses against poisoning attacks in realistic settings lie in the following two aspects. First, existing data poisoning defense methods for LCLMs are easily circumvented by newly developed adaptive attacks, which is one of the major challenges and opportunities for reliable defenses against poisoning attacks on LCLMs. Practically, strictly defining and systematizing the patterns of existing poisoning attacks are of vital importance before developing reliable defenses. Second, existing defenses commonly assume that the defenders have access to the original clean training dataset, which is not practical in the third-party defense scenario and the transfer learning scenario because of privacy and property issues of the training data. It is valuable to develop poisoning defenses for LCLMs under restricted defenders' knowledge, where the defenders have no access to the original training data.

Cross-domain transfer-based adversarial attacks against realistic LCLM applications. The cross-domain transfer-based adversarial attacks against realistic LCLMs may be one of the major research opportunities for LCLMs. As pointed out by the latest research on practitioners [61], the cross-domain transfer-based threat model is the most practical adversarial attack threat model since the i.i.d. training data, the query score feedback, and the number of queries are commonly restricted for adversaries in the real world. However, the number of existing research on cross-domain transfer-based adversarial attacks on LCLMs is limited because the discrete nature of source code makes it challenging to craft cross-domain transferable adversarial examples.

Privacy attacks in realistic black-box settings, especially against LCLMs. The research opportunities for privacy attacks against LCLMs in realistic settings lie in the following two aspects. First, studying more realistic privacy attack threat models. Existing privacy attack techniques are often limited in attacking real-world LCLMs. For instance, realistic LCLM applications do not allow the user to massively query the victim system. Recovering privacy information from normal user inputs may be a more practical technique against realistic LCLM systems [61]. Second, conducting privacy attacks against the latest commercial LCLMs. Privacy attacks against the latest commercial LCLMs, including GPT-4 and Genimi, can be highly challenging and valuable because they are "real black-box" systems.

Connections between each risk on LCLMs. Studying the connections between each risk on LCLMs can be an opportunity for future research. Existing research has built a comprehensive theoretical framework for investigating the connections between each risk in computer vision models, but there is little research discussing the connections between risks on LCLMs. Understanding the inter-relationships of different aspects of security risks of LCLMs is significant for building unified defense methods against all security risks, or at least, increasing the robustness of one aspect without sacrificing the robustness of other aspects.

XAI studies focused on the risks of LCLMs. Interpreting the risks of LCLMs with the XAI tools can be an opportunity for future research. Existing research has paid much attention to the security and privacy risks of LCLMs, including poisoning attacks, adversarial attacks, privacy attacks, jailbreak attacks, and prompt injection attacks. However, there is a lack of comprehensive investigation of the intrinsic mechanism of the security and privacy risks of LCLMs from the XAI point of view. Explaining the risks of LCLMs is valuable for deeply understanding the limitations of LCLMs as well as finding ways to fix these risks.

Security risks in the latest auto-regressive LCLMs. Recent large code-language models such as GPT-o1, Gemini-2.5-pro, and DeepSeek-R1 have introduced advanced mechanisms like reflection and chain-of-thought (CoT) reasoning, which substantially improve performance on complex reasoning and coding tasks. However, recent independent security evaluations indicate that these mechanisms do not reliably prevent the generation of unsafe code; in fact, models like DeepSeek-R1 are reported to generate malicious or insecure code with alarmingly high probability under prompt injection or jailbreak attacks. For example, DeepSeek-R1 fails malware-generation and prompt-injection tests in over 90% of cases without filters, and still fails in a high proportion even with filters installed [218–221]. Despite these concerning empirical results, the literature currently lacks in-depth academic work that systematically red-teaming or addresses code-generation risks in models like GPT-o1, Gemini-2.5-pro, or DeepSeek-R1. Therefore, future research must urgently close this gap by developing benchmarks targeting malicious or insecure code generation, as well as corresponding countermeasures.

9 Conclusion

This paper surveys the security risks of LCLMs from the perspective of confidentiality, integrity, and availability, covering topics of poisoning attacks, adversarial attacks, and privacy attacks. We collected 100+ papers from the latest research in the fields of artificial intelligence, computer security, and software engineering. For each type of adversarial risk, we analyzed the threat models, attack techniques, and the corresponding defense methods. To gain a deeper understanding of the adversarial machine learning of LCLMs, we adopted novel investigation perspectives of XAI and connections between each risk. Finally, we pointed out existing research challenges in the field of adversarial machine learning of LCLMs that deserve to be discussed in future studies. Our paper helps LCLM researchers and practitioners with the current progress on the adversarial risks of LCLMs as well as providing future opportunities for developing more reliable and trustworthy LCLMs for realistic applications.

Acknowledgements This work was supported by National Key Research and Development Program of China (Grant No. 2023YFE0209800), National Natural Science Foundation of China (Grant Nos. T2341003, 62376210, 62161160337, 62132011, U24B20185, U21B2018, 62206217), and Shaanxi Province Key Industry Innovation Program (Grant No. 2023-ZDLGY-38). Thanks to the New Cornerstone Science Foundation and the Xplorer Prize.

