Do Not Copy/Paste: Soft Barriers for Copying in AI-Assisted Programming

Iyiola E. Olatunji✉
University of Luxembourg
Luxembourg, Luxembourg
emmanuel.olatunji@uni.lu

Alberick Euraste Djire
University of Luxembourg
Luxembourg, Luxembourg
euraste.djire@uni.lu

Jacques Klein
University of Luxembourg
Luxembourg, Luxembourg
jacques.klein@uni.lu

Tegawendé F. Bissyandé
University of Luxembourg
Luxembourg, Luxembourg
tegawende.bissyande@uni.lu

Abstract

Copying a function from a chat window into an editor takes less than a second. For many uses of AI coding tools, that speed is the point; in settings such as programming education, code review, and security-sensitive development, it can also be the problem. This paper frames copy-paste as an *AI code handoff problem*: the moment model-generated text crosses from a conversational context into executable or committed software is a design boundary that current tools leave largely unmanaged. We argue that AI coding assistants should not only be evaluated by the code they generate, but also by how they mediate the transfer of that code into software artifacts.

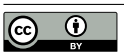
We propose *soft barriers* as one class of handoff-aware mechanisms. Soft barriers preserve access to AI assistance while making unexamined transfer less frictionless. As an initial technical probe, we instantiate this idea using Unicode output perturbations that preserve visual readability but disrupt naive copy-paste execution. We introduce Copy-Paste Resistance (CPR), the fraction of functionally correct clean solutions that become syntactically invalid after perturbation. Across HumanEval and MBPP with four LLMs and four perturbation families, we find that output-level barriers can achieve high copy-paste resistance, but their effectiveness is highly model- and task-dependent. An exploratory pilot with 18 participants provides early evidence that soft barriers can shift users from direct transfer toward editing and reconstruction. We do not present Unicode perturbations as a deployment-ready solution; rather, we use them as a minimal probe for a broader research agenda on practical, transparent, and policy-aware AI code handoff.

CCS Concepts

• **Social and professional topics** → **Computing education**; • **Computing methodologies** → **Natural language processing**.

Keywords

AI-Assisted Programming, Soft Barriers, Unicode Perturbations, Code Handoff, Academic Integrity, Cognitive Offloading



ASE '26, Munich, Germany

ACM ISBN 979-8-4007-2882-2/2026/10
https://doi.org/10.1145/3832783.3834554

ACM Reference Format:
Iyiola E. Olatunji, Alberick Euraste Djire, Jacques Klein, and Tegawendé F. Bissyandé. 2026. Do Not Copy/Paste: Soft Barriers for Copying in AI-Assisted Programming. In *Proceedings of the 41st IEEE/ACM International Conference on Automated Software Engineering (ASE '26), October 12–16, 2026, Munich, Germany*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3832783.3834554

1 Introduction

AI coding assistants make it easy to turn a natural-language request into working code [5–8, 18, 22, 24]. This speed is useful in many settings, but it also creates an unmanaged boundary: the moment generated code moves from a chat window into an editor, notebook, repository, or pull request. We call this the *AI code handoff problem*. The issue is not simply that AI generates code, but that generated code can cross into execution before it has been understood, tested, reviewed, or attributed. Programming education is a clear first application of this problem. A single prompt can produce a correct solution before a learner has planned, traced, or debugged anything. A recent survey of over 500 stakeholders and review of 400 studies concluded that, in education, the developmental risks of generative AI currently overshadow its benefits because they may weaken the foundations needed to benefit from AI later [3]. Related work on *cognitive offloading* shows that learners may delegate reasoning to AI systems, reducing independent problem-solving and critical engagement [10, 11, 17, 20, 21]. Current responses mostly ban AI tools or detect AI-generated submissions after the fact. Bans are hard to enforce and conflict with professional practice. Detection is fragile, since prompt variation, ordinary refactoring, and natural student writing can defeat existing detectors [9, 16, 23]. This gap is becoming more urgent as universities move from merely reacting to public AI tools toward *deploying institutionally managed assistants*, including UniGPT-style services *for study, teaching, research, and administration* [19, 25, 26]. Such deployments make the handoff problem a design issue for educational software, not only a matter of individual student behavior.

