

On Automated and Explainable Provenance of AI-Generated Code

Alejandro Velasco, Nathan Wintersgill, Trevor Stalnaker, Oscar Chaparro, Denys Poshyvanyk

School of Computing, Data Sciences and Physics

William & Mary - Williamsburg, VA, USA

{svelascodimate, njwintersgill, twstalnaker, oscarch, dposhyvanyk}@wm.edu

Abstract—Generative AI for code generation has transformed software development, but it has also introduced a critical transparency problem: the origins of AI-generated code are opaque to the developers who use it, the organizations that deploy it, and the compliance professionals responsible for ensuring its legal and quality standards. Existing mitigations flag problematic outputs after the fact without explaining why a model produced them or how future generation could be improved. We present a research vision, grounded in a U.S. NSF-funded research grant, that argues that the next generation of CodeGenAI tools must be built on a foundation of explainable provenance: automated, post-hoc traceability that links generated code back to the prompt components, training data instances, global data features, and internal model components that caused its generation. We grounded this vision in empirical evidence from studies of software developers, model users, and compliance/legal professionals, which show that provenance information is a practical necessity that current tools do not provide. We characterize the problem across four traceability dimensions, outline a research program combining large-scale empirical studies with post-hoc causal and interpretability techniques, and identify the key open challenges that the community must address to realize this vision.

Index Terms—Data Provenance, Generative AI, Code Generation, Source Code Analysis, Traceability

I. INTRODUCTION

Generative AI for code generation (CodeGenAI) has fundamentally transformed how developers create software. Tools such as Claude Code, GitHub Copilot, Tabnine, and ChatGPT allow practitioners to generate complex functionality from natural-language prompts, dramatically accelerating development and lowering barriers to entry for non-expert programmers [1], [2]. As developers increasingly delegate implementation decisions to AI models, the origins of the resulting code become opaque to the developers themselves, to their organizations, and to the compliance professionals responsible for ensuring legal and quality standards.

As developers increasingly rely on CodeGenAI, the benefits of this technology do not come without costs. Numerous studies and real-life cases show that CodeGenAI can introduce significant security and legal risks, many of which can lead to financial, reputational, and legal consequences for organizations and society [3]–[6]. Specifically, CodeGenAI models have been shown to reproduce verbatim fragments of copyrighted code [3], expose sensitive information memorized from training data [6], and generate insecure or low-quality code at scale [4], [5]. The ongoing lawsuit *Doe v. GitHub* [7]

has placed the legal status of AI-generated code under direct judicial scrutiny, and regulatory mandates now require Software Bills of Materials (SBOMs) to document the provenance of software components, including those produced with AI assistance: the US Executive Order on Cybersecurity [8] mandated SBOMs for federal software procurement, and the EU Cyber Resilience Act [9] makes SBOMs a legally binding requirement for all products with digital elements placed on the market.

Tracing the origins of AI-generated code is an extraordinarily difficult problem. Three compounding factors contribute to this difficulty. First, the *CodeGenAI supply chain* is long and opaque, spanning dataset curation, model pre-training, fine-tuning, deployment, and prompting, with each stage performed by different actors and each capable of influencing model output in ways that are difficult to disentangle [10]. It also often contains deficiencies in documentation, such as missing references to necessary components like datasets [11]. Second, the *black-box nature* of large language models (LLMs) makes it difficult to inspect why a particular prompt yields a particular output, and understanding model behavior requires interpretability techniques that are still maturing [12]. Third, *training data* are often inaccessible, withheld for legal or commercial reasons, making it impossible to directly verify whether a generated snippet originates from a specific source [13].

Existing mitigation strategies address this problem only partially. Filtering layers and specialized detection tools such as CopyrightCatcher flag problematic output after generation, without explaining *why* a model produced them or informing how future generation could be improved [14]. Attribution-by-link approaches, in which AI assistants cite web sources along with generated code, have been shown to produce largely irrelevant references [15]. No existing solution provides developers with an actionable *and explainable* traceability between AI-generated code and the factors that caused its generation.

