Chasing Shadows:
Pitfalls in LLM Security Research

Jonathan Evertz*[†], Niklas Risse*[‡], Nicolai Neuer[§], Andreas Müller[¶], Philipp Normann[∥],
Gaetano Sapia[‡], Srishti Gupta[#], David Pape[†], Soumya Shaw[†], Devansh Srivastav[†],
Christian Wressnegger[§], Erwin Quiring[◇], Thorsten Eisenhofer[†], Daniel Arp[∥], Lea Schönherr[†]

[†] CISPA Helmholtz Center for Information Security   [‡] Max Planck Institute for Security and Privacy
[§] Karlsruhe Institute of Technology   [¶] Ruhr University Bochum   [∥] TU Wien
[#] Sapienza University of Rome   [◇] _fbeta

* Equal contribution

Abstract—Large language models (LLMs) are increasingly prevalent in security research. Their unique characteristics, however, introduce challenges that undermine established paradigms of reproducibility, rigor, and evaluation. Prior work has identified common pitfalls in traditional machine learning research, but these studies predate the advent of LLMs. In this paper, we identify *nine* common pitfalls that have become (more) relevant with the emergence of LLMs and that can compromise the validity of research involving them. These pitfalls span the entire computation process, from data collection, pre-training, and fine-tuning to prompting and evaluation.

We assess the prevalence of these pitfalls across all 72 peer-reviewed papers published at leading Security and Software Engineering venues between 2023 and 2024. We find that every paper contains at least one pitfall, and each pitfall appears in multiple papers. Yet only 15.7% of the present pitfalls were explicitly discussed, suggesting that the majority remain unrecognized. To understand their practical impact, we conduct four empirical case studies showing how individual pitfalls can mislead evaluation, inflate performance, or impair reproducibility. Based on our findings, we offer actionable guidelines to support the community in future work.

I. INTRODUCTION

The intersection of large language models (LLMs) and security has become a fast-growing and influential line of research. Their capacity for easy adaptation and deployment makes LLMs powerful instruments for security-critical applications, including vulnerability detection [1]–[4], code analysis [5]–[8], and automated code repair [9]–[12]. At the same time, it is crucial to understand their inherent capabilities for resisting attacks: models struggle to distinguish commands from data [13], to identify context-dependent sensitive content [14], [15], and are generally vulnerable to prompt injection [16] and jailbreak attacks [17]. For LLMs to be trusted in critical appli-

cations, both in research and in practice, their implementations must be rigorous, reproducible, and methodologically sound.

Prior work has identified common pitfalls in traditional machine learning research [18], [19], offering valuable guidance for designing sound experiments. However, these studies predate the emergence of modern LLMs. Language models have fundamentally reshaped the stages of the machine learning workflow and introduced new risks: data collection (e.g., poisoning via large-scale web scraping), pre-training (e.g., inadvertent data leakage), fine-tuning (e.g., reliance on synthetic data), prompt engineering (e.g., context sensitivity), and evaluation (e.g., unstable API behavior). As a result, previous studies [18], [19] do not fully capture the complexity and fragility of current LLM practices.

In this paper, we bridge this gap by identifying nine pitfalls that commonly occur in LLM security research. These pitfalls span the entire development pipeline of LLMs. They reflect both structural changes in how models are built and used, as well as the unique challenges introduced by the scale, opacity, and natural language interface of LLMs. Some pitfalls are entirely new, such as *model collapse* caused by training on synthetic data or unpredictable model behavior due to *prompt sensitivity*. Others represent familiar concerns, such as *data leakage* or *spurious correlations*, but their implications change and intensify in the context of LLMs.

To assess how widespread these pitfalls are, we conducted a prevalence study across all LLM-centric research papers (72 in total) published between January 2023 and December 2024 at leading conferences in Security (IEEE S&P, NDSS, ACM CCS, and USENIX Security Symposium) and Software Engineering (IEEE/ACM ICSE, ACM ISSTA, ACM FSE, and IEEE/ACM ASE). With a team of 15 researchers, we collaboratively developed detailed labeling guidelines and reviewed each paper independently using a systematic and uniform agreement process.

To our surprise, *every* paper in our study contains at least one of the nine identified pitfalls as part of its main contribution. Five pitfalls, namely Data Leakage (P3), Context Truncation (P6), Prompt Sensitivity (P7), Surrogate Fallacy
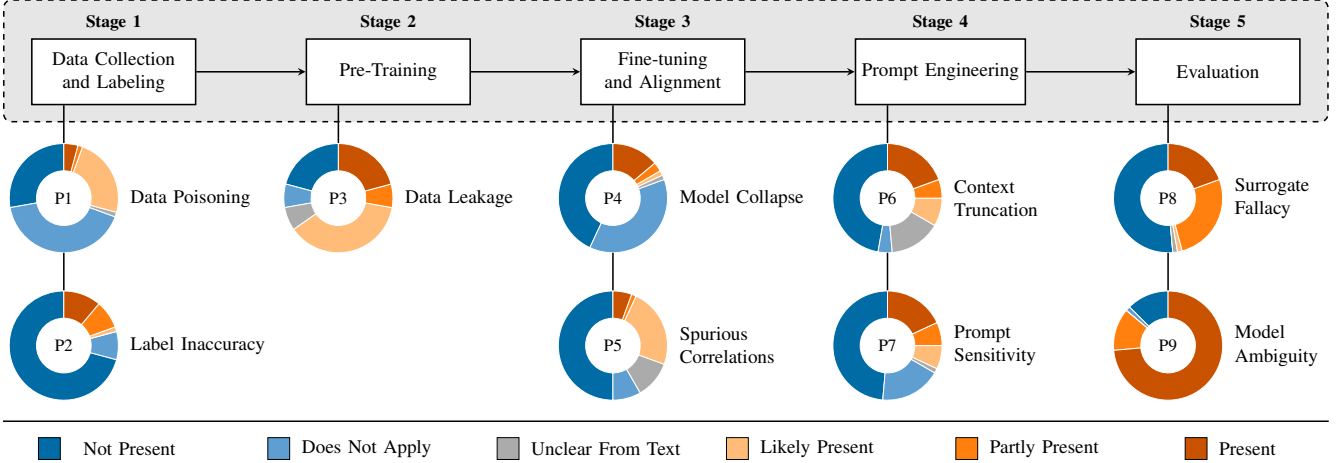
Large Language Model Pipeline

Fig. 1: Typical LLM pipeline, divided into its key stages. Each stage can introduce LLM-specific pitfalls that may distort evaluation, inflate reported performance, or undermine reproducibility. Colors indicate the prevalence of each pitfall according to our prevalence study (§III).

(P8), and Model Ambiguity (P9), appear in more than 20% of the papers. Prevalence also varies across research topics: papers on *Vulnerability Repair and Detection* show the highest average rate of pitfalls per paper (23–28%), while studies on *Fuzzing* and *Secure Code Generation* contain fewer pitfalls on average (15–18%).

Beyond identifying prevalence, we conduct four case studies that experimentally demonstrate how the identified pitfalls can mislead evaluation, inflate performance, or hinder reproducibility. Concretely, we examine the impact of (i) Model Ambiguity—snapshot versions and quantization affect robustness against jailbreaks and significantly alter precision/recall, thereby affecting reproducibility; (ii) Data Leakage—leaking 20% of test data into the fine-tuning process raises the F1 score by $\approx$ 0.08–0.11 with increases that grow almost linearly as leakage increases; (iii) Context Truncation—about 49% of vulnerable functions exceed common context windows of 512 tokens (29% > 1024 tokens), removing essential information and distorting evaluation; (iv) Model Collapse—recursive self-training in code generation increases perplexity across generations, leading to degradation and instability.

Based on our results, we provide concrete guidelines and recommendations for each identified pitfall. *Importantly, our goal is not to assign blame.* Pitfalls can occur in carefully conducted research, often without being explicitly recognized. Our aim is to support the community by raising awareness and offering actionable recommendations.

Contributions. We make the following key contributions:

- **Identification of LLM pitfalls.** We identify nine pitfalls that frequently arise in LLM-based security research spanning the entire computation pipeline from data collection to evaluation. We describe each pitfall, explain why it occurs, and discuss its potential impact on validity and reproducibility.

- **Prevalence study.** We assess the prevalence of all nine pitfalls across 72 peer-reviewed papers published between January 2023 and December 2024 at leading Security and Software Engineering venues. We find that every paper suffers from at least one pitfall.

- **Impact analysis.** We conduct four case studies that empirically demonstrate how individual pitfalls can distort evaluation, inflate performance metrics, or compromise reproducibility.

- **Guidelines and recommendations.** We present guidelines and recommendations for each identified pitfall and maintain a living appendix containing up-to-date information on best practices for preventing pitfalls, accessible at https://llmpitfalls.org.

We provide all code, datasets, and complete reproduction instructions at https://github.com/dormant-neurons/llm-pitfalls.

Additionally, an extended version of this paper with detailed information about pitfall definitions, guidelines, and information for used LLMs in its appendix is available at: https://doi.org/10.48550/arXiv.2512.09549.

II. PITFALLS ACROSS THE LLM PIPELINE

Research on pitfalls in machine learning is not new. Prior work has systematically analyzed common failure modes in traditional pipelines [18], [19]. While related in spirit, the pitfalls we investigate arise from the specific characteristics of LLM research. Our aim is not to replicate existing analyses, but to complement them with a targeted examination of challenges that are unique to or substantially intensified in LLMs.

LLMs have significantly changed the way machine learning systems are trained, adapted, and evaluated. Their development pipelines involve multiple interdependent stages: large-scale data collection and labeling, pre-training, fine-tuning, prompt

design, and evaluation. In the following sections, we outline the structure of this pipeline and describe the potential practical risks that can arise at each stage. This discussion forms the foundation for the nine pitfalls we identify. Figure 1 illustrates the overall pipeline and the corresponding pitfalls.

A. Stage 1: Data Collection and Labeling

LLMs require large-scale datasets to train effectively, often relying on content scraped from the Internet, which is far more extensive and noisy than the more curated datasets used in classical machine learning.

P1—Data Poisoning via Internet Scraping. The sheer scale of this data makes curation increasingly difficult, creating favorable conditions for *Data Poisoning*. Without sufficient precautions, data scraped from the Internet opens the door to data poisoning attacks, where malicious or biased content can be subtly inserted into the training data without detection. Although not new to machine learning, the scale of LLMs exacerbates this risk. Platforms like GitHub or Reddit, where anyone can anonymously upload data, make it especially difficult to ensure quality and safety at scale.

P2—LLM-generated Label Inaccuracy. At the same time, the demand for labeled data during fine-tuning and evaluation has led to the widespread use of models themselves to generate labels, a practice known as LLM-as-a-judge [20]. Although this approach can address the problem of generating labels in scenarios in which human-curated labeling is time-intensive or costly, it also introduces the risk of *Label Inaccuracy*.

The reliability of experimental results depends heavily on the quality of data and labeling. Since learning-based methods rely on accurate labels, any label errors or instabilities can degrade performance. This is particularly relevant when LLMs are used for automated annotation, where outputs may appear correct yet still contain subtle flaws.

B. Stage 2: Pre-Training

LLMs are typically pre-trained on large-scale datasets to capture general language patterns. Because these corpora are often assembled from broad internet scrapes with limited transparency, they increase the risk of data leakage.

P3—Data Leakage. Many model providers no longer disclose detailed information about the data used for training. For instance, GPT-4 was trained on "a variety of licensed, created, and publicly available data sources" [21], [22], without further specification. This lack of transparency increases the risk of *Data Leakage*, where evaluation data may inadvertently overlap with training inputs.

While creating traditional machine learning models with disjoint data splits is straightforward, the nature of LLMs complicates this. As LLMs require massive datasets, fine-tuning foundational models is a widespread practice. Foundational models may have been trained on (a subset of) samples from the test set. This risk is particularly high in LLMs, where many datasets overlap with pre-training sources like GitHub, Wikipedia, and Reddit [23], [24].

C. Stage 3: Fine-tuning and Alignment

During fine-tuning or alignment, LLMs are adapted for specific downstream tasks and user-facing behaviors. In this stage, models become particularly susceptible to issues where evaluation data may inadvertently overlap with pre-training data, as well as the reliance on synthetic or LLM-generated data.

P4—Model Collapse via Synthetic Training Data. Fine-tuning and alignment frequently rely on data that was itself generated by LLMs, introducing the risk of *Model Collapse* [25].