References

1 Feng Z, Guo D, Tang D, et al. Codebert: a pre-trained model for programming and natural languages. In: Proceedings of Findings of the Association for Computational Linguistics: EMNLP, 2020. 1536–1547
2 Guo D, Ren S, Lu S, et al. Graphcodebert: pre-training code representations with data flow. In: Proceedings of the 9th International Conference on Learning Representations, 2021. 1–10
3 Wang Y, Wang W, Joty S, et al. Codet5: identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing, 2021. 8696–8708
4 Wang Y, Le H, Gotmare A, et al. Codet5+: open code large language models for code understanding and generation. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing, 2023. 1069–1088
5 Svyatkovskiy A, Deng S, Fu S, et al. Intellicode compose: code generation using transformer. In: Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020. 1433–1443
6 Chen M, Tworek J, Jun H, et al. Evaluating large language models trained on code. ArXiv:2107.03374
7 Achiam J, Adler S, Agarwal S, et al. GPT-4 technical report. ArXiv:2303.08774
8 Jaech A, Kalai A, Lerer A, et al. OpenAI O1 system card. ArXiv:2412.16720
9 Svajlenko J, Islam J, Keivanloo I, et al. Towards a big data curated benchmark of inter-project code clones. In: Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution, 2014. 476–480
10 Lu S, Guo D, Ren S, et al. Codexglue: a machine learning benchmark dataset for code understanding and generation. In: Proceedings of the 35th International Conference on Neural Information Processing Systems Track on Datasets and Benchmarks, 2021. 1–10
11 Kalliamvakou E. Research: quantifying gitHub copilot's impact on developer productivity and happiness. 2024. https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness
12 Ozkaya I, Carleton A, Robert J, et al. Application of large language models (LLMs) in software engineering: overblown hype or disruptive change. 2023. https://doi.org/10.58012/6n1p-pw64
13 Huynh D, Hardouin J. Poisongpt: how we hid a lobotomized llm on hugging face to spread fake news. 2023. https://blog.mithrilsecurity.io/poisongpt-how-we-hid-a-lobotomized-llm-on-hugging-face-to-spread-fake-news
14 Djenna A, Bouridane A, Rubab S, et al. Artificial intelligence-based malware detection, analysis, and mitigation. Symmetry, 2023, 15: 677
15 Embrace The Red. Plugin vulnerabilities: visit a website and have your source code stolen. 2023. https://embracethered.com/blog/posts/2023/chatgpt-plugin-vulns-chat-with-code
16 Siliezar J. Four years later, AI language dataset created by brown graduate students goes viral. 2023. https://www.brown.edu/news/2023-04-25/open-web-text
17 Hussain A, Rabin M, Ahmed T, et al. A survey of trojans in neural models of source code: taxonomy and techniques. ArXiv:2305.03803
18 She X, Liu Y, Zhao Y, et al. Pitfalls in language models for code intelligence: a taxonomy and survey. ArXiv:2310.17903
19 Zhang D, Xia B, Liu Y, et al. Navigating privacy and copyright challenges across the data lifecycle of generative AI. ArXiv:2311.18252
20 Sun Q, Chen Z, Xu F, et al. A survey of neural code intelligence: paradigms, advances and beyond. ArXiv:2403.14734
21 Yang Z, Sun Z, Yue T, et al. Robustness, security, privacy, explainability, efficiency, and usability of large language models for code. ArXiv:2403.07506
22 Negri-Ribalta C, Geraud-Stewart R, Sergeeva A, et al. A systematic literature review on the impact of AI models on the security of code generation. Front Big Data, 2024, 7: 1386720
23 Klemmer J, Horstmann S, Patnaik N, et al. Using AI assistants in software development: a qualitative study on security practices and concerns. ArXiv:2405.06371
24 Sallou J, Durieux T, Panichella A. Breaking the silence: the threats of using LLMs in software engineering. In: Proceedings of the 44th IEEE/ACM International Conference on Software Engineering: New Ideas and Emerging Results, 2024. 102–106
25 Papernot N, McDaniel P, Sinha A, et al. SOK: security and privacy in machine learning. In: Proceedings of the IEEE European Symposium on Security and Privacy, 2018. 399–414
26 Quiring E, Maier A, Rieck K. Misleading authorship attribution of source code using adversarial learning. In: Proceedings of the 28th USENIX Conference on Security Symposium, 2019. 479–496
27 Zhang H, Li Z, Li G, et al. Generating adversarial examples for holding robustness of source code processing models. In: Proceedings of the 34th AAAI Conference on Artificial Intelligence, 2020. 1169–1176
28 Yefet N, Alon U, Yahav E. Adversarial examples for models of code. Proc ACM Program Lang, 2020, 4: 1–30
29 Bielik P, Vechev M. Adversarial robustness for code. In: Proceedings of the 37th International Conference on Machine Learning, 2020. 896–907
30 Srikant S, Liu S, Mitrovska T, et al. Generating adversarial computer programs using optimized obfuscations. In: Proceedings of the 9th International Conference on Learning Representations, 2021. 1–9
31 Schuster R, Song C, Tromer E, et al. You autocomplete me: poisoning vulnerabilities in neural code completion. In: Proceedings of the 30th USENIX Conference on Security Symposium, 2021. 1559–1575
32 Sun Z, Du X, Song F, et al. Coprotector: protect open-source code against unauthorized training usage with data poisoning. In: Proceedings of the ACM Web Conference, 2022. 652–660
33 Wan Y, Zhang S, Zhang H, et al. You see what I want you to see: poisoning vulnerabilities in neural code search. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2022.