This paper presents a research vision grounded in research grant IIS-2533367 [16] funded by the U.S. National Science Foundation (NSF). We argue that the next generation of CodeGenAI tools must be built on a foundation of *explainable provenance*: automated traceability that links generated code back to its contributing causes, including prompt components, training data instances and their global characteristics, and internal model components, presented in a form that enables developers and compliance practitioners to make informed

decisions about the code they integrate. This vision requires advances along two fronts: (i) a deeper empirical understanding of what provenance information different stakeholders actually need and how they currently seek it, and (ii) new post-hoc techniques that provide explainable traceability without requiring model retraining.

II. THE NEED FOR EXPLAINABLE PROVENANCE

Understanding the origins of AI-generated code has become a practical necessity for three groups of stakeholders who occupy distinct positions in the CodeGenAI supply chain: *model developers*, who curate training data and build and fine-tune models; *model users*, who integrate AI-generated code into software projects; and *compliance practitioners*, who audit AI-generated artifacts against legal and organizational requirements [10], [17].

Evidence from practitioners. Our past survey of 574 software developers worldwide found that while 89.7% regularly use CodeGenAI tools, 70.1% lack or do not have processes to document their use, creating significant traceability gaps [18]. Follow-up interviews revealed that developers resort to manual searches on GitHub and Stack Overflow to verify the origins of AI-generated code, a practice that is both inefficient and unreliable at scale. We conducted another survey of 138 software practitioners across five stakeholder groups. The survey established that tracking software provenance is a critical concern, particularly for AI systems where missing lineage information can obscure issues such as model poisoning, training data bias, or inadvertent inclusion of harmful content [17]. The legal professionals surveyed in a complementary study reinforced this picture [19]: we found that provenance information is a prerequisite for maintaining license compliance, and existing tools such as Black Duck and FOSSology do not offer coverage of AI-generated artifacts.

What provenance must explain. The information needs of these stakeholders converge on four traceability dimensions. *Prompt-to-output traceability* concerns which components of the prompt contribute to the generation of non-compliant or low-quality code. *Instance-to-output traceability* concerns which training data or external code instances are associated with a given output. *Factor-to-output traceability* concerns the global characteristics of the training data, such as the prevalence of restrictively licensed code or specific syntactic constructs, that lead the model to learn patterns producing problematic outputs. Finally, *model component-to-output traceability* concerns the internal components of the model, such as specific neuron circuits or attention heads, responsible for a given generation. Together, these four dimensions define the scope of explainable provenance as we envision it.

Why current approaches fall short. Provenance methods developed for traditional software, including clone detection, hashing, and software bertillonage [20], [21], were designed for settings where copying produces an identifiable artifact. In the generative setting, the relationship between a model output and any contributing source is probabilistic, mediated by billions of learned parameters, and shaped by prompt context that

shifts with every query. Intrinsic source attribution, which embeds citation information during training, requires access to the training pipeline and remains prone to citation hallucinations [22]. Post-hoc retrieval approaches can surface similarities between generated and existing code, but stop short of explaining the causal role of training data characteristics or model components in producing a given output [23]. The result is a significant gap: stakeholders have no principled, automated way to understand the causes behind a specific AI-generated output, nor guidance on how to adapt prompts, curate training data, or refine models to prevent problematic generations in the future.

III. A RESEARCH VISION FOR EXPLAINABLE PROVENANCE

We envision a research program organized around two interconnected goals. The first is empirical: to develop a comprehensive understanding of how stakeholders currently engage with provenance during the development and use of CodeGenAI systems, and what information they need to do so effectively. The second is technical: designing automated, post-hoc techniques that provide explainable traceability between AI-generated code and the artifacts and factors that caused its generation. Fig. 1 illustrates the overall research vision.

A. Comprehensive Surveys and Interviews for Understanding Stakeholder Provenance Needs

Despite the growing adoption of CodeGenAI tools, little is known about how model developers, model users, and compliance practitioners engage with provenance in practice. *What information do they seek when evaluating AI-generated code? What challenges prevent them from obtaining it? What would effective tool support look like from their perspective?* These questions remain largely unanswered, and their answers are a prerequisite for designing provenance techniques that are both technically sound and practically useful.