LLMs require vast amounts of data, and this need has increased substantially. To address data scarcity, synthetic data generated by other LLMs is frequently used. However, training on such data can lead to less diverse outputs, higher error rates, and model collapse over generations. While countermeasures exist, they are difficult to implement in practice [26]–[28].

P5—Spurious Correlations/Unrelated Features. Additionally, *Spurious Correlations*, already known as an issue in traditional ML, becomes especially problematic for LLMs because their massive parameter space allows them to memorize and exploit subtle, non-causal patterns.

Spurious correlations are artifacts that correlate with task labels but are not actually related to the underlying task. Such features may arise from selection bias in the training data or from shortcuts the model learns using unrelated patterns. This can lead to misleading performance and poor generalization.

D. Stage 4: Prompt Engineering

Prompting has emerged as a new form of control in LLMs. It effectively acts as a tunable hyperparameter that controls the model's behavior, and its design can influence the model's outputs and overall performance.

P6—Context Truncation. An LLM's context size refers to the maximum number of tokens it can process at once. Because LLMs are stateless, all relevant information must be included in the current context. If inputs exceed this limit, they are truncated, potentially omitting critical information and reducing performance.

P7—Prompt Sensitivity. Prompting also introduces the issue of *Prompt Sensitivity*, where minor changes in phrasing can lead to drastically different outputs and where different models can have distinct prompt preferences.

Language models are often fine-tuned to follow instruction-based inputs with specific formatting styles. While this allows the language model to adapt to desired tasks during runtime, the performance and functionality of the model also rely on the quality of the instructions and the correct input format. This

results in differences across evaluations if inputs are formatted for a specific LLM but evaluated on other models, or are not expressive enough for the task [29], [30].

E. Stage 5: Evaluation

As LLMs are often accessed through APIs and web interfaces, with several distinct model versions that differ in architecture, parameter count, or quantization. This ambiguity creates risks in reproducing results, as identifying the precise model instance becomes difficult or even infeasible.

P8—Proxy/Surrogate Fallacy. A single model name, such as ChatGPT, may refer to multiple underlying snapshots with different architectures, sizes, or quantization levels. This creates the risk of the *Proxy/Surrogate Fallacy*, where conclusions are drawn from models that are not representative of those actually used.

New LLMs are released frequently, driving the race for the "most state-of-the-art" model. These models vary in size and task specialization. Although models of the same class often share architecture and training data, differences in size or context window can significantly affect performance and behavior. Discrepancies are even greater across model classes, for example, between the open-source Llama models [31]–[33] and the commercial GPT family [34]. Without experimental validation, it is generally not possible to make claims between different model sizes and families.

P9—Model Ambiguity. The ambiguity of different architectures, sizes, or quantization levels also leads to *Model Ambiguity* risks, where determining the exact model instance used to reproduce a result is difficult or even impossible.

Both open-source models on platforms like Huggingface [35] and proprietary models like ChatGPT [34], Claude [36], or Gemini [37] are regularly updated. Minor updates are often noted only in changelogs and may not result in a new major version. A single model specifier can therefore refer to different internal versions. Because even small changes in tokenizers or system prompts can affect model behavior, specifying the exact snapshot (e.g., endpoint version, commit ID, or access date) is essential. For open-source models, the quantization level is often an additional source of variation and should be specified (cf., §IV-A).

III. PREVALENCE OF PITFALLS

Having identified nine pitfalls specific to the LLM computation pipeline, we now assess how frequently these issues appear in current LLM security research. We begin by outlining the paper selection process that underpins our study (§III-A) as well as the methodology used to evaluate each paper (§III-B). We then present the results per pitfall (§III-C), followed by a broader discussion (§III-D).

A. Paper Collection

As the basis for our study, we consider papers published between January 2023 and December 2024 at leading conferences in Security and Software Engineering.
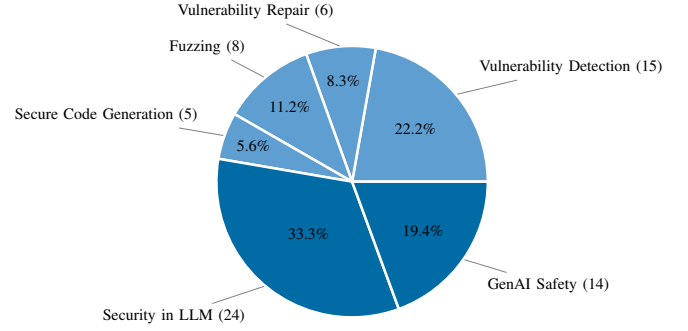

Fig. 2: Distribution of the 72 selected papers across topics. The top half, light blue segments (●) represent research that uses LLMs to address security problems. The bottom half, dark blue segments (●) correspond to research focused on the security and safety of LLMs themselves.

- For Security, we consider the IEEE Symposium on Security and Privacy (IEEE S&P), the Network and Distributed System Security Symposium (NDSS), the ACM Conference on Computer and Communications Security (ACM CCS), and the USENIX Security Symposium.

- For Software Engineering, we consider the IEEE/ACM International Conference on Software Engineering (IEEE/ACM ICSE), the ACM International Conference on the Foundations of Software Engineering (ACM FSE), the ACM SIGSOFT International Symposium on Software Testing and Analysis (ACM ISSTA), and the IEEE/ACM International Conference on Automated Software Engineering (IEEE/ACM ASE).

For all conferences, we scraped the full set of papers and applied a two-stage filtering process to identify relevant studies, as detailed next.

Selection Criteria. To identify relevant papers, we establish two primary selection criteria centered on the use of LLMs in security-relevant contexts. Specifically, papers were considered in scope if they (1) evaluate the security or safety of LLMs, or (2) apply LLMs to security-related tasks such as vulnerability detection or secure code generation. Importantly, the use of the LLM had to be integral to a paper's methodology, meaning the core contribution would not have been possible in its current form without them.

To guide the search, we define a list of LLM-relevant keywords: *LLM*, *LLMs*, *large language model*, *language model*, *GPT*, *ChatGPT*, *transformer*, *pre-trained*, *foundation model*, *prompt*, *learning*. We applied these keywords to search the titles and abstracts of all papers published in the selected venues using multiple specialized tools for literature search: IEEE Xplore Advanced Search[1], ACM DL Advanced Search[2], and Google Scholar Advanced Search[3].

---

[1]IEEE Xplore Advanced Search
[2]ACM DL Advanced Search
[3]Google Scholar Advanced Search

This process yielded over 1.000 candidate papers, which we manually reviewed for relevance. As a first step, we excluded papers that clearly did not fit our scope. For example, we removed the software engineering paper *Large Language Models are Few-Shot Summarizers: Multi-Intent Comment Generation via In-Context Learning* [38], which focuses on code summarization without any direct security aspect. Second, all remaining abstracts, and when necessary, the full texts, were manually reviewed to assess whether each paper met the inclusion criteria.

Selected Papers. Our collection process resulted in a final set of 72 papers. These papers span a wide range of topics within the field of language models and security: *Security in LLM* [39]–[62], *Vulnerability Detection* [1]–[3], [63]–[74], *Generative AI Safety* [75]–[88], *Fuzzing* [4], [89]–[95], *Secure Code Generation* [96]–[99], and *Vulnerability Repair* [9]–[12], [100], [101]. Figure 2 provides an overview of the distribution of papers across these categories.

B. Reviewing Methodology

To guide the review process, we developed a detailed set of guidelines to ensure consistency across reviewers. These guidelines define how to identify each pitfall and provide instructions on interpreting relevant cues within the papers. To refine the guidelines, we conducted a pilot review on a small subset of papers. This process helped clarify ambiguous cases and improve reviewer instructions. An extended version of our paper with the final guidelines in its appendix is available at: https://doi.org/10.48550/arXiv.2512.09549.

Labeling Scheme. For each of the nine pitfalls, reviewers assigned one of several labels reflecting its applicability and presence. A pitfall is marked as *Does not apply* (●) if it is outside the scope for the particular paper, or as *Unclear from text* (●) when the paper lacks sufficient detail to support a clear judgment. If the pitfall is applicable but not present, it is labeled as *Not present* (●). We use *Partly present* (●) to indicate that the pitfall affects only a part of the paper's methodology. A pitfall was marked as *Present* (●) when it appears throughout the paper. Finally, in case there is a high likelihood of a pitfall's presence but no explicit evidence, we use *Likely present* (●). For example, take the case of potential data leakage where the evaluation dataset was publicly available online before the training-data cutoff of a model that has been trained on web-scale data. In such cases, we cannot prove that the evaluation data were included during training, but their presence is highly plausible due to timing and accessibility. This process is also summarized in the appendix of the extended version of this paper.

It is important to note that some pitfalls may be difficult or impossible to avoid. Therefore, for any pitfall labeled as *Likely present*, *Partly present*, or *Present*, we also annotated whether the presence of the pitfall was discussed in the paper. This distinction helps recognize good practice in transparently acknowledging potential limitations or contextualizing the relevance of a pitfall's presence. For example, *Label Inaccuracy*

(P2) may be unavoidable in large-scale evaluations where labeling is expensive, or *Data Leakage* (P3) may be difficult to address due to limited access to the original training data of proprietary models.

Reviewing Phases. The review was conducted by a team of fifteen researchers with backgrounds in security, machine learning, and software engineering. Every paper was assigned to exactly two reviewers, aiming to match papers with reviewers whose expertise aligns closely with the paper's topic. To ensure consistency and rigor throughout the review process, we structured the evaluation into three phases:

Phase 1: Individual Reviews. In the first phase, each reviewer independently assessed their assigned papers using the pre-defined guidelines. Reviewers were blinded to each other's assessments to ensure independent judgment.
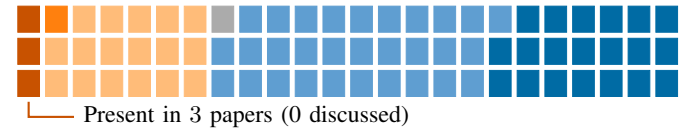
Phase 2: Review Discussions. In the second phase, reviewers discussed each paper to resolve differences from their initial assessments. Disagreements were clarified through 1-on-1 sessions or, when needed, group discussions (see Phase 3). For each pitfall, we recorded both the initial and final decisions.

Phase 3: Group Discussions. For cases where consensus could not be reached, we escalated the discussion to a larger group. These sessions included ten of the fifteen reviewers and served to resolve especially ambiguous or borderline cases.
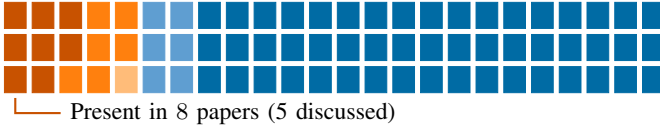
C. Pitfall Prevalence

We now present the results of our prevalence study. As discussed above, we begin by examining each pitfall individually, summarizing how frequently it occurs across the reviewed papers and highlighting potential implications. In the following section (§III-D), we broaden the discussion to consider general patterns and shared challenges across pitfalls.

P1—Data Poisoning via Internet Scraping. A dataset used to train a model is collected from the internet without strategies to verify the integrity and safety of the data [102].
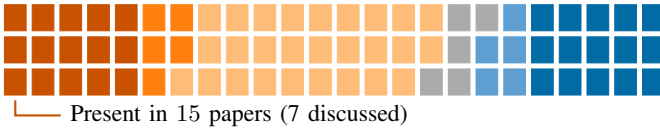

└─ Present in 3 papers (0 discussed)

Results & Implications. Although this pitfall was *Present* in 4.2% (3) and *Likely present* in 23.6% (17) of the papers, none acknowledged or discussed the risk of training on potentially poisoned internet data. This is especially concerning given recent work demonstrating that even minimal amounts of poison can compromise LLM behavior [103]–[105]. In security-relevant settings, this can result in unsafe code suggestions or behavior misalignment triggered by crafted inputs.