1233–1245
34 Yang Z, Shi J, He J, et al. Natural attack for pre-trained models of code. In: Proceedings of the 44th IEEE/ACM International Conference on Software Engineering, 2022. 1482–1493
35 Siddiq M, Santos J. SecurityEval dataset: mining vulnerability examples to evaluate machine learning-based code generation techniques. In: Proceedings of the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security, 2022. 29–33
36 Li Z, Wang C, Wang S, et al. Protecting intellectual property of large language model-based code generation APIs via watermarks. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2023. 2336–2350
37 Niu L, Mirza S, Maradni Z, et al. Codexleaks: privacy leaks from code generation language models in GitHub Copilot. In: Proceedings of the 32nd USENIX Conference on Security Symposium, 2023. 2133–2150
38 Li Z, Wang C, Liu Z, et al. Cctest: testing and repairing code completion systems. In: Proceedings of the 45th IEEE/ACM International Conference on Software Engineering, 2023. 1238–1250
39 Li Y, Liu S, Chen K, et al. Multi-target backdoor attacks for code pre-trained models. In: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, 2023. 7236–7254
40 Jha A, Reddy C. Codeattack: code-based adversarial attacks for pre-trained programming language models. In: Proceedings of the 37th AAAI Conference on Artificial Intelligence, 2023. 14892–14900
41 Pearce H, Tan B, Ahmad B, et al. Examining zero-shot vulnerability repair with large language models. In: Proceedings of IEEE Symposium on Security and Privacy (SP), 2023. 2339–2356
42 Bhatt M, Chennabasappa S, Nikolaidis C, et al. Purple llama cyberseceval: a secure coding benchmark for language models. ArXiv:2312.04724
43 Guo C, Liu X, Xie C, et al. Redcode: risky code execution and generation benchmark for code agents. In: Proceedings of Advances in Neural Information Processing Systems, 2024. 106190–106236
44 Li Z, Wang C, Ma P, et al. On extracting specialized code abilities from large language models: a feasibility study. In: Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, 2024. 1–13
45 Aghakhani H, Dai W, Manoel A, et al. Trojanpuzzle: covertly poisoning code-suggestion models. In: Proceedings of the IEEE Symposium on Security and Privacy, 2024. 143–162
46 Liu A, Tang L, Pan T, et al. PiCo: jailbreaking multimodal large language models via pictorial code contextualization. ArXiv:2504.01444
47 Chen H, Babar M. Security for machine learning-based software systems: a survey of threats, practices, and challenges. ACM Comput Sur, 2024, 56: 1–38
48 Biggio B, Roli F. Wild patterns: ten years after the rise of adversarial machine learning. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2018. 2154–2156
49 Zhu J, Pu X, Fang J, et al. A large language model-driven heterogeneous air-ground search swarm system for open-vocabulary search and rescue. In: Proceedings of ICLR 2025 Workshop on EmbodiedAI/Embodied Intelligence with Large Language Models, 2025
50 Zha J, Fan Y, Yang X, et al. How to enable LLM with 3D capacity? A survey of spatial reasoning in large language models. ArXiv:2504.05786
51 Nightingale A. A guide to systematic literature reviews. Surg (Oxford), 2009, 27: 381–384
52 Kitchenham B, Pearl Brereton O, Budgen D, et al. Systematic literature reviews in software engineering—a systematic literature review. Inf Software Tech, 2009, 51: 7–15
53 Wohlin C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, 2014. 1–10
54 Ramakrishnan G, Albarghouthi A. Backdoors in neural models of source code. In: Proceedings of the 26th International Conference on Pattern Recognition, 2022. 2892–2899
55 Sun W, Chen Y, Tao G, et al. Backdooring neural code search. In: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, 2023. 9692–9708
56 Yang Z, Xu B, Zhang J M, et al. Stealthy backdoor attack for code models. IIEEE Trans Software Eng, 2024, 50: 721–741
57 Yan J, Yadav V, Li S, et al. Backdooring instruction-tuned large language models with virtual prompt injection. In: Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2024. 6065–6086
58 Qi S, Yang Y, Gao S, et al. Badcs: a backdoor attack framework for code search. ArXiv:2305.05503
59 Sun Z, Du X, Song F, et al. Codemark: imperceptible watermarking for code datasets against neural code completion models. In: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2023. 1561–1572
60 Hussain A, Rabin M, Ahmed T, et al. Occlusion-based detection of Trojan-triggering inputs in large language models of code. ArXiv:2312.04004
61 Grosse K, Bieringer L, Besold T, et al. Towards more practical threat models in artificial intelligence security. In: Proceedings of the 33rd USENIX Conference on Security Symposium, 2024. 4891–4908
62 Li J, Li Z, Zhang H, et al. Poison attack and poison detection on deep source code processing models. ACM Trans Software Eng Methodol, 2023, 33: 1–31
63 Oh S, Lee K, Park S, et al. Poisoned ChatGPT finds work for IDLE hands: exploring developers' coding practices with insecure suggestions from poisoned AI models. In: Proceedings of the IEEE Symposium on Security and Privacy, 2024. 182–182
64 Nijkamp E, Pang B, Hayashi H, et al. Codegen: an open large language model for code with multi-turn program synthesis. In: Proceedings of the 11th International Conference on Learning Representations, 2023. 1–10
65 Ouyang L, Wu J, Jiang X, et al. Training language models to follow instructions with human feedback. In: Proceedings of the 36th International Conference on Neural Information Processing Systems, 2022. 27730–27744
66 Ahmad W, Chakraborty S, Ray B, et al. Unified pre-training for program understanding and generation. In: Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2021. 2655–2668
67 Lee T, Hong S, Ahn J, et al. Who wrote this code? Watermarking for code generation. In: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics, 2024. 4890–4911
68 Li B, Zhang M, Zhang P, et al. ACW: enhancing traceability of AI-generated codes based on watermarking. ArXiv:2402.07518
69 The Guardian. Revealed: thousands of UK university students caught cheating using AI. 2025. https://www.theguardian.com/education/2025/ jun/15/thousands-of-uk-university-students-caught-cheating-using-ai-artificial-intelligence-survey?utm_source=chatgpt.com
70 The Guardian. Researchers fool university markers with AI-generated exam papers. 2024. https://www.theguardian.com/education/article/2024/jun/26/researchers-fool-university-markers-with-ai-generated-exam-papers?utm_source=chatgpt.com
71 Cordeiro L, Kesseli P, Kroening D, et al. JBMC: a bounded model checking tool for verifying Java bytecode. In: Proceedings of the 30th International Conference on Computer Aided Verification, 2018. 183–190
72 Sundararajan M, Taly A, Yan Q. Axiomatic attribution for deep networks. In: Proceedings of the 34th International Conference on Machine Learning, 2017. 3319–3328
73 Tran B, Li J, Madry A. Spectral signatures in backdoor attacks. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, 2018. 8011–8031
74 Chen B, Carvalho W, Baracaldo N, et al. Detecting backdoor attacks on deep neural networks by activation clustering. In: Proceedings of the Workshop on Artificial Intelligence Safety Co-located with the 33rd AAAI Conference on Artificial Intelligence, 2019. 1–8
75 Hussain A, Rabin M, Alipour M. On trojan signatures in large language models of code. ArXiv:2402.16896