To address this gap, we propose a large-scale empirical study that combines an online survey and in-depth follow-up interviews with the three stakeholder groups. Each group plays a distinct role in the AI supply chain. First, *model developers* curate datasets and train, fine-tune, and deploy CodeGenAI models. Second, *model users* integrate AI-generated code into software projects via prompting. Third, *compliance practitioners*, including legal professionals and Open Source Program Office (OSPO) staff, evaluate AI-generated artifacts against legal and organizational mandates [19].

The study will be conducted iteratively over two years, allowing findings from early rounds to sharpen the questions posed in later ones. Survey questionnaires will be designed to uncover practices, needs, and challenges in provenance across the AI supply chain lifecycle, covering topics such as how stakeholders document the sources of training data, what metadata they need about AI-generated outputs to reason about quality and compliance, what traceability mechanisms would make their workflows more efficient, and what challenges they face in tracing low-quality or non-compliant code back to its origins. Respondents who provide insightful answers will be

Empirical Component: Comprehensive Surveys and Interviews for Understanding Stakeholder Provenance Needs

Technical Component: Automated Techniques for Explainable Traceability of AI-generated code

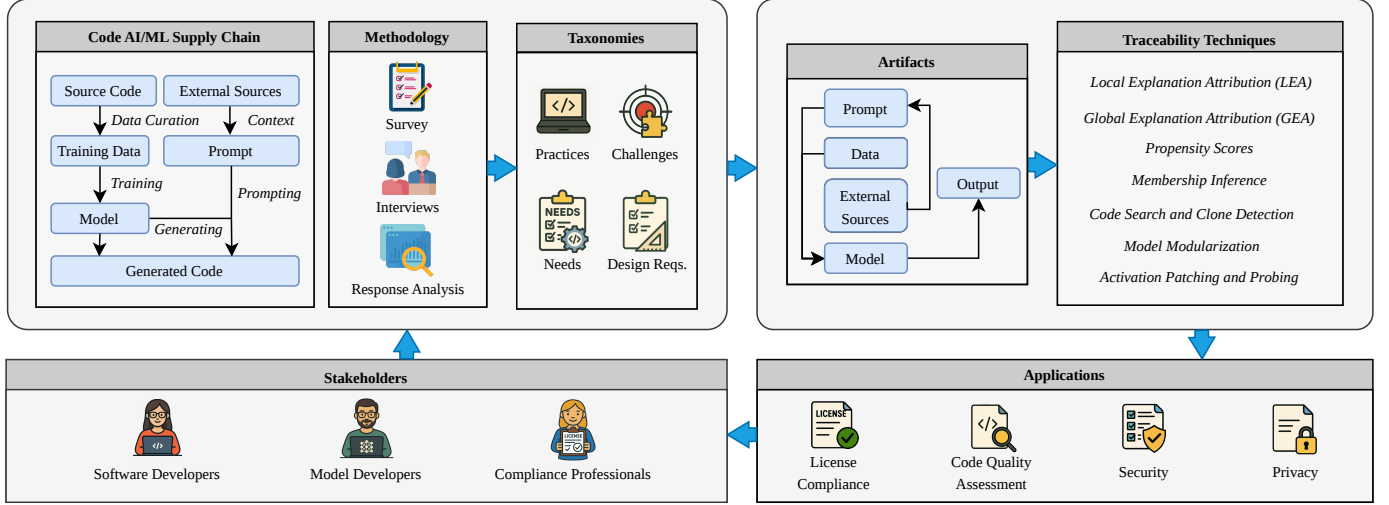


Fig. 1: Overview of the proposed research vision.

invited to in-depth follow-up interviews of 40–60 minutes to collect richer, contextualized information about their specific needs and practices.

Responses will be analyzed qualitatively using open coding and card sorting to develop structured taxonomies [17], [18]. These taxonomies will capture provenance practices, needs, challenges, and design requirements for each stakeholder group, organized around the stages of the AI supply chain and the artifact types involved, from raw training data and model checkpoints to prompts and generated code. The design requirements will inform the development of the envisioned techniques described in Sec. III-B. Inter-rater reliability will be measured throughout, and taxonomy development will proceed until theoretical saturation is reached. The resulting taxonomies will define the functional and non-functional requirements that the techniques described next must satisfy.