P2—LLM-generated Label Inaccuracy. LLMs are used to annotate data with certain labels via classification or LLM-as-a-judge procedures without further validation of correctness.
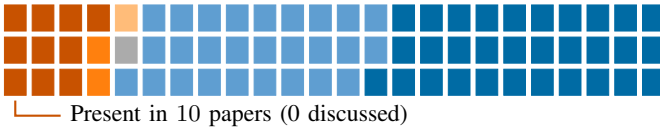
Present in 8 papers (5 discussed)

Results & Implications. This pitfall was *Present* in 15.3% (8) of the papers (five of which discussed the pitfall), *Partly present* in 8.3% (6) of the papers (four of which discussed it), and *Likely present* in 1.4% (1) of the papers. While not among the most common pitfalls overall, 60% of relevant cases showed a relatively high awareness by discussing the issue. Nonetheless, unvalidated LLM-generated labels can distort results and lead to faulty conclusions. This is especially relevant in areas like jailbreak detection, where LLMs are often used as judges and external validation is critical to ensure correctness.

P3—Data Leakage. An LLM is trained or fine-tuned with data that is normally not available in practice, or the training data is contaminated with possible test data [106].

Present in 15 papers (7 discussed)

Results & Implications. Data Leakage is highly prevalent: 65.2% (47) of the papers either contain it or are likely affected by it. Specifically, Data Leakage is *Present* in 20.8% (15) of the papers (of which seven explicitly discuss the pitfall), *Partly present* in 6.9% (5) of the papers (two of which discussed the pitfall), and *Likely present* in another 37.5% (27) of the papers (one of which discussed the pitfall). This raises concerns about inflated performance metrics, as models may memorize rather than generalize. Even state-of-the-art LLMs like GPT have suffered from such issues in tasks like code completion and vulnerability detection [107].

P4—Model Collapse via Synthetic Training Data. An LLM is trained on data that is generated by other language models, risking an amplification of bias and degradation of data quality [25], [108], [109].

Present in 10 papers (0 discussed)

Results & Implications. Although this pitfall was *Present* in 13.9% (10) of the papers, none of the papers in our study acknowledged or discussed the risk of using synthetic LLM-generated training data. Shumailov et al. have demonstrated a decrease in model performance when trained on synthetic data, making the performance tied to the LLMs training data [25]. Thus, training on LLM-generated data risks inheriting specific behaviors, biases, and errors [110]. This
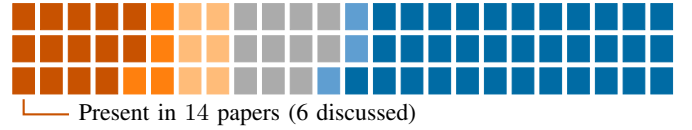
can undermine generalization, particularly in tasks such as code generation, potentially distorting results and degrading performance. Notably, for 37.5% (27) of papers, this pitfall *Does not apply*, as they do not involve model training.

P5—Spurious Correlations/Unrelated Features. The LLM adapts to unrelated artifacts from the problem space instead of generalizing onto the actual task [111].

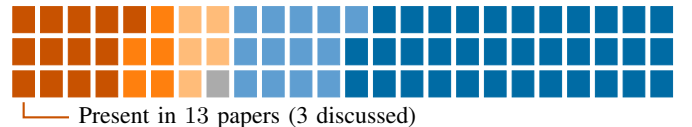Present in 4 papers (1 discussed)

Results & Implications. This pitfall appears in 31% of papers: *Present* in 5.6% (4) of the papers (one of which discussed the pitfall), *Likely present* in 23.6% (17) of the papers, and *Partly present* in 1.4% (1) of the papers (which discussed it). Notably, it is discussed explicitly in only two papers. Due to the complexity of tasks and potentially black-box settings in which LLMs are deployed, Spurious Correlations often remain unidentified. This not only distorts experimental results but also leads to drawing false conclusions and using LLMs in scenarios for which they are not suited. In the security domain, for example, LLMs for vulnerability detection were unable to distinguish between vulnerable functions and the same functions that had been patched. The dependence on unrelated features led to top scores on benchmarks, even after removing the code structure itself [112].

P6—Context Truncation. The LLM's context size is not large enough for its intended task, and the input needs to be truncated.
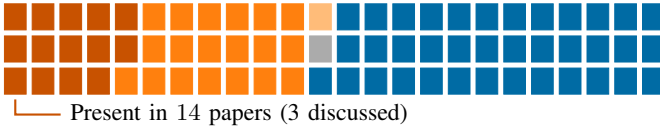
Present in 14 papers (6 discussed)

Results & Implications. While not as prevalent as Data Leakage, context limitations remain a notable pitfall: 33.3% papers (24) were affected, either marked as *Present* in 19.4% of the papers (14, 6 of which discussed the pitfall), *Partly Present* in 5.5% of the papers (4, all of which discussed the pitfall), or *Likely present* in 8.3% of the papers (6, none of which discussed the pitfall). This is especially relevant in tasks like vulnerability detection [112] or prompt-based attacks [17], where long inputs are common. A limited context size can cause models to overlook key information or distort evaluation results.

P7—Prompt Sensitivity. The prompt used to instruct the language models is fixed for all models and experiments, or is not expressive enough for the given task. This allows for prompt-based fluctuations in evaluations.
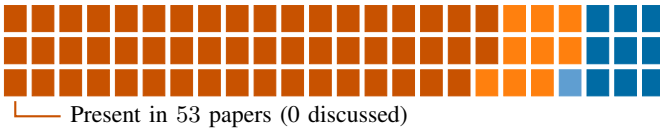
Present in 13 papers (3 discussed)

Results & Implications. Prompt Sensitivity affected 23 papers (31.9%) in our analysis: *Present* in 18.1% (13) of the papers (3 of which discussed the pitfall), *Likely present* in 6.9% (5) of the papers, and *Partly present* in 6.9% (5) of the papers (2 of which discussed it). If prompt formatting is not adapted to the model, e.g., by using incorrect delimiters or generic instructions, this can reduce model performance or lead to misleading evaluations. In tasks such as jailbreak detection or alignment testing, vague prompts (e.g., "You are a helpful AI assistant") may under represent a model's true capabilities or vulnerabilities. As a result, evaluations might miss important failure modes or overstate alignment, leading to flawed comparisons across models [113].

P8—Proxy/Surrogate Fallacy. Findings from specific LLMs are often inappropriately generalized to other, often larger and more capable models or even to entire classes of language models, without sufficient empirical validation.

└── Present in 14 papers (3 discussed)

Results & Implications. Surrogate Fallacy is highly prevalent, appearing in 47.2% (34) of the papers: *Present* in 19.4% (14) papers (3 of which discussed the pitfall), *Partly present* in 26.4% (19) of the papers (2 of which discussed it), and 1 paper is marked as *Likely present*. Due to the distinct differences in behavior, claims about specific LLMs do not generalize to other models. Especially in security, where attacks and defenses hinge on specific model behavior, extrapolating results from smaller open-source models to larger proprietary ones is methodologically flawed and risks overstating generalizability.

P9—Model Ambiguity. The model details are insufficient for precise identification, preventing reproducibility (e. g., missing model ID, snapshot, commit ID, quantization level).

└── Present in 53 papers (0 discussed)

Results & Implications. This is the most prevalent pitfall in our study: It was *Present* in 73.6% (53) of all papers, and *Partly present* in 12.5% (9). Notably, none of them acknowledged or discussed this issue, indicating a low level of community awareness. The lack of precise versioning details makes it nearly impossible to reproduce results reliably. Vendors frequently update LLMs to improve reliability or harden them against attacks, changes that can alter behavior without changing the model name. Similarly, quantization variants in open-source models can produce different outputs due to reduced precision. Microsoft highlighted this in the case of *Phi3mini4k*, where a major update significantly improved benchmarks; they published results for both versions and documented the relevant commit [114], [115].

D. Main Findings

Every pitfall occurs in multiple papers, and every paper contains at least one pitfall that is either fully or partly present (cf. the appendix of the extended paper version for a side-by-side comparison of all pitfalls). The most common issues, present in over 20% of papers, are *Data Leakage (P3)*, *Context Truncation (P6)*, *Model Ambiguity (P9)*, *Prompt Sensitivity (P7)*, and *Surrogate Fallacy (P8)*. We analyze the impact of four of these in §IV.

Furthermore, for three pitfalls, namely *Data Leakage (P3)*, *Data Poisoning (P1)*, and *Spurious Correlations (P5)*, the *Likely present* label applies to more than 20% of papers, which means that the pitfall was probably present despite the absence of explicit evidence.

> **Finding.** *Every paper in our study contains at least one pitfall that is either fully or partly present.*

Discussed vs. Not Discussed. Overall, only 15.71% of pitfalls rated at least *Likely present* (*Likely present*, *Partly present*, or *Present*) were discussed by the authors of the papers. The three most frequently discussed pitfalls were *Label Inaccuracy (P2)* with 60.0%, *Context Truncation (P6)* with 41.67%, and *Data Leakage (P3)* with 21.28% of cases being explicitly addressed. In contrast, some pitfalls received no discussion at all: *Model Collapse (P4)*, *Data Poisoning (P1)*, and *Model Ambiguity (P9)*, indicating a lack of awareness of these issues.

> **Finding.** *Only 15.71% of pitfalls are discussed, with three pitfalls (Model Collapse via Synthetic Training Data (P4), Data Poisoning via Internet Scraping (P1), and Model Ambiguity (P9)) remaining entirely undiscussed across all papers in our study.*

Topics. We further analyze the prevalence of pitfalls across the six research topics defined in our paper selection process. Figure 3 shows the average percentage of pitfalls per paper within each topic. This reflects the proportion of pitfalls labeled as *Present*, including those where the pitfall was discussed.

The results show notable differences across topics. *Vulnerability Repair* papers stand out with the highest average pitfall presence (27.78%), followed by *Vulnerability Detection* (23.70%), indicating a high density of pitfalls in work that uses LLMs to find or fix software vulnerabilities. At the other end, *Fuzzing* exhibits the lowest pitfall presence (15.28%). *Secure Code Generation* averages 17.78%, suggesting this subfield is comparatively more robust with respect to the pitfalls we assessed. Research under *Security in LLMs* (19.91%) and *GenAI Safety* (19.84%) are close to 20%.
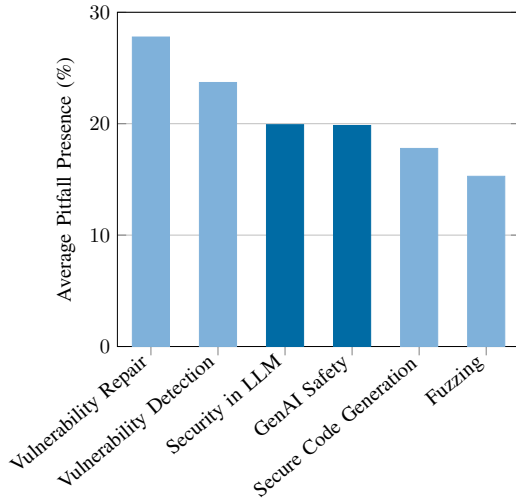
Fig. 3: Percentage of pitfalls labeled as *Present* (including cases where the pitfall was discussed), averaged over all pitfalls and papers within each topic. Light blue bars (•) represent research that uses LLMs to address security problems. Dark blue (•) bars correspond to research focused on the security and safety of LLMs themselves.

> **Finding.** *LLM-based research on* Vulnerability Detection *and* Vulnerability Repair *has the highest average number of pitfalls, whereas* Secure Code Generation *and* Fuzzing *papers tend to avoid such issues more consistently.*

IV. IMPACT ANALYSIS

Having assessed the prevalence of pitfalls, we now examine their potential impact in greater detail. To this end, we focus on four of the most prevalent pitfalls identified in §III, each of which appears in at least 20% of the papers. Specifically, we examine the impact of Model Ambiguity and the Surrogate Fallacy (§IV-A), the influence of Data Leakage on experimental results (§IV-B), and the limitations that Context Truncation imposes on model capabilities (§IV-C).

Together, these pitfalls illustrate the broader risks they pose to the integrity and reliability of LLM research. Another issue that appears in over 20% of the reviewed papers is Prompt Sensitivity. While important, Prompt Sensitivity has already been extensively studied in prior work (e.g., [29], [113]). For this reason, we do not provide a separate analysis here. Instead, we consider Model Collapse (§IV-D), which has received little attention in the security context so far but has recently been shown to degrade model reliability in the text domain [25]. We argue that understanding the potential implications for security research is necessary, especially as fine-tuned or retrained models become increasingly common.