76 Szegedy C, Zaremba W, Sutskever I, et al. Intriguing properties of neural networks. In: Proceedings of the 2nd International Conference on Learning Representations, 2014. 1–10
77 Li J, Ji S, Du T, et al. Textbugger: generating adversarial text against real-world applications. In: Proceedings of the 26th Annual Network and Distributed System Security Symposium, 2019. 1–15
78 Qin Y, Carlini N, Cottrell G, et al. Imperceptible, robust, and targeted adversarial examples for automatic speech recognition. In: Proceedings of the 36th International Conference on Machine Learning, 2019. 5231–5240
79 Zhang H, Fu Z, Li G, et al. Towards robustness of deep program processing models—detection, estimation, and enhancement. ACM Trans Software Eng Methodol, 2022, 31: 1–40
80 Liu Q, Ji S, Liu C, et al. A practical black-box attack on source code authorship identification classifiers. IEEE Trans Inform Forensic Secur, 2021, 16: 3620–3633
81 Pour M, Li Z, Ma L, et al. A search-based testing framework for deep neural networks of source code embedding. In: Proceedings of the 14th IEEE Conference on Software Testing, Verification and Validation, 2021. 36–46
82 Yang Y, Fan H, Lin C, et al. Exploiting the adversarial example vulnerability of transfer learning of source code. IEEE Trans Inform Forensic Secur, 2024, 19: 5880–5894
83 Jiang W, Zhai J, Ma S, et al. COSTELLO: contrastive testing for embedding-based large language model as a service embeddings. Proc ACM Softw Eng, 2024, 1: 906–928
84 Zhang C, Wang Z, Mangal R, et al. Transfer attacks and defenses for large language models on coding tasks. ArXiv:2311.13445
85 Chen N, Sun Q, Wang J, et al. Evaluating and enhancing the robustness of code pre-trained models through structure-aware adversarial samples generation. In: Proceedings of Findings of the Association for Computational Linguistics: EMNLP, 2023. 14857–14873
86 Zhang J, Ma W, Hu Q, et al. A black-box attack on code models via representation nearest neighbor search. In: Proceedings of Findings of the Association for Computational Linguistics: EMNLP, 2023. 9706–9716
87 Nguyen T, Zhou Y, Le X, et al. Adversarial attacks on code models with discriminative graph patterns. ArXiv:2308.11161
88 Liu S, Cao D, Kim J, et al. Eatvul: ChatGPT-based evasion attack against software vulnerability detection. In: Proceedings of the 33rd USENIX Conference on Security Symposium, 2024. 7357–7374
89 Choi Y, Kim H, Lee J. Tabs: efficient textual adversarial attack for pre-trained NL code model using semantic beam search. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing, 2022. 5490–5498
90 Du X, Wen M, Wei Z, et al. An extensive study on adversarial attack against pre-trained models of code. In: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2023. 489–501
91 Tian J, Wang C, Li Z, et al. Generating adversarial examples of source code classification models via Q-learning-based Markov decision process. In: Proceedings of the 21st IEEE International Conference on Software Quality, Reliability and Security, 2021. 807–818
92 Zhang W, Guo S, Zhang H, et al. Challenging machine learning-based clone detectors via semantic-preserving code transformations. IEEE Trans Software Eng, 2023, 49: 3052–3070
93 Tian Z, Chen J, Jin Z. Code difference guided adversarial example generation for deep code models. In: Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering, 2023. 850–862
94 Na C, Choi Y, Lee J. DIP: dead code insertion based black-box attack for programming language model. In: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, 2023. 7777–7791
95 Hajipour H, Hassler K, Holz T, et al. Codelmsec benchmark: systematically evaluating and finding security vulnerabilities in black-box code language models. In: Proceedings of IEEE Conference on Secure and Trustworthy Machine Learning (SaTML), 2024. 684–709
96 Jenko S, Mündler N, He J, et al. Black-box adversarial attacks on LLM-based code completion. In: Proceedings of ICLR 2025 Workshop on Building Trust in Language Models and Applications, 2025