B. Automated Techniques for Explainable Traceability

The technical core of the vision is a suite of post-hoc traceability techniques that address each of the four provenance dimensions introduced in Sec. II.

Prompt-to-output traceability. We propose two complementary techniques for linking prompts to generated outputs. The first is a *propensity scoring* approach that analyses token-level log-probabilities from the model’s output distribution to estimate the likelihood that a given prompt will lead to non-compliant or low-quality code [24], [25]. The second is **Local Explanation Attribution (LEA)**, a post-hoc interpretability technique that identifies the prompt tokens most responsible for a problematic generation [24]–[27], using feature attribution methods such as SHAP [28] or rationale-based approaches [29], [30]. As illustrated in Fig. 2, LEA proceeds in three steps: identifying the generated code fragment that is non-compliant or low-quality, applying feature attribution to rank the prompt tokens by their influence on that fragment, and generating a

natural-language explanation that model users can act on to refine their prompts.

Instance-to-output traceability. Linking generated code to specific training data or external code instances requires a retrieval pipeline that combines traditional clone detection, neural similarity models, and membership inference attacks [26], [31]. The pipeline adapts to three scenarios depending on data availability. When the training set is accessible, similarity-based retrieval can be applied directly against indexed training instances. When it is not, web-scale search using code features extracted from the generated output serves as a proxy. When the model provides source links, those links are crawled, parsed, and verified through clone detection to assess their relevance and associated licensing terms [15]. In all scenarios, causal analysis of training dynamics can further reveal which artifact types carry the highest leakage risk, informing where retrieval efforts should be concentrated [27].

Factor-to-output traceability. Understanding how global characteristics of the training data influence model behavior requires moving beyond instance-level retrieval to causal analysis. We propose **Global Explanation Attribution (GEA)**, a framework that extends causal inference techniques developed for code models [32] to the provenance setting. As shown in Fig. 3, *GEA* uses causal graphs to model the relationships between training data factors (treatments Z), such as code complexity, syntactic constructs, and license distributions, and the propensity of the model to generate problematic outputs (potential outcomes Y^Z), while accounting for confounders X , instrumental variables I , hidden common causes X^0 , and effect modifiers E . By estimating average treatment effects (ATEs) through do-calculus and standard adjustment techniques, *GEA* produces actionable, interpretable insights into how training data composition shapes model behavior, without requiring model retraining.

Model component-to-output traceability. At the finest

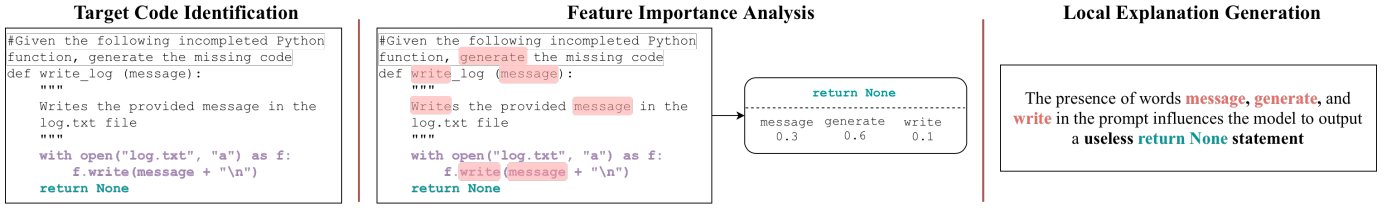


Fig. 2: Workflow of Local Explanation Attribution (LEA). Given a prompt and a smelly generated fragment (highlighted in green), SHAP values identify the prompt tokens most responsible for the problematic output, and a natural-language explanation is generated for the developer.