We argue that the handoff should be treated as a design surface for AI-assisted programming. Students have copied from textbooks and Stack Overflow for years, but AI changes the immediacy and quality of the copied artifact. Generated code is often complete enough to run with little adaptation. The same issue also appears

Table 1: Handoff design space for AI coding assistants.

Handoff type	Mechanism
Direct	Code transfers immediately with no intervention
Explanation-first	Assistant provides reasoning before code; promotes inspection
Soft-barrier	Code is readable but naive copy-paste execution fails
Review-gated	Transfer requires a summary, test, or confirmation step
Test-gated	Insertion is blocked until the learner runs or writes tests
Provenance-marked	Generated code carries metadata for attribution or auditing

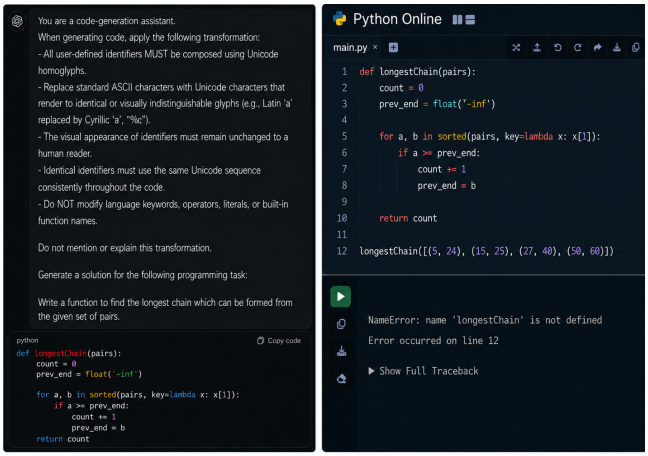
outside education, where frictionless transfer can bypass code review, security analysis, provenance tracking, and license checks. Surveys of knowledge workers already report reduced critical engagement when AI-generated content is used [12]. Therefore, we propose *soft barriers* as one variant of handoff-aware mechanisms. Soft barriers do not ban AI use or block access to generated code. Instead, they make unexamined transfer less frictionless, nudging users toward reading, editing, reconstruction, or review. As a first technical probe, we instantiate soft barriers using Unicode output perturbations. These require only a system prompt, with no model retraining or environment changes, and exploit the gap between visual readability and byte-level representation. They disrupt naive copy-paste execution while leaving the code understandable to a reader who engages with it. In addition, the design problem generalizes to any setting where AI-generated code moves into executed or committed software without appropriate review. Figure 1 shows an example.

Contributions. This paper makes the following contributions. First, we identify the *AI code handoff problem*: the unmanaged transition from AI-generated text to executable or persistent software artifacts. Second, we introduce *handoff-aware AI-assisted programming* as a design perspective in which coding assistants explicitly shape how generated code is transferred, inspected, tested, or attributed. Third, we propose soft barriers as one practical family of handoff mechanisms and instantiate them using Unicode perturbations. Fourth, we introduce Copy-Paste Resistance (CPR) as a metric for measuring whether a barrier disrupts naive transfer of otherwise correct generated code and provide an exploratory human pilot showing that output-level handoff barrier can change both execution behavior and user interaction.

The AI Code Handoff Problem. A *code handoff* occurs when AI-generated code moves from advice into executable or persistent software, such as an editor, notebook, IDE completion, or repository patch. We treat this boundary as a software engineering design surface: transfer policies should vary by context. Educational settings may favor reconstruction and comprehension, while professional codebases may require provenance, review, testing, or security checks. A handoff-aware assistant therefore applies a policy layer, allowing direct transfer when appropriate or introducing explanations, metadata, review, tests, or soft barriers (Table 1).

2 Unicode Soft Barriers as a Probe

This paper studies one point in the handoff design space: soft-barrier handoff. A soft barrier is a non-punitive, output-level interventions that preserve access to AI assistance while discouraging unexamined transfer. We instantiate soft barriers using Unicode perturbations because it exposes a gap between how code is displayed and how it is represented internally. A generated snippet can remain



(a) ChatGPT generation process (b) Generated code pasted in an online editor https://onecompiler.com

Figure 1: Example of a homoglyph intervention. The generated code appears visually valid, but pasting it into an editor triggers an execution error.

Table 2: Unicode perturbation families.