A. Model Ambiguity & Surrogate Fallacy

The most widespread issue in our analysis is Model Ambiguity, which is present in 73.6% of all papers. To examine the potential impact of this pitfall on experimental results, we first

conduct a targeted evaluation of hate speech detection using state-of-the-art proprietary LLMs across multiple snapshots of the same base model. We then evaluate the robustness of proprietary models against prompt-based attacks across different model snapshots, as well as that of open-source models with varying levels of quantization. In this context, we also examine the impact of another common pitfall, the Surrogate Fallacy, which appears in 47% of the papers.

1) Hate Detection: In the first experiment, we analyze the impact of (missing) LLM version or snapshot information for the example of hate speech detection. To this end, we revisit a recently proposed method to prevent waves of hateful comments that often build up for a particular topic, as happened, for instance, during the COVID-19 pandemic on $\mathbb{X}$ (formerly Twitter) [116]. The core idea is to use an LLM to benefit from its enhanced reasoning capabilities, as detecting hate with all its nuances and newly introduced derogatory terms is a challenging task. Given a text, the LLM is instructed to answer a series of questions designed to guide its reasoning process.

Experimental Setup. We re-implement the experiments from Vishwamitra et al. [116] with their pre-labeled dataset of hateful and normal tweets from $\mathbb{X}$. This dataset contains tweets from three polarizing topics termed hate waves (COVID-19 pandemic, US Capitol insurrection, and Russia's invasion of Ukraine). The COVID-19 pandemic is further divided into subtopics, from which we focus on the "vaccine" subtopic. The calibration and testing for each topic are done separately, with each hate wave divided into four quarters. For simplicity, we calibrate the detection method using examples from the first quarter and test on the second quarter of each wave. The original publication stated the use of GPT-4. Thus, we test three GPT-4 snapshots available at OpenAI at the time of writing.

The detection is done by extracting keywords using Key-BERT and testing the novelty with NLTK's WordNet. Detailed specifications for all models can be found in the appendix of the extended version of this paper. We then provide the text and the extracted targets and terms to the LLM *gpt-4.1-mini-2025-04-14* and instruct it to check which keywords are a target or derogatory term. Equipped with these new targets and terms, the prompt template from the original work [116] is used, which instructs an LLM to answer a series of questions to determine if a text is hateful or not. The extracted targets and terms are appended at the end of the prompt. Finally, we evaluate the LLM on the test set to decide on each posting and record this as its final decision.

Results. Table I demonstrates that the detection performance varies considerably across the different GPT snapshots.

An analysis of the LLM outputs suggests a shift in how different AI models classify hate speech. We exemplify this using the tweet "Say it to my face, you tide pod eating, unvaccinated fuck", which was labeled as *hate* in the dataset. The *gpt-4-0613* model identifies derogatory and insulting words directed at anti-vaxxers and therefore categorizes the text as identity

TABLE I: Model Ambiguity in Hate Detection

Model	Accuracy	Precision	Recall
gpt-4-0613	88.70 %	87.07 %	82.05 %
gpt-4-0125-preview	77.16 %	95.52 %	41.03 %
gpt-4.1-2025-04-14	82.45 %	88.07 %	61.54 %

hate. In contrast, *gpt-4.1-2025-04-14* also recognizes the text's insulting content but determines that the language does not meet the threshold for identity hate, as it does not incite violence or hatred. The model considers the text as toxic only. In general, this observation is also reflected in the precision and recall rates in Table I. The models *gpt-4-0125-preview* and *gpt-4.1-2025-04-14* flag fewer tweets as hateful due to the higher threshold for hate, resulting in a significant drop in the recall rate. At the same time, the precision increases due to the stricter detection. While it is likely that such behavior can be adjusted through more tailored prompting, our aim with this experiment is to highlight the considerable, underestimated impact of the LLM version.

2) LLM Robustness: We continue by evaluating the robustness of several LLMs against a suite of prompt-based attacks, including prompt injections, adversarial suffixes, and complex jailbreaks [14]. To do this, we adopt the *secret-key game* introduced by Evertz et al. [14]. In this setup, the model is given a *secret key* in its system prompt, along with explicit instructions to keep the key confidential. The attack then attempts to elicit the secret from the model. If the model outputs the key as part of its response, the attack is considered successful. Each model is evaluated over 1,000 attack attempts, distributed evenly across 16 attack types. To minimize bias, system prompts are sampled from a set of 1,000 randomly selected variations.

We first examine proprietary models across multiple snapshots. As in the previous case, it is common to refer to these models by general handles (e.g., GPT-4o), but these labels may internally refer to different model snapshots. As of the time of writing, we evaluate all available snapshots of OpenAI's GPT-3.5 Turbo, GPT-4, GPT-4 Turbo, and GPT-4o.

In contrast to proprietary models, open-source models are typically not accessed through APIs and therefore do not expose versioned snapshots under the same identifier. Instead, on platforms such as Huggingface [35] and Ollama [117], they are provided in different variations of quantization, and older snapshots can be accessed via their commit history. To complement the snapshot-based evaluation of proprietary models, we analyze how quantization affects robustness in open-source models. We test various quantization levels (2-bit through 8-bit) as well as the original unquantized snapshots. We evaluate three models (CodeLlama-7b [118], Llama-2-7b [31], and Llama3.1 8b [32]) and two different inference engines (Ollama [117] and TheBloke [119]). Detailed specifications for all models are listed in the appendix of the extended version of this paper.

Results. Table III presents the results for different snapshots

of OpenAI's GPT. We observe that different snapshots lead to meaningful differences in robustness. In particular, GPT-3.5 Turbo exhibits significant variability across snapshots, though smaller but still notable differences are visible for the other models as well. The effect of different quantization levels is summarized in Table II. Interestingly, models with stronger quantization, such as 2-bit and 3-bit, appear more vulnerable to prompt-based attacks. Additionally, the results indicate a difference between inference engines. Note that these results not only demonstrate the potentially severe impact of the Model Ambiguity pitfall, but also serve as an example for the Surrogate Fallacy, since they show that findings do not necessarily transfer to other and particular larger models.

Overall, these findings highlight the importance of reporting detailed model information, including snapshot identifiers and quantization parameters. Such details are essential for the reproducibility and interpretability of experimental results. The observations are further supported by recent disclosures from Microsoft, showing that performance-relevant updates are being applied to their hosted models on Huggingface. For example, the *Phi 3 mini 4k instruct* model received an update together with a full disclosure about its new performance scores[4].

B. Data Leakage

We continue with the analysis of another very prevalent pitfall, Data Leakage, which occurs in 57% of all papers we considered. LLMs are typically trained on broad snapshots of public internet data collected up to a fixed cutoff date [120]. This practice introduces a critical concern: public benchmark datasets used to evaluate model performance may have been included in the training data, thereby undermining the validity of such evaluations.

To examine this issue, we focus on vulnerability detection as a representative task within LLM security research. It is widely studied, relies on well-known datasets, and is commonly used to benchmark the capabilities of code-oriented LLMs. We first conduct a controlled experiment designed to simulate leakage under lab conditions, followed by an empirical analysis of commercial models.

1) Leakage in Lab Setting: We consider three widely used vulnerability detection datasets: *Devign* [121], *DiverseVul* [122], and *PrimeVul* [123]. For each dataset, we construct splits for training (60%), validation (20%) and testing (20%), and fine-tune a CodeT5+ [124] model. Finally, we evaluate each model on the test split to obtain a baseline F1-score.

To simulate leakage, we fine-tune five additional models per dataset for 10 epochs, each trained on an increasingly larger fraction of the test set added to the training data (20%, 40%, 60%, 80%, 100%). We deliberately ignore the effects of limited context window size (512 tokens) and potential spurious correlations, as these factors apply equally across all data splits and do not affect the relative comparison. Full

---

[4]https://huggingface.co/microsoft/Phi-3-mini-4k-instruct; see Release Notes and June 2024 Update

TABLE II: Attack success rates (ASR) across different quantization levels, inference engines (if applicable), and model architectures. Each configuration is evaluated using 16 prompt-based attacks, evenly distributed across a total of 1,000 trials.

Model	Quantization						
	2-bit	3-bit	4-bit	5-bit	6-bit	8-bit	No Quant.
CodeLlama 7b (Ollama)	69.52 %	40.95 %	15.49 %	18.61 %	15.69 %	14.29 %	18.21 %
CodeLlama 7b (TheBloke)	67.35 %	70.41 %	58.16 %	62.24 %	61.22 %	63.27 %	19.39 %
Llama 2 7b (Ollama)	1.61 %	8.05 %	4.02 %	6.04 %	6.64 %	5.73 %	6.14 %
Llama 2 7b (TheBloke)	13.88 %	17.35 %	29.59 %	25.51 %	7.24 %	6.64 %	11.22 %
Llama 3.1 8b (Ollama)	32.33 %	22.59 %	16.87 %	18.37 %	11.35 %	10.04 %	21.18 %

TABLE III: Attack success rate (ASR) across snapshots of current OpenAI models evaluated on 16 different prompt-based attacks over 1,000 iterations per snapshot in total.

Model	ASR	Model	ASR
GPT-3.5 Turbo		**GPT-4**	
↪ gpt-3.5-turbo-1106	4.63 %	↪ gpt-4-0314	9.56 %
↪ gpt-3.5-turbo-0125	16.11 %	↪ gpt-4-0613	8.05 %
GPT-4 Turbo		**GPT-4o**	
↪ gpt-4-1106-preview	1.81 %	↪ gpt-4o-2024-05-13	1.01 %
↪ gpt-4-0125-preview	2.13 %	↪ gpt-4o-2024-08-06	0.10 %
↪ gpt-4-turbo-2024-04-09	2.31 %	↪ gpt-4o-2024-11-20	0.20 %

model configurations are provided in the appendix of the extended version of this paper.

Results. The results are depicted in Figure 4. We observe a near-linear increase in F1-score with the degree of test set leakage. Even a modest leakage of 20% leads to a gain of 0.08–0.11 over the baseline. This demonstrates that even partial leakage can significantly inflate evaluation metrics, leading to overly optimistic conclusions.

2) Evaluation on Commercial LLMs: Having observed the effects of leakage under controlled conditions, we now turn to the question of whether similar effects can be found in proprietary LLMs. The main challenge in this setting is the lack of transparency regarding the training data and procedures used for these models. To investigate this, we focus on the *PrimeVul* [123] dataset. It consists of real-world vulnerabilities in public C/C++ projects, making it a plausible candidate for inclusion in the pre-training data of proprietary models. We evaluate four proprietary models from different vendors (*gpt-3.5-turbo-0125*, *gpt-4o-2024-08-06*, *DeepSeek-V3-0324*, *claude-3-5-haiku-20241022*) as well as three open-source models (*meta-llama-3-8b-instruct*, *qwen3-14b*, *qwen2.5-coder-14b*). For all evaluations, we set the temperature to 0. Detailed model information is available in the appendix of the extended version of this paper.

For each model, we conduct the following two experiments: (1) We sample 100 commits from the dataset, extract the original commit messages, and prompt the model with the project name, commit SHA, and the first half of the message, asking it to reconstruct the full commit message. (2) We repeat this process, using 100 sampled functions. We provide the same context, but instead of the commit message, we prompt the model with the first half of the function body and ask it to reconstruct the complete function.

Results. Surprisingly, none of the models successfully reconstructed a single commit message. Even allowing for fuzzy matching ($\geq 75\%$ word-level Jaccard similarity), all models yielded 0 out of 100 matches. The function completion task showed the same pattern: no exact or fuzzy matches across any of the models. To rule out the possibility that the alignment procedures may suppress memorization, we fine-tune *gpt-3.5-turbo-0125* on 1,000 full commit messages (following the divergence attack from Carlini et al. [125]). Despite this targeted fine-tuning, the model still failed to reproduce any of the original messages.

To investigate whether these commits may have been absent from the model's pre-training data, we searched all *Common Crawl* snapshots listed in the GPT-3 paper [120] for any URLs related to the repositories used in *PrimeVul* (with wildcards). We found no matches. While we cannot rule out the use of private or proprietary data sources, the absence of *PrimeVul*-related content in *Common Crawl* suggests that *PrimeVul*-style data was likely not part of the pre-training mix.

These findings are somewhat unexpected, given that *PrimeVul* contains data from many of the most prominent and widely used open-source C/C++ projects on GitHub. Nonetheless, we found no evidence that any of the tested models were trained on *PrimeVul* commits or functions.