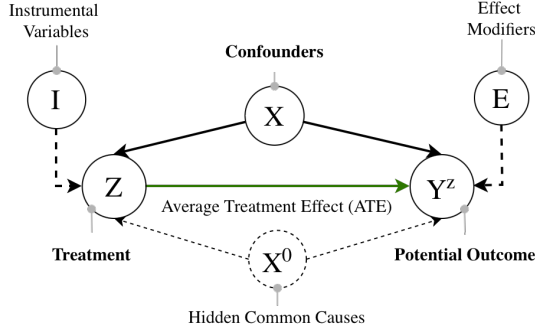


Fig. 3: Structure of the causal graphs used by *GEA*. Treatments Z represent training data factors; confounders X account for contextual variables such as model architecture and data distribution; the green arrow represents the Average Treatment Effect (ATE) estimated via do-calculus.

level of granularity, provenance requires linking generated outputs to specific internal components of the model, which is possible only for open-weight models. We propose to investigate three complementary approaches: functional model modularization, which identifies sub-networks associated with specific behaviors [33]; sparse autoencoders, which extract interpretable low-dimensional representations from internal activations [34]; and activation patching combined with probing, which tests the effect of modifying specific neuron circuits on model outputs and identifies which internal representations encode patterns associated with non-compliant or low-quality code [35].

IV. CHALLENGES AND OPEN QUESTIONS

Realizing the vision of an explainable provenance for AI-generated code requires addressing several fundamental challenges that cross the empirical and technical dimensions of the research program.

Defining provenance granularity. Provenance information can be provided at many levels of granularity, from individual tokens to entire functions, from specific training instances to broad data characteristics, from single attention heads to functional model modules. The appropriate granularity depends on the stakeholder and the task at hand. A model developer diagnosing systematic quality issues needs different information from a compliance practitioner auditing a specific

code snippet for license violations. Determining which granularities are most actionable for each stakeholder group and designing techniques that can operate across multiple levels of granularity without prohibitive computational cost remain an open problem that the empirical component of the research program must inform.

Scalability under training data inaccessibility. Many of the most widely used CodeGenAI models are proprietary and their training data is withheld for legal or commercial reasons. Provenance techniques that assume access to training data are therefore inapplicable to a large and growing fraction of real-world deployment scenarios. Membership inference attacks offer a partial solution, but their accuracy is reduced for large models with diverse training corpora, and provide statistical evidence rather than definitive attribution [31]. Developing provenance techniques that remain informative and reliable in black-box or grey-box settings, where only model outputs or output probabilities are accessible, represents one of the most pressing open challenges in this space.

Causal attribution in the presence of confounders. Establishing causal links between training data characteristics and model output is complicated by the presence of confounders that are difficult to observe or control. Model architecture, training hyperparameters, tokenization strategies, and data preprocessing decisions all influence model behavior in ways that interact with training data factors in complex, non-linear ways. Causal graphs constructed from domain knowledge may miss important confounders, leading to biased estimates of treatment effects [32]. Developing robust causal discovery procedures that can identify relevant confounders from data and sensitivity analyses that quantify the impact of unobserved confounding on provenance conclusions will be essential for the credibility of the *GEA* framework.

Evaluating provenance without ground truth. Provenance techniques require evaluation, but ground truth is difficult to obtain in realistic settings. For commercial models, the true training data is unknown. For open-source models, the relationship between a specific generated output and any particular training instance is probabilistic rather than deterministic. Constructing evaluation benchmarks that simulate realistic provenance scenarios, with controlled license violations and known causal relationships between training data and output, requires careful design to avoid artificial simplifications that do not reflect the complexity of real-world code generation [24]. Hu-

man evaluation is an alternative, but inter-annotator agreement on provenance judgments, particularly for license compliance, is itself a research question given the inherent ambiguity of copyright law [19].

Bridging technical provenance and legal reasoning. Even technically precise provenance information may not translate directly into legally actionable conclusions. The copyright law does not operate on similarity scores or membership inference probabilities; it requires judgments about substantial similarity, the expression of ideas rather than the ideas themselves, and the intent of the parties involved [3]. Designing provenance tools that present technical evidence in forms that are interpretable and useful to legal professionals, without overstating or understating their significance, requires close collaboration between software engineering researchers and legal scholars. This interdisciplinary challenge may ultimately prove as difficult as the technical problems the tools are designed to solve.

V. RELATED WORK

Our vision is based on four areas: empirical studies of AI-assisted development, software provenance and compliance, source attribution for generated code, and interpretability of code models. Each contributes motivation or technical building blocks, but none provides end-to-end, post-hoc traceability across all four provenance dimensions we identify.