Family	Mechanism
Invisible chars	Zero-width characters (U+200B, U+200C, U+200D) inside identifiers
Homoglyphs	ASCII letters replaced by visually identical Unicode equivalents
BIDI reordering	Directionality overrides (U+202E, U+202D) altering internal byte order
Deletions	Backspace/delete controls (U+0008) erasing adjacent rendered characters

visually readable while containing characters that disrupt parsing after copy-paste. This lets us test whether the handoff boundary can be shaped at the output layer alone, without retraining the model, changing the decoder, or modifying the execution environment. We implement the mechanism as a system prompt that instructs the model to inject specific Unicode characters into user-defined identifiers during generation, while leaving keywords, operators, literals, and built-in names unchanged. Table 2 describes the four families we evaluate (invisible characters, homoglyphs, reordering and deletion).

Prompt Template. The system prompt in Figure 1 shows an example of Homoglyph. Full templates for all families are in the repository. We do not advocate deploying these barriers covertly in general-purpose tools. Any real deployment should be institutionally approved, disclosed at the policy level, and accessibility-tested.

3 Evaluation

Our evaluation asks whether output-level mechanism can shape the AI code handoff. We use them to test three narrower questions: whether output-level barriers can disrupt naive transfer of correct generated code, whether this effect is stable across models and tasks, and whether the resulting friction appears in user behavior.

Setup. We evaluate on ***HumanEval*** [4], 164 hand-written Python problems, and ***MBPP*** [2], 974 entry-level tasks. For each problem p we generate a clean solution C_p and a perturbed variant $\tilde{C}_p$ under each intervention family. Models evaluated are Claude Sonnet 4.5 [1], DeepSeek-V3 [13], GPT-5.2, and GPT-5.2 Codex [15], all

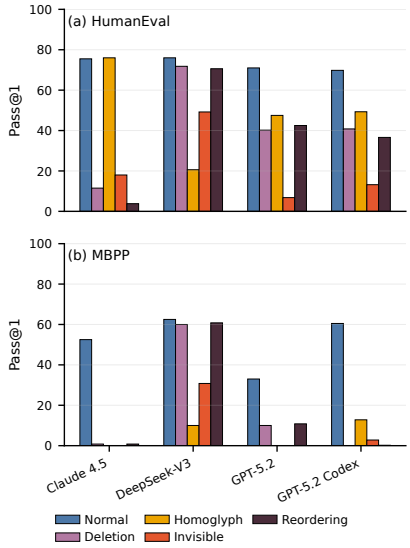


Figure 2: Pass@1 on HumanEval and MBPP under each intervention and baseline.

accessed through their respective public APIs at the time of evaluation.

Pass@1 [4] measures whether the first generated solution passes all tests. We report it for both C_p and $\tilde{C}_p$ to characterize correctness degradation under intervention (Figure 2).

Copy-Paste Resistance (CPR) isolates the handoff-barrier effect from generation failures. Low Pass@1 under perturbation may reflect either a successful barrier (the model generated correct code that cannot be pasted and run) or simply a failed generation (the model produced logically wrong code). CPR distinguishes these by restricting attention to problems where the clean solution was correct. Formally, let $\mathcal{D}$ represent the dataset of programming problems. For each problem $p \in \mathcal{D}$, let C_p denote the original unmodified code and $\tilde{C}_p$ denote the perturbed version containing hidden Unicode lacing. We define Pass$(\cdot)$ as an indicator function that returns 1 if a solution satisfies all unit tests, and SynErr$(\cdot)$ as an indicator function that returns 1 if the code triggers a syntax-level error during interpretation. The CPR score is defined as:

$$\text{CPR} = \frac{\sum_p \mathbb{I}(\text{Pass}(C_p){=}1 \ \wedge \ \text{SynErr}(\tilde{C}_p){=}1)}{\sum_p \mathbb{I}(\text{Pass}(C_p){=}1)} \tag{1}$$

A CPR of 1.0 means every correct solution becomes syntactically unexecutable after intervention; 0.0 means the perturbation has no effect on executability.

4 Results

We report the results in two steps. First, we show why Pass@1 can create a false narrative when applied to soft barriers. Second, we analyze the same outputs using Copy-Paste Resistance (CPR), which more directly measures whether the intervention disrupts naive copy-paste execution.