Discussion. This finding is both surprising and encouraging. Despite the prominence of the underlying projects in *PrimeVul*, we find no evidence that the evaluated proprietary and open-source models were trained on these. While we cannot definitively rule out the possibility of data leakage, the results provide strong evidence that data leakage is unlikely in this case. To support future evaluations and reduce the risk of leakage, we recommend the following: Ideally, test data should be drawn entirely from after the model's training cut-off date. In this case, it can be used with high confidence. If the test data spans both before and after the cut-off, a pre/post probing analysis can help: a significant drop in performance on the post cut-off portion may indicate the absence of leakage. If the test data predates the cut-off or if timing is unclear, targeted memorization probes can be performed as done in our reconstruction experiments. To aid this evaluation, the extended version of this paper summarizes publicly disclosed training
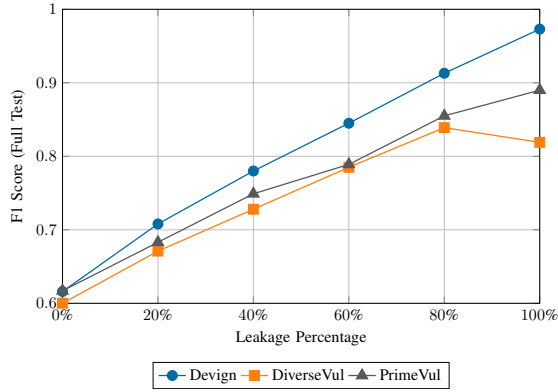
Fig. 4: F1-scores of CodeT5+ models fine-tuned on *Devign*, *DiverseVul*, and *PrimeVul* with varying amounts of test data (0–100%) leaked into the training data.
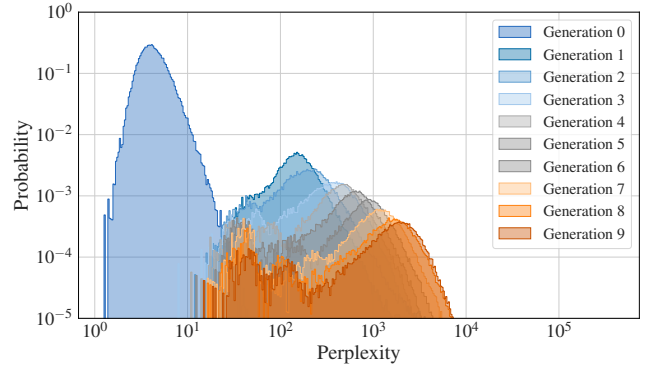

Fig. 5: Distribution of perplexities for training samples generated across multiple model generations. Perplexity is measured using the original model fine-tuned on real data. Both the mean and variance of perplexity increase over generations, suggesting that models become less stable when repeatedly trained on synthetic data.

data sources for several representative language models in its appendix.

C. Context Truncation

Next, we shift our focus to prompting, specifically the issue of limited context windows. This pitfall appears in 28% of the reviewed papers. To this end, we examine how constrained context length can affect the reliability of performance measurements and potentially lead to misleading conclusions. More specifically, our goal is to understand whether limited context during training or evaluation can prevent a model from detecting vulnerabilities, thereby producing misleading performance results. For instance, *CVE-2014-2669*[5] illustrates a case where the line responsible for an integer overflow lies outside a 512-token context window; hence, a model would be unable to detect it since the full context is not available.

Experimental Setup. Among the 15 papers on vulnerability detection included in our study, we observe that eight papers use at least one model with a context size of 512 tokens or less, one paper uses a model with a 1024-token context size, and eight papers use at least one model with a context size larger than 2048 tokens.

To assess whether these context sizes are sufficient, we analyze function lengths in the datasets used in §IV-B. Using the CodeT5 [126] tokenizer, we compute the number of tokens for each function labeled as vulnerable. We then calculate the proportion of functions in each dataset that exceed context limits of 512, 1024, and 2048 tokens. This allows to quantify the proportion of functions that are partially truncated under typical context size assumptions. Further details on the tokenizer are provided in the appendix of the extended version of this paper.

Results. We find that a substantial proportion of vulnerable functions exceed the context sizes used in papers on vulnerability detection in our study. As shown in Table IV, on average,

[5]https://nvd.nist.gov/vuln/detail/CVE-2014-2669; see the appendix of the extended version of this paper.

49.3% of all vulnerable functions across *Devign*, *DiverseVul*, and *PrimeVul* contain more than 512 tokens when tokenized with the CodeT5 tokenizer. At the 1024-token threshold, 29.1% still exceed the limit, and 13.7% surpass 2048 tokens.

These findings show that using limited context windows systematically truncates a large portion of vulnerable functions, potentially removing the very code that contains or explains the vulnerability. As such, evaluation under small context settings may fundamentally misrepresent a model's performance and lead to misleading conclusions about its ability to detect vulnerabilities. Notably, these percentages represent a lower bound on the issue; Risse et al. [112] show that vulnerability detection often depends on code outside the function itself, meaning even full-function context may be insufficient in many cases.

TABLE IV: Proportion of functions labeled as *vulnerable* whose CodeT5-tokenized length exceeds context-window sizes used in the papers on vulnerability detection identified by our pitfall study (512, 1,024, and 2,048 tokens).

Dataset	# Funcs	# Tokens		
		> 512	> 1024	> 2048
Devign	12,460	5,196 (41.7%)	2,642 (21.2%)	984 (7.9%)
DiverseVul	18,945	9,814 (51.8%)	5,835 (30.8%)	2,747 (14.5%)
PrimeVul	6,004	3,897 (64.9%)	2,594 (43.2%)	1,357 (22.6%)
Average	–	**52.8%**	**31.7%**	**15.0%**

D. Model Collapse

Finally, we consider another important aspect of LLM training: the potential effects of model collapse, which we found to be present in 13.9% of the papers. Repeated training on synthetic data, particularly content generated by models, can gradually degrade model quality, amplify existing biases, and reduce output diversity [25].

Experimental Setup. We build on the setup introduced by Shumailov et al. [25], who studied model collapse in natural language generation. Code generation is more challenging than generating natural language text, as source code is subject to stricter syntactic and semantic constraints. For this experiment, we use the Qwen2.5-Coder-0.5B-Instruct [127], [128] model (full configuration details are provided in the appendix of the extended version of this paper) and consider training sequences with a fixed length of 128 tokens in line with the setup from Shumailov et al. [25].

We begin by fine-tuning the base model on the *self-oss-instruct-sc2-exec-filter-50k* [129], [130] dataset, which contains 50,000 Python code samples. For each sequence, we predict up to 2,048 subsequent tokens to generate synthetic samples, ensuring that no code sample is truncated or incomplete. These synthetic outputs form a new dataset of equal sample size, which is then used to fine-tune the next model iteration. We repeat this process for ten generations, where each model is trained on data generated by its immediate predecessor. Although it cannot be ruled out that the base model is trained on (subsets) of this dataset, the experiment's primary focus is on the re-training on LLM-generated data. Hence, a possible data leakage would not affect the results.

Results. Figure 5 shows the distribution of perplexities across generations, measured by evaluating each generation's outputs using the fine-tuned base model. *Generation 0* refers to the fine-tuned base model evaluated on the unmodified training data. *Generation 1* corresponds to the synthetic outputs from the first-generation model, and so on.

We observe a clear trend of degradation in model performance. With each generation, the distribution shifts to the right, indicating increased perplexity and suggesting that the base model becomes increasingly uncertain when predicting tokens in samples produced by later generations. We also observe growing variance in perplexity across generations, reflecting greater instability in models trained on synthetic data, and indicating that errors accumulate over time. Furthermore, models trained primarily on synthetic outputs fail to generalize effectively.

These results extend prior concerns raised by Shumailov et al. [25] to the code domain. Our findings suggest that similar degradation can occur in code-related applications as well. This trend is especially concerning as AI-powered coding assistants become more widespread and, thus, an increasing share of source code may be generated by models rather than written by humans. From a security perspective, this trend also poses a serious risk. For example, when the underlying models are used for code generation, their reliability may degrade over time, potentially introducing subtle bugs or vulnerabilities. Similarly, when such models are used for, e.g., vulnerability detection, the degradation of the model may result in a lower detection rate.

V. RECOMMENDATIONS

Our analysis reveals a concerning pattern: every reviewed paper suffers from at least one pitfall, and each pitfall appears in multiple papers. In the following, we propose recommendations that directly target the pitfalls while considering real-world constraints such as the use of proprietary systems, limited access to training data, and computational limitations.

We provide an extended discussion covering each pitfall individually in the appendix of the extended paper version available at: https://doi.org/10.48550/arXiv.2512.09549. In addition to this discussion, we offer a project website with guidelines to avoid pitfalls and a living appendix where all information is open for further contributions and kept up to date. The website is available at https://llmpitfalls.org.

R1: Transparency and Reproducibility. Transparent reporting and reproducibility are crucial in research. Even if full reproducibility is not always feasible, especially with proprietary APIs, researchers should share enough information to enable others to interpret, replicate, or critically evaluate the findings.

We recommend the following reporting practices:

- Specify the exact model used, including version identifier, access method (e.g., API or interface), and access date. For open-source models, include repository links, commit hashes, and any fine-tuning steps that were taken. Describe the evaluation pipeline in detail, including prompt format, quantization level, decoding settings, and any post-processing. As shown in §IV-A, missing or vague model details can lead to substantial variation in outcomes and hinder reproducibility ($\longrightarrow$ *Model Ambiguity (P9)*).

- If evaluation labels are generated by language models or if LLMs are used as judges, disclose this and assess label quality via manual review on a statistically meaningful subset. Report inter-annotator agreement where applicable ($\longrightarrow$ *LLM-generated Label Inaccuracy (P2)*).

- Avoid generalizing beyond what the evidence supports. Claims should be scoped to the specific models evaluated unless a diverse and representative set of models is tested and limitations are explicitly acknowledged ($\longrightarrow$ *Proxy-/Surrogate Fallacy (P8)*).

R2: Understand and Communicate Data Quality. Data plays a central role in both training and evaluation of language models. Whether data is synthetic, scraped, or manually labeled, its origin, quality, and security can significantly influence model behavior and reported results.

To mitigate these risks, we recommend:

- When using synthetic or LLM-generated data for training, clearly report its proportion relative to real data and assess distributional differences. In iterative self-training setups, monitor for signs of model collapse or performance degradation. As demonstrated in §IV-D, iterative training on synthetic data can lead to significant performance decline and instability over time ($\longrightarrow$ *Model Collapse via Synthetic Training Data (P4)*).

- For tasks with realistic poisoning threats, such as vulnerability detection or misinformation detection, explicitly assess whether poisoning is plausible in the training setup, particularly when relying on large-scale web data or proprietary models (⟶ *Data Poisoning via Internet Scraping (P1)*).

- Acknowledge any risk of data leakage by checking whether evaluation data (especially answers or labels) was publicly available prior to the model's training cutoff date. Where cutoff dates are known, analyze performance on pre- vs. post-cutoff data and consider probing for memorization. We demonstrate the impacts of potential data leakage in §IV-B. Whenever possible, prefer models for which information about training data composition is available (see the appendix of the extended version of this paper). As a community, we should push for state-of-the-art models with transparent or public training data. While this is often not possible for proprietary models, ongoing regulatory efforts such as the EU AI Act's transparency requirements for General-Purpose AI (GPAI) providers [131] are expected to improve disclosure practices (⟶ *Data Leakage (P3)*).

R3: Alignment between Models and Tasks. Many evaluation failures stem not from the models themselves, but from a mismatch between the model constraints and the task requirements. Researchers should assess whether the models used are capable of handling the inputs, outputs, and task complexity involved and report limitations clearly.

To that end:

- Report the model's maximum context window and analyze whether typical input sizes (including prompts) exceed this limit. If truncation occurs frequently, quantify how often and assess the potential impact on performance. Detailed evaluation of possible side effects of a too small context size is given in §IV-C (⟶ *Context Truncation (P6)*).

- Recognize that prompt design can meaningfully influence model behavior. When feasible, use established prompting techniques or conduct prompt variation experiments to gauge robustness. If prompt optimization is not applicable, explain why (⟶ *Prompt Sensitivity (P7)*).

- Evaluate whether the model may be exploiting spurious correlations. Use robustness checks (e.g., input perturbations, ablations) or feature attribution techniques to test whether the model relies on unintended features (⟶ *Spurious Correlations/Unrelated Features (P5)*).