AI-assisted programming and risks. Studies of GitHub Copilot and related tools show that developers use CodeGenAI for completion, testing, debugging, and review [2], but do not consistently document the presence or origin of AI-generated code. Previous work establishes that generated code can introduce security vulnerabilities [4], code smells [5], [24], and memorized training fragments [6]. Detecting problematic snippets is insufficient: developers and compliance practitioners also need to understand which prompt elements, training examples, and model components contributed to them.

Provenance, SBOMs, and compliance. Clone detection, dependency analysis, and software bill of materials [20], [21] support provenance recovery for traditional software. Work on Software Bills of Materials shows that stakeholders need structured lineage across the supply chain [17]. Legal and empirical studies further show that practitioners need provenance evidence connected to legal concepts such as attribution and substantial similarity [3], [10], [18], [19]. These approaches assume provenance is attached to identifiable artifacts. CodeGenAI breaks this assumption: a generated function may reflect many training instances and prompt choices without being a direct clone of any single source.

Source attribution and code inclusion. Source-aware training teaches models to associate outputs with source identifiers [22], but requires access to the training pipeline. Post-hoc approaches are therefore necessary. AI assistants that provide source links have been shown to produce largely irrelevant references [15]. Model-agnostic techniques can detect whether a code artifact was included in a training set [23], and membership inference attacks provide complementary statistical

evidence [31]. Recent work further shows that semantically equivalent code transformations can be used to probe the robustness of membership inference for code models [26], while causal analysis of training dynamics reveals that leakage risks differ substantially across personally identifiable information types and correlate with how easily each type is learned [27]. These works address instance-to-output traceability but do not explain how prompt choices, corpus-level factors, or model internals caused a given output.

Interpretability for code models. Feature-attribution and rationale-based methods identify which prompt tokens most influence a model prediction [28], [30]. Causal frameworks for code models distinguish correlation from causal influence [32], motivating our GEA direction. Recent work applies causal analysis to measure, explain, and mitigate smell propensity in LLM-generated code, showing that prompt formulation and model architecture causally shape structural quality [25]. At the model-internal level, sparse autoencoders and activation patching can localize learned behaviors to functional modules and internal circuits [34], [35], providing a foundation for model component-to-output traceability. These techniques have not yet been connected to provenance tasks such as license compliance or code quality audit.

Existing approaches address individual levels of provenance evidence in isolation. Our agenda treats provenance as a multi-level explanation problem spanning all four traceability dimensions, operating post-hoc without access to proprietary training pipelines.

VI. CONCLUSIONS & FUTURE PLANS

We have presented a vision for explainable provenance of AI-generated code, framed as a multi-level traceability problem spanning prompt components, training data instances, global data features, and internal model components. Empirical evidence shows that current practices leave significant provenance gaps for all three stakeholder groups in the CodeGenAI supply chain, and this vision motivates a research program along two complementary fronts. The empirical front will produce taxonomies of stakeholder provenance needs, challenges, and design requirements through large-scale surveys and interviews, while the technical front will develop and evaluate post-hoc traceability techniques across all four provenance dimensions. Several open challenges remain (Sec. IV), from determining actionable provenance granularities to bridging the gap between technical evidence and legal reasoning. We invite the community to engage with this vision and contribute to defining the methods and evaluation standards that explainable provenance will require.

VII. ACKNOWLEDGMENTS

This work has been funded by the U.S. NSF award IIS-2533367 [16]. We would like to thank Massimiliano Di Penta from the University of Sannio for his valuable contributions to the grant proposal. The opinions, findings and conclusions expressed in this work are solely those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