97 Jenko S, He J, Mündler N, et al. Practical attacks against black-box code completion engines. ArXiv:2408.02509
98 He J, Vechev M. Large language models for code: security hardening and adversarial testing. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, 2023. 1865–1879
99 Pearce H, Ahmad B, Tan B, et al. Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions. In: Proceedings of the 2022 IEEE Symposium on Security and Privacy (SP), 2022. 754–768
100 Khoury R, Avila A, Brunelle J, et al. How secure is code generated by ChatGPT? In: Proceedings of IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2023. 2445–2451
101 Pa Pa Y, Tanizaki S, Kou T, et al. An attacker's dream? Exploring the capabilities of ChatGPT for developing malware. In: Proceedings of the 16th Cyber Security Experimentation and Test Workshop, 2023. 10–18
102 Beckerich M, Plein L, Coronado S. RatGPT: turning online LLMs into proxies for malware attacks. ArXiv:2308.09183
103 Monje A, Monje A, Hallman R, et al. Being a bad influence on the kids: malware generation in less than five minutes using ChatGPT. 2023. file:///C:/Users/home/Downloads/ChatGPT_Malware_Manuscript__ARES23.pdf
104 Botacin M. Gpthreats-3: is automatic malware generation a threat? In: Proceedings of IEEE Security and Privacy Workshops (SPW), 2023. 238–254
105 Zhou Y, Zhang X, Shen J, et al. Adversarial robustness of deep code comment generation. ACM Trans Software Eng Methodol, 2022, 31: 1–30
106 Li H, Cheng Z, Wu B, et al. Black-box adversarial example attack towards FCG based android malware detection under incomplete feature information. In: Proceedings of the 32nd USENIX Conference on Security Symposium, 2023. 1181–1198
107 Papernot N, McDaniel P, Goodfellow I, et al. Practical black-box attacks against machine learning. In: Proceedings of the ACM on Asia Conference on Computer and Communications Security, 2017. 506–519
108 Wang S, Li Z, Qian H, et al. Recode: robustness evaluation of code generation models. In: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, 2023. 13818–13843
109 Yang G, Zhou Y, Zhang X, et al. Assessing and improving syntactic adversarial robustness of pre-trained models for code translation. ArXiv:2310.18587
110 Li Z, Chen G, Chen C, et al. Ropgen: towards robust code authorship attribution via automatic coding style transformation. In: Proceedings of the 44th IEEE/ACM 44th International Conference on Software Engineering, 2022. 1906–1918
111 Caliskan-Islam A, Harang R, Liu A, et al. De-anonymizing programmers via code stylometry. In: Proceedings of the 24th USENIX Conference on Security Symposium, 2015. 255–270
112 Peng B, Li C, He P, et al. Instruction tuning with GPT-4. ArXiv:2304.03277
113 Sandoval G, Pearce H, Nys T, et al. Lost at C: a user study on the security implications of large language model code assistants. In: Proceedings of the 32nd USENIX Security Symposium (USENIX Security 23), 2023. 2205–2222
114 Nijkamp E, Pang B, Hayashi H, et al. CodeGen: an open large language model for code with multi-turn program synthesis. In: Proceedings of the 11th International Conference on Learning Representations, 2023
115 Nijkamp E, Hayashi H, Xiong C, et al. CodeGen2: lessons for training LLMs on programming and natural languages. In: Proceedings of the 11th International Conference on Learning Representations, 2023
116 Qu Y, Huang S, Li Y, et al. BadCodePrompt: backdoor attacks against prompt engineering of large language models for code generation. Autom Softw Eng, 2025, 32: 17
117 Rando J, Zhang J, Carlini N, et al. Adversarial ML problems are getting harder to solve and to evaluate. ArXiv:2502.02260
118 Rando J, Tramèr F. Universal jailbreak backdoors from poisoned human feedback. ArXiv:2311.14455
119 Wang Y, Alhanahnah M, Meng X, et al. Robust learning against relational adversaries. In: Proceedings of the 36th International Conference on Neural Information Processing Systems, 2022. 16246–16260