VI. DISCUSSION

Our study provides a systematic analysis of common pitfalls in LLM security research. However, like any such effort, it is not without limitations. In the following, we reflect on potential threats to validity and the generalizability of our findings, as well as the broader implications of the issues we uncovered.

Addressing Inevitable Pitfalls. Some pitfalls in LLM research are difficult to avoid due to the inherent complexity and scale of current systems. The vast amount of data required to train LLMs introduces inherently challenging-to-manage risks. For example, Data Poisoning attacks (P1), although realistic in principle, are challenging to detect and mitigate, especially when experiments rely on publicly available data. Similarly, the lack of transparency from LLM providers regarding training corpora increases the likelihood of Data Leakage (P3). In addition, the complexity and often opaque behavior of LLMs can lead to Spurious Correlations (P5) that are difficult to identify or explain. Although these issues cannot always be fully resolved (see §V for possible mitigation strategies), researchers must acknowledge, address, and ideally explicitly discuss such pitfalls when reporting their findings.

Coverage of Literature and Pitfalls. Our paper selection may not be exhaustive. Relevant work may have been missed due to limitations in our keyword filters or because it appeared outside the selected venues. To reduce this risk, we conducted an intentionally broad search across eight leading security and software engineering conferences, used an extensive keyword list, and applied a two-stage manual selection process. This resulted in 72 in-scope papers. While omissions remain possible, we believe they are unlikely to affect our qualitative conclusions.

Likewise, other pitfalls may exist beyond the nine we identified for our study. We iteratively refined our pitfall list and definitions during test reviews with a diverse reviewing team to limit this threat. We encourage future work to build on and extend this taxonomy.

Review Subjectivity. While some degree of subjectivity is unavoidable in qualitative reviews, we took several steps to improve consistency. Each paper was reviewed by two independent reviewers, who were blinded during the first phase of evaluation. We then conducted a structured discussion phase to reconcile disagreements.

To quantify inter-reviewer consistency, we computed Fleiss' κ for each pitfall. Overall agreement was solid, with a mean κ of 0.55 across all pitfalls. Out of 648 individual ratings (72 papers $\times$ 9 pitfalls), only 25 required resolution through discussion in the larger reviewer group. This corresponds to less than 5% of all decisions. Most disagreements were minor and resolved through brief conversations between reviewers. Full κ values and qualitative interpretations are reported in the appendix of the extended paper version.

Generalizability of the Impact Analysis. Our experimental study focuses on four case studies and five selected pitfalls, covering key stages of the LLM pipeline. Thus, four pitfalls remain unexplored in the experimental analysis, and even for the five chosen ones, their impact may vary across tasks, datasets, and model families. We emphasize that the impact analysis aims to demonstrate plausible mechanisms and concrete examples rather than provide generalizable effect sizes. As such, the results should be interpreted as illustrative lower bounds, not as comprehensive or definitive impact measures.

VII. Conclusion

In this work, we systematically reviewed 72 papers from leading security and software engineering conferences and identified nine recurring pitfalls in LLM security research. Every paper we considered contained at least one of these pitfalls, and each pitfall appeared across multiple studies. Our impact analysis shows how seemingly minor issues, such as Data Leakage, Context Truncation, or Model Ambiguity, can distort results, reduce reliability, and undermine reproducibility. Together, these findings highlight the broader risks these pitfalls pose to the integrity of LLM security research and underscore the need for more rigorous, transparent, and reproducible practices.

Looking ahead, strengthening LLM security research involves applying the lessons we outline: reporting exact model identifiers, testing for leakage, using task-appropriate context sizes, verifying LLM-generated labels, and avoiding over-generalization. Our guidelines are not meant to serve as a fixed checklist. However, by treating these pitfalls as design constraints rather than afterthoughts, the community can move from "chasing shadows" to conducting LLM security research that is reproducible, trustworthy, and grounded in evidence.

Ethical Considerations

This work critically examines methodological patterns in LLM security research. To ensure transparency while minimizing the risk of reputational harm, we cite all sources transparently but report pitfalls in aggregate. We deliberately refrain from assigning specific pitfalls to individual papers. Corresponding authors of the analyzed work are welcome to contact us for private discussion and clarification.

Our aim is constructive. Alongside our critique, we provide concrete, community-oriented guidelines to support more rigorous, transparent, and reproducible practices in LLM-focused security research.

References

[1] A. Stafeev, T. Recktenwald, G. D. Stefano, S. Khodayari, and G. Pellegrino, "Yurascanner: Leveraging llms for task-driven web app scanning." in *NDSS*. The Internet Society, 2025.

[2] Y. Sun, D. Wu, Y. Xue, H. Liu, H. Wang, Z. Xu, X. Xie, and Y. Liu, "Gptscan: Detecting logic vulnerabilities in smart contracts by combining gpt with program analysis," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. Association for Computing Machinery, 2024.

[3] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, "PentestGPT: Evaluating and harnessing large language models for automated penetration testing," in *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, 2024.

[4] C. S. Xia, M. Paltenghi, J. Le Tian, M. Pradel, and L. Zhang, "Fuzz4All: Universal Fuzzing with Large Language Models," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. Association for Computing Machinery, 2024.

[5] C. Fang, N. Miao, S. Srivastav, J. Liu, R. Zhang, R. Fang, Asmita, R. Tsang, N. Nazari, H. Wang, and H. Homayoun, "Large language models for code analysis: Do LLMs really do their job?" in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[6] X. Zhou, S. Cao, X. Sun, and D. Lo, "Large language model for vulnerability detection and repair: Literature review and the road ahead," *2024 IEEE/ACM 46th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER*, 2025.

[7] L. Derczynski, E. Galinkin, J. Martin, S. Majumdar, and N. Inie. (2024) garak: A Framework for Security Probing Large Language Models.

[8] M. D. Purba, A. Ghosh, B. J. Radford, and B. Chu, "Software vulnerability detection using large language models," in *2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2023.

[9] H. Pearce, B. Tan, B. Ahmad, R. Karri, and B. Dolan-Gavitt, " Examining Zero-Shot Vulnerability Repair with Large Language Models ," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2023.

[10] X. Zhou, K. Kim, B. Xu, D. Han, and D. Lo, "Out of sight, out of mind: Better automatic vulnerability repair by broadening input ranges and sources," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. Association for Computing Machinery, 2024.

[11] S. B. Hossain, N. Jiang, Q. Zhou, X. Li, W.-H. Chiang, Y. Lyu, H. Nguyen, and O. Tripp, "A deep dive into large language models for automated bug localization and repair," *Proc. ACM Softw. Eng.*, 2024.

[12] J. Zhang, C. Wang, A. Li, W. Wang, T. Li, and Y. Liu, "Vuladvisor: Natural language suggestion generation for software vulnerability repair," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '24. Association for Computing Machinery, 2024.

[13] E. Zverev, S. Abdelnabi, S. Tabesh, M. Fritz, and C. H. Lampert, "Can LLMs separate instructions from data? and what do we even mean by that?" in *The Thirteenth International Conference on Learning Representations*, 2025.

[14] J. Evertz, M. Chlosta, L. Schönherr, and T. Eisenhofer, "Whispers in the machine: Confidentiality in agentic systems," *CoRR*, 2024.

[15] R. Hankache, K. N. Acheampong, L. Song, M. Brynda, R. Khraishi, and G. A. Cowan, "Evaluating the sensitivity of llms to prior context," 2025.

[16] Y. Liu, Y. Jia, R. Geng, J. Jia, and N. Z. Gong, "Formalizing and benchmarking prompt injection attacks and defenses," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[17] Z. Yu, X. Liu, S. Liang, Z. Cameron, C. Xiao, and N. Zhang, "Don't listen to me: Understanding and exploring jailbreak prompts of large language models," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[18] D. Arp, E. Quiring, F. Pendlebury, and A. Warnecke, "Dos and Don'ts of Machine Learning in Computer Security," in *USENIX Security Symposium*, 2022.

[19] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin, "On evaluating adversarial robustness," *CoRR*, 2019.

[20] L. Zheng, W. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica, "Judging llm-as-a-judge with mt-bench and chatbot arena," in *Advances in Neural Information Processing Systems 36: Annual Conference on*

Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023, 2023.

[21] OpenAI, "Gpt-4 system card," 2023, system card.

[22] ——, "Gpt-4 technical report," 2023.

[23] Golem.de. (2025) Künstliche intelligenz: Reddit verklagt anthropic. https://www.golem.de/news/kuenstliche-intelligenz-reddit-verklagt-anthropic-2506-196895.html.

[24] O. Nickel. (2024) Answers: Reddit testet KI-suche. https://www.golem.de/news/answers-reddit-testet-ki-suche-2412-191572.html.

[25] I. Shumailov, Z. Shumaylov, Y. Zhao, and N. Papernot, "AI models collapse when trained on recursively generated data," *Nature*, 2024.

[26] N. Gillman, M. Freeman, D. Aggarwal, C.-H. Hsu, and C. Luo, "Self-correcting self-consuming loops for generative model training," in *Proceedings of the 41st International Conference on Machine Learning*, 2024.

[27] Y. Kossale, M. Airaj, and A. Darouichi, "Mode collapse in generative adversarial networks: An overview," in *2022 8th International Conference on Optimization and Applications (ICOA)*, 2022.

[28] S. Alemohammad, J. Casco-Rodriguez, L. Luzi, A. I. Humayun, H. Babaei, D. LeJeune, A. Siahkoohi, and R. Baraniuk, "Self-consuming generative models go MAD," in *The Twelfth International Conference on Learning Representations*, 2024.

[29] B. Cao, D. Cai, Z. Zhang, Y. Zou, and W. Lam, "On the worst prompt performance of large language models," in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[30] R. Pryzant, D. Iter, J. Li, Y. T. Lee, C. Zhu, and M. Zeng, "Automatic prompt optimization with "gradient descent" and beam search," in *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[31] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023.

[32] M. AI, "Introducing llama 3.1: Our most capable models to date," Meta AI Blog, 2024.

[33] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, and A. Kadian, "The llama 3 herd of models," *CoRR*, 2024.

[34] OpenAI. (2025) Openai chatgpt. https://chatgpt.com/.

[35] Huggingface. (2025) Hugging face – the AI community building the future. https://huggingface.co/.

[36] Anthropic. (2025) Anthropic claude. https://claude.ai/.

[37] Google. (2025) Google gemini. https://gemini.google.com.

[38] M. Geng, S. Wang, D. Dong, H. Wang, G. Li, Z. Jin, X. Mao, and X. Liao, "Large language models are few-shot summarizers: Multi-intent comment generation via in-context learning," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE '24)*, 2024.

[39] H. Wang, K. Dong, Z. Zhu, H. Qin, A. Liu, X. Fang, J. Wang, and X. Liu, "Transferable multimodal attack on vision-language pre-training models," in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024.

[40] Z. Zhang, G. Shen, G. Tao, S. Cheng, and X. Zhang, "On large language models' resilience to coercive interrogation," in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024.

[41] J. Liu, Y. Kang, D. Tang, K. Song, C. Sun, X. Wang, W. Lu, and X. Liu, "Order-disorder: Imitation adversarial attacks for black-box neural ranking models," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. Association for Computing Machinery, 2022.

[42] A. Naseh, K. Krishna, M. Iyyer, and A. Houmansadr, "Stealing the decoding algorithms of language models," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23. Association for Computing Machinery, 2023.

[43] M. Du, X. Yue, S. S. M. Chow, T. Wang, C. Huang, and H. Sun, "Dp-forward: Fine-tuning and inference on language models with differential privacy in forward pass," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23. Association for Computing Machinery, 2023.

[44] S. Yan, S. Wang, Y. Duan, H. Hong, K. Lee, D. Kim, and Y. Hong, "An LLM-Assisted Easy-to-Trigger backdoor attack on code completion models: Injecting disguised vulnerabilities against strong detection," in *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, 2024.

[45] Y. Liu, Y. Jia, R. Geng, J. Jia, and N. Z. Gong, "Formalizing and benchmarking prompt injection attacks and defenses," in *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, 2024.

[46] R. Zhang, H. Li, R. Wen, W. Jiang, Y. Zhang, M. Backes, Y. Shen, and Y. Zhang, "Instruction backdoor attacks against customized LLMs," in *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, 2024.