- [1] C. Bird, D. Ford, T. Zimmermann, N. Forsgren, E. Kalliamvakou, T. Lowdermilk, and I. Gazit, "Taking Flight with Copilot: Early insights and opportunities of AI-powered pair-programming tools," *ACM Queue*, Jan. 2023.
- [2] J. T. Liang, C. Yang, and B. A. Myers, "A large-scale survey on the usability of ai programming assistants: Successes and challenges," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, (New York, NY, USA), Association for Computing Machinery, 2024.
- [3] Z. Kahveci, "Attribution problem of generative ai: a view from us copyright law," *Journal of Intellectual Property Law and Practice*, vol. 18, pp. 796–807, 09 2023.
- [4] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the keyboard? assessing the security of github copilot's code contributions," *Commun. ACM*, vol. 68, p. 96–105, Jan. 2025.
- [5] M. L. Siddiq, S. H. Majumder, M. R. Mim, S. Jajodia, and J. C. S. Santos, "An empirical study of code smells in transformer-based code generation techniques," in *2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pp. 71–82, 2022.
- [6] Z. Yang, Z. Zhao, C. Wang, J. Shi, D. Kim, D. Han, and D. Lo, "Unveiling memorization in code models," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, (New York, NY, USA), Association for Computing Machinery, 2024.
- [7] DOE 1 et al., "DOE 1 et al. v. GitHub, Inc., et al.," 2022. Case No. 4:22-cv-06823 (N.D. Cal.).
- [8] "Executive Order 14028, Improving the Nation's Cybersecurity," *NIST*, Apr. 2021. Last Modified: 2022-07-11T15:36:04:00.
- [9] "Regulation - 2024/2847 - EN - EUR-Lex."
- [10] K. Lee, A. F. Cooper, and J. Grimmelmann, "Talkin' 'bout ai generation: Copyright and the generative-ai supply chain (the short version)," in *Proceedings of the 2024 Symposium on Computer Science and Law*, CSLAW '24, (New York, NY, USA), p. 48–63, Association for Computing Machinery, 2024.
- [11] T. Stalnak, N. Wintersgill, O. Chaparro, L. A. Heymann, M. Di Penta, D. M. German, and D. Poshyvanyk, "An empirical analysis of machine learning model and dataset documentation, supply chain, and licensing challenges on hugging face," *ACM Trans. Softw. Eng. Methodol.*, Nov. 2025.
- [12] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," 2017.
- [13] S. Longpre, R. Mahari, N. Obeng-Marnu, W. Brannon, T. South, J. Kabbara, and S. Pentland, "Data Authenticity, Consent, and Provenance for AI Are All Broken: What Will It Take to Fix Them?," *An MIT Exploration of Generative AI*, mar 27 2024. <https://mit-genai.pubpub.org/pub/uk7op8zs>.
- [14] R. Patel, "Introducing Code Referencing for GitHub Copilot Chat in Visual Studio," Dec. 2024.
- [15] D. Bifolco, P. Cassieri, G. Scanniello, M. D. Penta, and F. Zampetti, "Do llms provide links to code similar to what they generate? a study with gemini and bing copilot," in *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*, pp. 223–235, 2025.
- [16] "Automated and explainable provenance of ai-generated code." https://www.nsf.gov/awardsearch/show-award/?AWD_ID=2533367, 2026.
- [17] T. Stalnak, N. Wintersgill, O. Chaparro, M. Di Penta, D. M. German, and D. Poshyvanyk, "Boms away! inside the minds of stakeholders: A comprehensive study of bills of materials for software systems," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, (New York, NY, USA), Association for Computing Machinery, 2024.
- [18] T. Stalnak, N. Wintersgill, O. Chaparro, L. A. Heymann, M. D. Penta, D. M. German, and D. Poshyvanyk, "Developer perspectives on licensing and copyright issues arising from generative ai for software development," *ACM Trans. Softw. Eng. Methodol.*, vol. 35, Jan. 2026.
- [19] N. Wintersgill, T. Stalnak, L. A. Heymann, O. Chaparro, and D. Poshyvanyk, "the law doesn't work like a computer": Exploring software licensing issues faced by legal practitioners," *Proc. ACM Softw. Eng.*, vol. 1, July 2024.