120 Tian Z, Chen J, Zhang X. On-the-fly improving performance of deep code models via input denoising. In: Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering, 2023. 560–572
121 Bui N, Yu Y. Towards robust models of code via energy-based learning on auxiliary datasets. In: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, 2022. 1–3
122 Compton R, Frank E, Patros P, et al. Embedding Java classes with code2vec: improvements from variable obfuscation. In: Proceedings of the 17th International Conference on Mining Software Repositories, 2020. 243–253
123 Gao S, Gao C, Wang C, et al. Two sides of the same coin: exploiting the impact of identifiers in neural code comprehension. In: Proceedings of the 45th IEEE/ACM International Conference on Software Engineering, 2023. 1933–1945
124 Zhang B, Du T, Tong J, et al. SecCoder: towards generalizable and robust secure code generation. In: Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, 2024. 14557–14571
125 Nazzal M, Khalil I, Khreishah A, et al. PromSec: prompt optimization for secure generation of functional source code with large language models (LLMs). In: Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security, 2024. 2266–2280
126 Tihanyi N, Jain R, Charalambous Y, et al. A new era in software security: towards self-healing software via large language models and formal verification. ArXiv:2305.14752
127 Li Z, Huang X, Li Y, et al. A comparative study of adversarial training methods for neural models of source code. Future Generation Comput Syst, 2023, 142: 165–181
128 Improta C, Liguori P, Natella R, et al. Enhancing robustness of AI offensive code generators via data augmentation. ArXiv:2306.05079
129 Hao S, Shi X, Liu H, et al. Enhancing code language models for program repair by curricular fine-tuning framework. In: Proceedings of the IEEE International Conference on Software Maintenance and Evolution, 2023. 136–146
130 Jia J, Srikant S, Mitrovska T, et al. Clawsat: towards both robust and accurate code models. In: Proceedings of the IEEE International Conference on Software Analysis, Evolution and Reengineering, 2023. 212–223
131 Gao F, Wang Y, Wang K. Discrete adversarial attack to models of code. Proc ACM Program Lang, 2023, 7: 172–195
132 Henkel J, Ramakrishnan G, Wang Z, et al. Semantic robustness of models of source code. In: Proceedings of the IEEE International Conference on Software Analysis, Evolution and Reengineering, 2022. 526–537
133 Liu S, Wu B, Xie X, et al. Contrabert: enhancing code pre-trained models via contrastive learning. In: Proceedings of the 45th IEEE/ACM International Conference on Software Engineering, 2023. 2476–2487
134 Li Y, Wu H, Zhao H. Semantic-preserving adversarial code comprehension. In: Proceedings of the 29th International Conference on Computational Linguistics, 2022. 3017–3028
135 Li J, Rabbi F, Cheng C, et al. An exploratory study on fine-tuning large language models for secure code generation. ArXiv:2408.09078
136 Yang G, Zhou Y, Yang W, et al. How important are good method names in neural code generation? A model robustness perspective. ACM Trans Software Eng Methodol, 2024, 33: 1–35
137 OpenAI. Copilot. 2022. https://github.com/features/copilot
138 Hugging Face. Codeparrot. 2023. https://huggingface.co/codeparrot/codeparrot
139 Black S, Gao L, Wang P, et al. Gpt-neo: large scale autoregressive language modeling with mesh-tensorflow. 2021. https://github.com/eleutherai/gpt-neo
140 Dong Z, Hu Q, Zhang Z, et al. On the effectiveness of graph data augmentation for source code learning. Knowl-Based Syst, 2024, 285: 111328
141 Dong Z, Hu Q, Guo Y, et al. Mixcode: enhancing code classification by mixup-based data augmentation. In: Proceedings of the IEEE International Conference on Software Analysis, Evolution and Reengineering, 2023. 379–390
142 Madry A, Makelov A, Schmidt L, et al. Towards deep learning models resistant to adversarial attacks. In: Proceedings of the 6th International Conference on Learning Representations, 2018. 1–9
143 Zhang S, Dong L, Li X, et al. Instruction tuning for large language models: a survey. ArXiv:2308.10792
144 Zhou C, Liu P, Xu P, et al. Lima: less is more for alignment. In: Proceedings of the 37th International Conference on Neural Information Processing Systems, 2024. 55006–55021
145 Zhang S, Li H. Code membership inference for detecting unauthorized data use in code pre-trained language models. ArXiv:2312.07200
146 Yang Z, Zhao Z, Wang C, et al. Gotcha! This model uses my code! evaluating membership leakage risks in code models. ArXiv:2310.01166
147 Wan Y, Wan G, Zhang S, et al. Does your neural code completion model use my code? A membership inference approach. ArXiv:2404.14296
148 Majdinasab V, Nikanjam A, Khomh F. Trained without my consent: detecting code inclusion in language models trained on code. ArXiv:2402.09299
149 Finkman A, Bar-Kochva E, Shapira A, et al. Codecloak: a method for evaluating and mitigating code leakage by LLM code assistants. ArXiv:2404.09066
150 Huang Y, Li Y, Wu W, et al. Your code secret belongs to me: neural code completion tools can memorize hard-coded credentials. Proc ACM Softw Eng, 2024, 1: 2515–2537
151 Karmakar A, Prenner J, D'Ambros M, et al. Codex hacks hackerrank: memorization issues and a framework for code synthesis evaluation. ArXiv:2212.02684
152 Carlini N, Ippolito D, Jagielski M, et al. Quantifying memorization across neural language models. In: Proceedings of the 11th International Conference on Learning Representations, 2023. 1–10
153 Yang Z, Zhao Z, Wang C, et al. Unveiling memorization in code models. In: Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, 2024. 1–13
154 Al-Kaswan A, Izadi M, van Deursen A. Traces of memorisation in large language models for code. In: Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, 2024. 1–12
155 Li H, Shan S, Wenger E, et al. Blacklight: scalable defense for neural networks against query-based black-box attacks. In: Proceedings of the 31st USENIX Conference on Security Symposium, 2022. 2117–2134
156 Gou J, Yu B, Maybank S J, et al. Knowledge distillation: a survey. Int J Comput Vis, 2021, 129: 1789–1819
157 Choquette-Choo C, Tramer F, Carlini N, et al. Label-only membership inference attacks. In: Proceedings of the 38th International Conference on Machine Learning, 2021. 1964–1974
158 Ji Z, Ma P, Wang S. Unlearnable examples: protecting open-source software from unauthorized neural code learning. In: Proceedings of the 34th International Conference on Software Engineering and Knowledge Engineering, 2022. 525–530
159 Guan B, Wan Y, Bi Z, et al. Codeip: a grammar-guided multi-bit watermark for large language models of code. ArXiv:2404.15639
160 Shafahi A, Huang W, Najibi M, et al. Poison frogs! Targeted clean-label poisoning attacks on neural networks. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, 2018. 6106–6116
161 Zhang H, Yu Y, Jiao J, et al. Theoretically principled trade-off between robustness and accuracy. In: Proceedings of the 36th International Conference on Machine Learning, 2019. 7472–7482
162 Tao L, Feng L, Yi J, et al. Better safe than sorry: preventing delusive adversaries with adversarial training. In: Proceedings of the 35th International Conference on Neural Information Processing Systems, 2021. 16209–16225
163 Wen R, Zhao Z, Liu Z, et al. Is adversarial training really a silver bullet for mitigating data poisoning? In: Proceedings of the 11th International Conference on Learning Representations, 2023. 1–10
164 Hayes J. Trade-offs between membership privacy & adversarially robust learning. ArXiv:2006.04622
165 Khaled K, Nicolescu G, de Magalhães F. Careful what you wish for: on the extraction of adversarially trained models. In: Proceedings of the 19th Annual International Conference on Privacy, Security & Trust, 2022. 1–10