[47] M. Meeus, S. Jain, M. Rei, and Y.-A. de Montjoye, "Did the neurons read your book? document-level membership inference for large language models," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[48] J. Yu, X. Lin, Z. Yu, and X. Xing, "LLM-Fuzzer: Scaling assessment of large language model jailbreaks," in *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, 2024.

[49] Z. Yu, X. Liu, S. Liang, Z. Cameron, C. Xiao, and N. Zhang, "Don't listen to me: Understanding and exploring jailbreak prompts of large language models," in *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, 2024.

[50] T. Liu, Y. Zhang, Z. Zhao, Y. Dong, G. Meng, and K. Chen, "Making them ask and answer: Jailbreaking large language models in few queries via disguise and reconstruction," in *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, 2024.

[51] S. Liu, D. Cao, J. Kim, T. Abraham, P. Montague, S. Camtepe, J. Zhang, and Y. Xiang, "EaTVul: ChatGPT-based evasion attack against software vulnerability detection," in *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, 2024.

[52] W. M. Si, M. Backes, Y. Zhang, and A. Salem, "Two-in-One: A model hijacking attack against text generation models," in *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, 2023.

[53] L. Shen, Y. Pu, S. Ji, C. Li, X. Zhang, C. Ge, and T. Wang, "Improving the robustness of transformer-based large language models with dynamic attention," in *NDSS*, 2024.

[54] C. Wei, W. Meng, Z. Zhang, M. Chen, M. Zhao, W. Fang, L. Wang, Z. Zhang, and W. Chen, "Lmsanitator: Defending prompt-tuning against task-agnostic backdoors," in *NDSS*, 2024.

[55] G. Deng, Y. Liu, Y. Li, K. Wang, Y. Zhang, Z. Li, H. Wang, T. Zhang, and Y. Liu, "Masterkey: Automated jailbreaking of large language model chatbots," in *NDSS*, 2024.

[56] M. Nazzal, I. Khalil, A. Khreishah, and N. Phan, "Promsec: Prompt optimization for secure generation of functional source code with large language models (llms)," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. Association for Computing Machinery, 2024.

[57] G. Chen, Z. Qin, M. Yang, Y. Zhou, T. Fan, T. Du, and Z. Xu, "Unveiling the vulnerability of private fine-tuning in split-based frameworks for large language models: A bidirectionally enhanced attack," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. Association for Computing Machinery, 2024.

[58] X. Shen, Z. Chen, M. Backes, Y. Shen, and Y. Zhang, ""do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. Association for Computing Machinery, 2024.

[59] B. Hui, H. Yuan, N. Gong, P. Burlina, and Y. Cao, "Pleak: Prompt leaking attacks against large language model applications," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. Association for Computing Machinery, 2024.

[60] H. Aghakhani, W. Dai, A. Manoel, X. Fernandes, A. Kharkar, C. Kruegel, G. Vigna, D. Evans, B. Zorn, and R. Sim, "Trojanpuzzle: Covertly poisoning code-suggestion models," in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024.

[61] D. Li, M. Yan, Y. Zhang, Z. Liu, C. Liu, X. Zhang, T. Chen, and D. Lo, "Cosec: On-the-fly security hardening of code llms via supervised co-decoding," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2024. Association for Computing Machinery, 2024.

[62] J. Chen, Q. Zhong, Y. Wang, K. Ning, Y. Liu, Z. Xu, Z. Zhao, T. Chen, and Z. Zheng, "Rmcbench: Benchmarking large language models' resistance to malicious code," in *Proceedings of the 39th IEEE/ACM*

International Conference on Automated Software Engineering, ser. ASE '24. Association for Computing Machinery, 2024.

[63] S. Ullah, M. Han, S. Pujar, H. Pearce, A. Coskun, and G. Stringhini, " LLMs Cannot Reliably Identify and Reason About Security Vulnerabilities (Yet?): A Comprehensive Evaluation, Framework, and Benchmarks ," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024.

[64] P. Liu, J. Liu, L. Fu, K. Lu, Y. Xia, X. Zhang, W. Chen, H. Weng, S. Ji, and W. Wang, "Exploring ChatGPT's capabilities on vulnerability management," in *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, 2024.

[65] Z. Liu, Z. Tang, J. Zhang, X. Xia, and X. Yang, "Pre-training by predicting program dependencies for vulnerability analysis tasks," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. Association for Computing Machinery, 2024.

[66] M. M. Rahman, I. Ceka, C. Mao, S. Chakraborty, B. Ray, and W. Le, "Towards causal deep learning for vulnerability detection," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. Association for Computing Machinery, 2024.

[67] A. Sejfia, S. Das, S. Shafiq, and N. Medvidović, "Toward improved deep learning-based vulnerability detection," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. Association for Computing Machinery, 2024.

[68] X.-C. Wen, C. Gao, S. Gao, Y. Xiao, and M. R. Lyu, "Scale: Constructing structured natural language comment trees for software vulnerability detection," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2024. Association for Computing Machinery, 2024.

[69] Y. Ding, S. Chakraborty, L. Buratti, S. Pujar, A. Morari, G. Kaiser, and B. Ray, "Concord: Clone-aware contrastive learning for source code," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2023. Association for Computing Machinery, 2023.

[70] Y. Zhao, L. Gong, Z. Huang, Y. Wang, M. Wei, and F. Wu, "Coding-ptms: How to find optimal code pre-trained models for code embedding in vulnerability detection?" in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '24. Association for Computing Machinery, 2024.

[71] C. Ni, X. Yin, K. Yang, D. Zhao, Z. Xing, and X. Xia, "Distinguishing look-alike innocent and vulnerable code by subtle semantic representation learning and explanation," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2023. Association for Computing Machinery, 2023, p. 1611–1622.

[72] M. Eshghie and C. Artho, "Oracle-guided vulnerability diversity and exploit synthesis of smart contracts using llms," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '24. Association for Computing Machinery, 2024.

[73] Y. Wu, X. Xie, C. Peng, D. Liu, H. Wu, M. Fan, T. Liu, and H. Wang, "Advscanner: Generating adversarial smart contracts to exploit reentrancy vulnerabilities using llm and static analysis," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '24. Association for Computing Machinery, 2024.

[74] J. Sun, J. Chen, Z. Xing, Q. Lu, X. Xu, and L. Zhu, "Where is it? tracing the vulnerability-relevant files from vulnerability reports," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. Association for Computing Machinery, 2024.

[75] N. Lukas, A. Salem, R. Sim, S. Tople, L. Wutschitz, and S. Zanella-Beguelin, " Analyzing Leakage of Personally Identifiable Information in Language Models ," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2023.

[76] X. He, S. Zannettou, Y. Shen, and Y. Zhang, " You Only Prompt Once: On the Capabilities of Prompt Learning on Large Language Models to Tackle Toxic Content ," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024.

[77] N. Vishwamitra, K. Guo, F. T. Romit, I. Ondracek, L. Cheng, Z. Zhao, and H. Hu, " Moderating New Waves of Online Hate with Chain-of-Thought Reasoning in Large Language Models ," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024.

[78] S. S. Roy, P. Thota, K. V. Naragam, and S. Nilizadeh, "From chatbots to phishbots?: Phishing scam generation in commercial large language models," in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024.

[79] W. M. Si, M. Backes, J. Blackburn, E. De Cristofaro, G. Stringhini, S. Zannettou, and Y. Zhang, "Why so toxic? measuring and triggering toxic behavior in open-domain chatbots," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. Association for Computing Machinery, 2022.

[80] K. Tang, W. Zhou, J. Zhang, A. Liu, G. Deng, S. Li, P. Qi, W. Zhang, T. Zhang, and N. Yu, "Gendercare: A comprehensive framework for assessing and reducing gender bias in large language models," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. Association for Computing Machinery, 2024.

[81] Y. Li, C. Huang, S. Deng, M. L. Lock, T. Cao, N. Oo, H. W. Lim, and B. Hooi, "KnowPhish: Large language models meet multimodal knowledge graphs for enhancing Reference-Based phishing detection," in *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, 2024.

[82] R. Zhang, S. S. Hussain, P. Neekhara, and F. Koushanfar, "REMARK-LLM: A robust and efficient watermarking framework for generative large language models," in *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, Aug. 2024.

[83] Z. Lin, J. Cui, X. Liao, and X. Wang, "Malla: Demystifying real-world large language model integrated malicious services," in *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, 2024.

[84] K. Guo, A. Utkarsh, W. Ding, I. Ondracek, Z. Zhao, G. Freeman, N. Vishwamitra, and H. Hu, "Moderating illicit online image promotion for unsafe user generated content games using large Vision-Language models," in *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, 2024.

[85] L. Niu, S. Mirza, Z. Maradni, and C. Pöpper, "CodexLeaks: Privacy leaks from code generation language models in GitHub copilot," in *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, 2023.

[86] P. Lv, P. Li, S. Zhu, S. Zhang, K. Chen, R. Liang, C. Yue, F. Xiang, Y. Cai, H. Ma, Y. Zhang, and G. Meng, "Ssl-wm: A black-box watermarking approach for encoders pre-trained by self-supervised learning," in *NDSS*, 2022.

[87] C. Wu, J. Chen, Z. Wang, R. Liang, and R. Du, "Semantic sleuth: Identifying ponzi contracts via large language models," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '24. Association for Computing Machinery, 2024.

[88] M. Bethany, B. Wherry, E. Bethany, N. Vishwamitra, A. Rios, and P. Najafirad, "Deciphering textual authenticity: A generalized strategy through the lens of large language semantics for detecting human vs. Machine-Generated text," in *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, 2024.

[89] Asmita, Y. Oliinyk, M. Scott, R. Tsang, C. Fang, and H. Homayoun, "Fuzzing {BusyBox}: Leveraging {LLM} and Crash Reuse for Embedded Bug Unearthing," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[90] Y. Deng, C. S. Xia, H. Peng, C. Yang, and L. Zhang, "Large Language Models Are Zero-Shot Fuzzers: Fuzzing Deep-Learning Libraries via Large Language Models," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. Association for Computing Machinery, 2023.

[91] (2024) Large Language Model guided Protocol Fuzzing. NDSS Symposium.

[92] Y. Lyu, Y. Xie, P. Chen, and H. Chen, "Prompt Fuzzing for Fuzz Driver Generation," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, 2024.

[93] X. Ma, L. Luo, and Q. Zeng, "From One Thousand Pages of Specification to Unveiling Hidden Bugs: Large Language Model Assisted Fuzzing of Matter {IoT} Devices," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[94] J. Wang, L. Yu, and X. Luo, "LLMIF: Augmented Large Language Model for Fuzzing IoT Devices," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024.

[95] D. Wang, G. Zhou, L. Chen, D. Li, and Y. Miao, "ProphetFuzz: Fully Automated Prediction and Fuzzing of High-Risk Option Combinations with Only Documentation via Large Language Model," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. Association for Computing Machinery, 2024.

[96] Z. Li, C. Wang, S. Wang, and C. Gao, "Protecting intellectual property of large language model-based code generation apis via watermarks," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23. Association for Computing Machinery, 2023.

[97] C. Fang, N. Miao, S. Srivastav, J. Liu, R. Zhang, R. Fang, Asmita, R. Tsang, N. Nazari, H. Wang, and H. Homayoun, "Large language models for code analysis: Do LLMs really do their job?" in *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, 2024.

[98] G. Sandoval, H. Pearce, T. Nys, R. Karri, S. Garg, and B. Dolan-Gavitt, "Lost at c: A user study on the security implications of large language model code assistants," in *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, 2023.

[99] P. Hu, R. Liang, and K. Chen, "Degpt: Optimizing decompiler output with llm," *Proceedings 2024 Network and Distributed System Security Symposium*, 2024.

[100] C. Wang, J. Zhang, J. Gao, L. Xia, Z. Guan, and Z. Chen, "Contract-tinker: Llm-empowered vulnerability repair for real-world smart contracts," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '24. Association for Computing Machinery, 2024.

[101] Y. Wu, N. Jiang, H. V. Pham, T. Lutellier, J. Davis, L. Tan, P. Babkin, and S. Shah, "How effective are neural networks for fixing security vulnerabilities," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2023. Association for Computing Machinery, 2023.

[102] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, and C. Studer, "Poison frogs! targeted clean-label poisoning attacks on neural networks," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018.