- [20] M. W. Godfrey, D. M. German, J. Davies, and A. Hindle, "Determining the provenance of software artifacts," in *Proceedings of the 5th International Workshop on Software Clones*, IWSC '11, (New York, NY, USA), p. 65–66, Association for Computing Machinery, 2011.
- [21] D. M. German, M. Di Penta, Y.-G. Gueheneuc, and G. Antoniol, "Code siblings: Technical and legal implications of copying code between applications," in *2009 6th IEEE International Working Conference on Mining Software Repositories*, pp. 81–90, 2009.
- [22] M. Khalifa, D. Wadden, E. Strubell, H. Lee, L. Wang, I. Beltagy, and H. Peng, "Source-Aware Training Enables Knowledge Attribution in Language Models," *arXiv preprint arXiv:2404.01019*, 2024.
- [23] V. Majdinasab, A. Nikanjam, and F. Khomh, "Trained without my consent: Detecting code inclusion in language models trained on code," *ACM Trans. Softw. Eng. Methodol.*, vol. 34, Apr. 2025.
- [24] A. Velasco, D. Rodriguez-Cardenas, L. R. Alif, D. N. Palacio, and D. Poshyvanyk, "How propense are large language models at producing code smells? a benchmarking study," in *2025 IEEE/ACM 47th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, p. 96–100, IEEE Press, 2025.
- [25] A. Velasco, D. Rodríguez-Cárdenas, D. Khati, D. N. Palacio, L. R. Alif, and D. Poshyvanyk, "A causal perspective on measuring, explaining and mitigating smells in llm-generated code," in *Proceedings of the 48th International Conference on Software Engineering (ICSE '26)*, November 2026. To appear.
- [26] H. Yang, A. Velasco, T. Le-Cong, M. N. Haque, B. Xu, and D. Poshyvanyk, "How do semantically equivalent code transformations impact membership inference on LLMs for code?," in *Proceedings of the 48th International Conference on Software Engineering (ICSE '26)*, November 2026. To appear.
- [27] H. Yang, A. Velasco, S. Fang, B. Xu, and D. Poshyvanyk, "Understanding privacy risks in code models through training dynamics: A causal approach," *arXiv preprint arXiv:2512.07814*, 2025.
- [28] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, (Red Hook, NY, USA), p. 4768–4777, Curran Associates Inc., 2017.
- [29] D. Khati, D. Rodriguez-Cardenas, D. N. Palacio, A. Velasco, M. Tufano, and D. Poshyvanyk, "Enabling Global, Human-Centered Explanations for LLMs: From Tokens to Interpretable Code and Test Generation," in *Proceedings of the 48th International Conference on Software Engineering (ICSE '26)*, November 2026. To appear.
- [30] K. Vafa, Y. Deng, D. M. Blei, and A. M. Rush, "Rationales for sequential predictions," in *Empirical Methods in Natural Language Processing*, 2021.
- [31] H. Hu, Z. Salicic, L. Sun, G. Dobbie, P. S. Yu, and X. Zhang, "Membership inference attacks on machine learning: A survey," *ACM Comput. Surv.*, vol. 54, Sept. 2022.
- [32] D. Nader Palacio, A. Velasco, N. Cooper, A. Rodriguez, K. Moran, and D. Poshyvanyk, "Toward a Theory of Causation for Interpreting Neural Code Models," *IEEE Transactions on Software Engineering*, vol. 50, pp. 1215–1243, May 2024.
- [33] S. M. Imtiaz, F. Batole, A. Singh, R. Pan, B. D. Cruz, and H. Rajan, "Decomposing a recurrent neural network into modules for enabling reusability and replacement," in *Proceedings of the 45th International Conference on Software Engineering*, ICSE '23, p. 1020–1032, IEEE Press, 2023.
- [34] T. Bricken, A. Templeton, J. Batson, B. Chen, A. Jermyn, T. Conerly, N. Turner, C. Anil, C. Denison, A. Askell, R. Lasenby, Y. Wu, S. Kravec, N. Schiefer, T. Maxwell, N. Joseph, Z. Hatfield-Dodds, A. Tamkin, K. Nguyen, B. McLean, J. E. Burke, T. Hume, S. Carter, T. Henighan, and C. Olah, "Towards monosemanticity: Decomposing language models with dictionary learning," *Transformer Circuits Thread*, 2023. <https://transformer-circuits.pub/2023/monosemantic-features/index.html>.
- [35] K. Meng, D. Bau, A. Andonian, and Y. Belinkov, "Locating and editing factual associations in gpt," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, (Red Hook, NY, USA), Curran Associates Inc., 2022.