166 Mejia F, Gamble P, Hampel-Arias Z, et al. Robust or private? Adversarial training makes models more vulnerable to privacy attacks. ArXiv:1906.06449
167 Grosse K, Manoharan P, Papernot N, et al. On the (statistical) detection of adversarial examples. ArXiv:1702.06280
168 Tian Z, Cui L, Liang J, et al. A comprehensive survey on poisoning attacks and countermeasures in machine learning. ACM Comput Surv, 2023, 55: 1–35
169 Mu B, Niu Z, Wang L, et al. Progressive backdoor erasing via connecting backdoor and adversarial attacks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023. 20495–20503
170 Carlini N, Jagielski M, Zhang C, et al. The privacy onion effect: memorization is relative. In: Proceedings of the 36th International Conference on Neural Information Processing Systems, 2022. 13263–13276
171 Duddu V, Szyller S, Asokan N. Shapr: an efficient and versatile membership privacy risk metric for machine learning. ArXiv:2112.02230
172 Jayaraman B, Evans D. Are attribute inference attacks just imputation? In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2022. 1569–1582
173 Tao L, Feng L, Wei H, et al. Can adversarial training be manipulated by non-robust features? In: Proceedings of the 36th International Conference on Neural Information Processing Systems, 2022. 26504–26518
174 Chen Y, Shen C, Shen Y, et al. Amplifying membership exposure via data poisoning. In: Proceedings of the 36th International Conference on Neural Information Processing Systems, 2022. 29830–29844
175 Tramèr F, Shokri R, San Joaquin A, et al. Truth serum: poisoning machine learning models to reveal their secrets. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2022. 2779–2792
176 Wang B, Yao Y, Shan S, et al. Neural cleanse: identifying and mitigating backdoor attacks in neural networks. In: Proceedings of the IEEE Symposium on Security and Privacy, 2019. 707–723
177 Pang L, Sun T, Ling H, et al. Backdoor cleansing with unlabeled data. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023. 12218–12227
178 Liu K, Dolan-Gavitt B, Garg S. Fine-pruning: defending against backdooring attacks on deep neural networks. In: Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses, 2018. 273–294
179 Adi Y, Baum C, Cisse M, et al. Turning your weakness into a strength: watermarking deep neural networks by backdooring. In: Proceedings of the 27th USENIX Conference on Security Symposium, 2018. 1615–1631
180 Li Y, Bai Y, Jiang Y, et al. Untargeted backdoor watermark: towards harmless and stealthy dataset copyright protection. In: Proceedings of the 36th International Conference on Neural Information Processing Systems, 2022. 13238–13250
181 Zhang J, Gu Z, Jang J, et al. Protecting intellectual property of deep neural networks with watermarking. In: Proceedings of the Asia Conference on Computer and Communications Security, 2018. 159–172
182 Lukas N, Jiang E, Li X, et al. SOK: how robust is image classification deep neural network watermarking? In: Proceedings of the 43rd IEEE Symposium on Security and Privacy, 2022. 787–804
183 Jin K, Zhang T, Shen C, et al. Can we mitigate backdoor attack using adversarial detection methods? IEEE Trans Dependable Secure Comput, 2022, 20: 2867–2881
184 Han S, Lin C, Shen C, et al. Interpreting adversarial examples in deep learning: a review. ACM Comput Surv, 2023, 55: 1–38
185 Sotgiu A, Pintor M, Biggio B. Explainability-based debugging of machine learning for vulnerability discovery. In: Proceedings of the 17th International Conference on Availability, Reliability and Security, 2022. 1–8
186 Steenhoek B, Rahman M, Sharmin S, et al. Do language models learn semantics of code? a case study in vulnerability detection. ArXiv:2311.04109
187 Liu Y, Tantithamthavorn C, Liu Y, et al. On the reliability and explainability of language models for program generation. ACM Trans Software Eng Methodol, 2024, 33: 1–26
188 Ribeiro M, Singh S, Guestrin C. "Why should I trust you?" Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016. 1135–1144
189 Lundberg S, Lee S. A unified approach to interpreting model predictions. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017. 4765–4774
190 Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017. 6000–6010
191 Wan Y, Zhao W, Zhang H, et al. What do they capture? A structural analysis of pre-trained language models for source code. In: Proceedings of the 44th IEEE/ACM 44th International Conference on Software Engineering, 2022. 2377–2388
192 Saad M, Sharma T. Naturalness of attention: revisiting attention in code language models. In: Proceedings of the IEEE/ACM 44th International Conference on Software Engineering: New Ideas and Emerging Results, 2024. 107–111
193 Harel-Canada F, Wang L, Gulzar M, et al. Is neuron coverage a meaningful measure for testing deep neural networks? In: Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020. 851–862
194 Sharma A, Hu Z, Quinn C, et al. Interpreting pretrained source-code models using neuron redundancy analyses. ArXiv:2305.00875
195 Karmakar A, Robbes R. What do pre-trained code models know about code? In: Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering, 2021. 1332–1336
196 Troshin S, Chirkova N. Probing pretrained models of source code. In: Proceedings of the 5th BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP, 2022. 371–383
197 Ma W, Zhao M, Xie X, et al. Unveiling code pre-trained models: investigating syntax and semantics capacities. ACM Trans Software Eng Methodol, 2024, 33: 1–29
198 Hernández López J, Weyssow M, Cuadrado J, et al. Ast-probe: recovering abstract syntax trees from hidden representations of pre-trained language models. In: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, 2022. 1–11
199 Shi E, Wang Y, Zhang H, et al. Towards efficient fine-tuning of pre-trained code models: an experimental study and beyond. In: Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, 2023. 39–51
200 Karmakar A, Robbes R. INSPECT: intrinsic and systematic probing evaluation for code transformers. IEEE Trans Software Eng, 2023, 50: 220–238
201 Rodriguez-Cardenas D, Palacio D, Khati D, et al. Benchmarking causal study to interpret large language models for source code. In: Proceedings of the IEEE International Conference on Software Maintenance and Evolution, 2023. 329–334
202 Nader Palacio D, Velasco A, Cooper N, et al. Toward a theory of causation for interpreting neural code models. IIEEE Trans Software Eng, 2024, 50: 1215–1243
203 Cito J, Dillig I, Murali V, et al. Counterfactual explanations for models of code. In: Proceedings of the 44th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, 2022. 125–134
204 Cito J, Dillig I, Kim S, et al. Explaining mispredictions of machine learning models using rule induction. In: Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2021. 716–727
205 Hajipour H, Yu N, Staicu C, et al. Simscood: systematic analysis of out-of-distribution generalization in fine-tuned source code models. In: Proceedings of Findings of the Association for Computational Linguistics: NAACL, 2024. 1400–1416
206 Rabin M R I, Hussain A, Alipour M A, et al. Memorization and generalization in neural code intelligence models. Inf Software Tech, 2023, 153: 107066
207 Hussain A, Rabin M, Ayoobi N, et al. Measuring impacts of poisoning on model parameters and neuron activations: a case study of poisoning CodeBERT. ArXiv:2402.12936