[103] Mitre. (2025) PoisonGPT | MITRE ATLAS™. https://atlas.mitre.org/.

[104] D. Bowen, B. Murphy, W. Cai, D. Khachaturov, and A. Gleave, "Data poisoning in LLMs: Jailbreak-tuning and scaling laws," 2025.

[105] A. Wan, E. Wallace, S. Shen, and D. Klein, "Poisoning language models during instruction tuning," in *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[106] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning From Data*. AMLBook, 2012.

[107] Y. Dong, X. Jiang, H. Liu, Z. Jin, and B. Gu, "Generalization or memorization: Data contamination and trustworthy evaluation for large language models," in *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

[108] M. E. A. Seddik, S.-W. Chen, S. Hayou, P. Youssef, and M. A. DEBBAH, "How bad is training on synthetic data? a statistical analysis of language model collapse," in *First Conference on Language Modeling*, 2024.

[109] Y. Guo, G. Shang, M. Vazirgiannis, and C. Clavel, "The curious decline of linguistic diversity: Training language models on synthetic text," in *Findings of the Association for Computational Linguistics: NAACL 2024*, 2024.

[110] A. Cloud, M. Le, J. Chua, J. Betley, A. Sztyber-Betley, J. Hilton, S. Marks, and O. Evans, "Subliminal learning: Language models transmit behavioral traits via hidden signals in data," *CoRR*, 2025.

[111] N. Risse and M. Böhme, "Uncovering the limits of machine learning for automatic vulnerability detection," in *USENIX Security*, 2024.

[112] N. Risse, J. Liu, and M. Böhme, "Top score on the wrong exam: On benchmarking in machine learning for vulnerability detection," *PACMSE*, 2025.

[113] M. Sclar, Y. Choi, Y. Tsvetkov, and A. Suhr, "Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting," in *The Twelfth International Conference on Learning Representations*, 2024.

[114] M. Abdin, J. Aneja, H. Awadalla, A. Awadallah, and A. A. Awan, "Phi-3 technical report: A highly capable language model locally on your phone."

[115] Microsoft. (2025) Microsoft phi-3-mini-4k-instruct. https://huggingface.co/microsoft/Phi-3-mini-4k-instruct.

[116] N. Vishwamitra, K. Guo, F. T. Romit, I. Ondracek, L. Cheng, Z. Zhao, and H. Hu, "Moderating new waves of online hate with chain-of-thought reasoning in large language models," in *IEEE Symposium on Security and Privacy (S&P)*, 2024.

[117] Ollama. (2025) Ollama. https://ollama.com.

[118] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. Bhatt, C. C. Ferrer, A. Grattafiori, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, and G. Synnaeve, "Code llama: Open foundation models for code," 2024.

[119] T. Jobbins. (2024) TheBloke - llm quantization. https://huggingface.co/TheBloke.

[120] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020.

[121] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, *Devign: effective vulnerability identification by learning comprehensive program semantics via graph neural networks*. Curran Associates Inc., 2019.

[122] Y. Chen, Z. Ding, L. Alowain, X. Chen, and D. Wagner, "Diversevul: A new vulnerable source code dataset for deep learning based vulnerability detection," in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, 2023.

[123] Y. Ding, Y. Fu, O. Ibrahim, C. Sitawarin, X. Chen, B. Alomair, D. Wagner, B. Ray, and Y. Chen, " Vulnerability Detection with Code Language Models: How Far are We? ," in *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, 2025.

[124] Y. Wang, H. Le, A. D. Gotmare, N. D. Q. Bui, J. Li, and S. Hoi, "Codet5+: Open code large language models for code understanding and generation," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

[125] M. Nasr, J. Rando, N. Carlini, J. Hayase, M. Jagielski, A. F. Cooper, D. Ippolito, C. A. Choquette-Choo, F. Tramèr, and K. Lee, "Scalable extraction of training data from aligned, production language models," in *The Thirteenth International Conference on Learning Representations*, 2025.

[126] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

[127] A. Yang, B. Yang, B. Hui, and B. Zheng, "Qwen2 technical report," *CoRR*, 2024.

[128] B. Hui, J. Yang, Z. Cui, and J. Yang, "Qwen2. 5-coder technical report," *CoRR*, 2024.

[129] B. Team. (2024) bigcode/self-oss-instruct-sc2-exec-filter-50k · datasets at hugging face. https://huggingface.co/datasets/bigcode/self-oss-instruct-sc2-exec-filter-50k.

[130] F. Cassano, J. Gouwar, F. Lucchetti, C. Schlesinger, A. Freeman, C. J. Anderson, M. Q. Feldman, M. Greenberg, A. Jangda, and A. Guha, "Knowledge transfer from high-resource to low-resource programming languages for code llms," *Proc. ACM Program. Lang.*, 2024.

[131] E. Commission. (2025) Commission presents template for general-purpose ai model providers to summarise data used to train their models.

Appendix A
Artifact

A. Description and Requirements

The artifact provides source code to reproduce the four case studies presented in the paper:

1) **Model Ambiguity** (Experiments A.1 and A.2, presented in §IV-A)
2) **Data Leakage** (Experiments B.1 and B.2, presented in §IV-B)
3) **Context Truncation** (Experiment C, presented in §IV-C)
4) **Model Collapse** (Experiment D, presented in §IV-D)

1) How to access: The artifact is available as a permanently archived version at: https://doi.org/10.5281/zenodo.17847798

2) Hardware requirements: Experiments A.1, A.2, B.2, and C can be run on a standard desktop machine (x86-64 CPU, 8 cores, 16 GB RAM). Experiments B.1 and D require GPUs to fully reproduce the results in the paper. If GPUs are not available to the AEC, we additionally provide scaled-down versions of these experiments. The scaled-down versions do not reproduce the exact results in the paper but demonstrate that the artifact runs correctly. Instructions for running them are included in the repository README.

3) Software dependencies: All experiments are implemented in Python. The artifact includes the following software-related components and requirements:

- **Python Environment:** Each experiment directory contains a `requirements.txt` file listing the Python packages needed. We also provide Dockerfiles for containerized setup. The repository README explains how to build and run these Docker environments.

- **API Tokens:** Some experiments require API access to external LLM providers (HuggingFace, OpenAI, Anthropic, and DeepSeek). The README explains the placement and use of all API keys.

- **Ollama (Experiment A.2):** Experiment A.2 requires an installation of Ollama (https://ollama.com/download). Setup and usage instructions are included in the README.

- **LM Studio (Experiment B.2):** Experiment B.2 requires an installation of LM Studio (https://lmstudio.ai/). Setup and usage instructions are included in the README.

4) Datasets:

- **Experiment A.1:** Uses the *New-Hate-Wave* dataset from HuggingFace. We provide this dataset directly to the AEC (via the GitHub repository), as access normally must be requested from the dataset authors on HuggingFace.

- **Experiments B.1, B.2, and C:** Use the *PrimeVul* [123], *Devign* [121], and *DiverseVul* [122] datasets. These are downloaded automatically from HuggingFace during execution.

- **Experiment D:** Uses the *self-oss-instruct-sc2-exec-filter-50k* dataset, which is also downloaded automatically from HuggingFace.

B. Artifact Installation & Configuration

The experiments are implemented in Python. Each experiment directory includes a `requirements.txt` file listing the necessary Python packages. For convenience and reproducibility, we also provide Dockerfiles that allow setting up the environment via Docker. The repository README details the steps for building the Docker images and running the experiments.

C. Experiment Workflow

Each experiment is self-contained (own directory, Dockerfile, requirements). Typical flow: *prepare environment → build & run experiment → collect artifacts in* `plots/`. Deviations from this flow are specified in the repository README.

D. Major Claims

- **Claim 1:** Model ambiguity affects performance and robustness (§IV-A). Supported by A.1 (hate detection accuracy shifts across GPT snapshots) and A.2 (robustness attack success rate differences across snapshots and quantization).

- **Claim 2:** Data leakage inflates metrics near-linearly in controlled laboratory settings (§IV-B). Supported by B.1.

- **Claim 3:** We find no evidence of memorization in the tested commercial and local models for widely used vulnerability detection datasets (§IV-B). Supported by B.2.

- **Claim 4:** Context truncation is common and can hide key input information (§IV-C). Supported by C.

- **Claim 5:** Iterative training on self-generated synthetic data leads to model collapse, indicated by increasing perplexity mean and variance across generations (§IV-D). Supported by D.

E. Evaluation

1) Experiment A.1:

- **Name:** Model Ambiguity & Surrogate Fallacy - Hate Detection

- **Effort:** 10 minutes of human effort; less than 1 hour evaluation time for the reduced evaluation; 4-5 hours evaluation time for the full evaluation.

- **Explanation:** We re-implement a hate speech detection experiment from previous literature. We evaluate the detection on three different snapshots of GPT-4 from OpenAI.

- **How to:** Setup and execution instructions are described in the README.

- **Results:** The results will be printed to `stdout`. The different snapshots of GPT-4 yield deviating results, supporting the claim in the paper.

2) Experiment A.2:

- **Name:** Model Ambiguity & Surrogate Fallacy - LLM Robustness

- **Effort:** 10 minutes of human effort; 2-24 hours, depending on the used models. Hosted LLMs like ChatGPT will be done faster, while local models require a GPU and more time; a scaled-down version with less attack iterations is included.

- **Explanation:** Evaluation of LLM robustness by initializing different LLMs with a "secret key" which is then tried to be exfiltrated via different attack strategies despite being instructed to keep it safe and secure.

- **How to:** Setup and execution instructions are described in the README.

- **Results:** Logs and results will be printed to `stdout` and saved to the `logs/` directory. Different model snapshots of the same base model yield different robustness when tested against confidentiality attacks. This supports the claim in the paper.

3) Experiment B.1:

- **Name:** Lab-Setting Data Leakage

- **Effort:** 10 minutes of human effort; 5 days compute time for full run on GPUs; 1-10 hours for scaled-down version on a commodity desktop machine.

- **Explanation:** The experiment fine-tunes CodeT5+ on the Devign/DiverseVul/PrimeVul datasets with leakage ratios {0, 0.2, ..., 1.0} added from the test set to the training set.

- **How to:** The experiment only requires two steps to run: set up the environment (HuggingFace token) and build/run the experiment. Both steps, including the exact commands, are described in the README in our artifact repository.

- **Results:** Plots will be saved to the `plots/` directory. For the full run, Figure 4 will be reproduced. For the scaled-down version, a monotonic increase of the F1 scores can be observed. Therefore, the claim (C2) is still supported.

4) Experiment B.2:

- **Name:** Commercial LLM Data Leakage

- **Effort:** 10 minutes of human effort; 1–5 hours compute time on a commodity desktop machine.

- **Explanation:** Commit-message and function completion on the PrimeVul dataset across commercial and local LLMs.

- **How to:** The experiment only requires two steps to run: set up the environment (API tokens) and build/run the experiment. Both steps, including the exact commands, are described in the README.

- **Results:** Results will be printed to standard output. Expect results consistent with those reported in the paper

(0/100 matches for all models), although minor variations may occur due to randomness and API endpoint behavior.

5) Experiment C:

- **Name:** Context Truncation

- **Effort:** 10 minutes of human effort; 5 minutes compute time on a commodity desktop machine.

- **Explanation:** Tokenize vulnerable functions and compute proportions exceeding context limits.

- **How to:** The experiment requires two steps: set up the environment (HuggingFace token) and build/run the experiment. Instructions are provided in the README.

- **Results:** The resulting table will be saved as a LaTeX file to the `plots/` directory. All numbers will exactly match Table IV in our paper.

6) Experiment D:

- **Name:** Model Collapse

- **Effort:** 10 minutes of human effort; 4-5 days compute time for full runs on GPUs; a scaled-down demonstration is included with only 20-24 hours of compute time for one single generation. (However, the scaled-down version still requires some kind of GPU to work. A free hosted solution may be sufficient.)

- **Explanation:** Iterative self-training on synthetic data and evaluation of perplexity mean and variance across generations.

- **How to:** Setup and execution instructions are described in the README.

- **Results:** The plot will be saved to the `plots/` directory. The trend of increasing perplexity mean and variance will match the claim demonstrated in the paper.

F. Notes

Some results may fluctuate slightly due to internal randomness and non-deterministic behavior of both LLM APIs as well as locally hosted LLMs.