208 Hussain A, Rabin M, Alipour M. Measuring impacts of poisoning on model parameters and embeddings for large language models of code. In: Proceedings of the 1st ACM International Conference on AI-Powered Software, 2024. 1–8

209 Zhang Z, Zhang H, Shen B, et al. Diet code is healthy: simplifying programs for pre-trained models of code. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2022. 1073–1084

210 Wang L, Xie X, Du X, et al. Distxplore: distribution-guided testing for evaluating and enhancing deep learning systems. In: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2023. 68–80

211 Ahmed T, Yu D, Huang C, et al. Towards understanding what code language models learned. ArXiv:2306.11943

212 Rabin M, Hellendoorn V, Alipour M. Understanding neural code intelligence through program simplification. In: Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2021. 441–452

213 Shao C, Li G, Wu J, et al. Exploring semantic redundancy using backdoor triggers: a complementary insight into the challenges facing DNN-based software vulnerability detection. ACM Trans Softw Eng Methodol, 2024, 33: 1–28

214 Kanade A, Maniatis P, Balakrishnan G, et al. Learning and evaluating contextual embedding of source code. In: Proceedings of the 37th International Conference on Machine Learning, 2020. 5110–5121

215 Xie X, Chen H, Li Y, et al. Coverage-guided fuzzing for feedforward neural networks. In: Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering, 2019. 1162–1165

216 Xie X, Li T, Wang J, et al. NPC: neuron path coverage via characterizing decision logic of deep neural networks. ACM Trans Softw Eng Methodol, 2022, 31: 1–27

217 Xie X, Ma L, Xu J F, et al. Deephunter: a coverage-guided fuzz testing framework for deep neural networks. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2019. 146–157

218 AppSOC Research Labs. Report: DeepSeek-R1 LLM security testing. 2025. https://www.appsoc.com/collateral/appsoc-labs-report-deepseek-r1-security-testing

219 PointGuard AI Security Platform. Testing the DeepSeek-R1 model: a Pandora's box of security risks. 2025. https://www.pointguardai.com/blog/testing-the-deepseek-r1-model-a-pandoras-box-of-security-risks

220 Enkrypt AI/Euronews. "Harmful and toxic output": DeepSeek has "major security and safety gaps," study warns. 2025. https://www.euronews.com/next/2025/01/31/harmful-and-toxic-output-deepseek-has-major-security-and-safety-gaps-study-warns

221 Zhou K, Liu C, Zhao X, et al. The hidden risks of large reasoning models: a safety assessment of R1. ArXiv:2502.12659