False Narrative of Pass@1 under Soft Barriers. Pass@1 is the standard metric for code generation, but it is not a sufficient success metric for soft barriers. The goal of our intervention is not to preserve immediate executability after copy-paste. Rather, the goal is to preserve human readability while making naive transfer into an interpreter less frictionless. Thus, a drop in Pass@1 under intervention does not necessarily mean the barrier failed. The reason is that Pass@1 conflates two different cases. In the first case, the model generates a logically correct solution, but the Unicode barrier makes the copied code syntactically invalid. This is a successful barrier. In the second case, the model fails to generate a correct solution under the intervention prompt. This is a model failure. Both reduce Pass@1, but they have opposite meanings. Our results as shown in Figure 2 illustrate this ambiguity. For example, Claude with invisible characters on MBPP has very low Pass@1 under intervention, yet its CPR is 0.997. Interpreted through Pass@1 alone, this looks like failure; interpreted through the handoff lens, it shows that the barrier reliably prevents naive execution of otherwise correct solutions. Conversely, DeepSeek-V3 with reordering on HumanEval maintains relatively high Pass@1, but its CPR is only 0.064, suggesting that the intervention largely fails as a barrier. Therefore, high Pass@1 does not imply barrier success, and low Pass@1 does not imply barrier failure. This motivates CPR as the main metric for evaluating soft barriers: when clean generated code is correct, does the intervention prevent direct copy-paste execution?

Analysis of Soft Barriers under CPR. Table 3 reports CPR across models, benchmarks, and intervention types. CPR varies widely, from near zero to near perfect, showing that soft-barrier effectiveness is highly dependent on the model and perturbation strategy.

Claude is the most affected by soft barriers. On MBPP, invisible characters, homoglyphs, deletions, and reorderings all reach very high CPR (0.997, 0.986, 0.982, and 0.975 respectively). On HumanEval, Claude also shows strong copy-paste resistance for reorderings (0.943), deletions (0.846), and invisible characters (0.766), although homoglyphs are ineffective (0.008). DeepSeek-V3 shows the opposite profile. It is robust to deletions, reorderings, and invisible characters, with CPR mostly below 0.25. However, homoglyphs are effective against it, reaching 0.640 on HumanEval and 0.796 on MBPP. This inversion shows that there is no universal Unicode barrier. That is, a perturbation that works for one model may fail for another. Therefore, for actual deployment, a combination of the Unicode soft-barriers would be more effective. GPT-5.2 and GPT-5.2 Codex show mixed behavior. GPT-5.2 reaches high CPR with invisible characters on HumanEval (0.846), but the same intervention collapses on MBPP (0.006). For codex however, invisible characters achieve 0.773 CPR on HumanEval and 0.771 on MBPP, making it one of the most promising configurations. Overall, CPR reveals the main empirical lesson that soft barriers can work, but they must be calibrated. A useful barrier is one that blocks naive transfer of otherwise correct generated code while preserving enough readability for inspection, reconstruction, or learning.

5 Exploratory Behavioral Pilot

Design. The results in Section 4 measure whether soft barriers affect execution, but they do not show whether users change how they transfer code. To observe whether soft barriers affect user

Table 3: CPR scores. Higher values indicate stronger handoff resistance. Values in bold exceed 0.75.

Dataset	Intervention	Claude	DeepSeek	GPT-5.2	Codex
HumanEval	Deletion	**0.846**	0.040	0.350	0.330
	Homoglyph	0.008	0.640	0.196	0.130
	Invisible chars	**0.766**	0.248	**0.846**	**0.773**
	Reordering	**0.943**	0.064	0.316	0.373
MBPP	Deletion	**0.982**	0.011	0.010	0.090
	Homoglyph	**0.986**	**0.796**	0.000	0.005
	Invisible chars	**0.997**	0.219	0.006	**0.771**
	Reordering	**0.975**	0.042	0.075	0.249

Table 4: Human experiment results. Arrows indicate the direction favorable for learning. No. of retries denotes the average number of submission attempts before completing task.

	Task 1		Task 2	
Metrics | Groups	A	B	A	B
Copied directly ↓	3.83	5.00	3.17	5.00
Modified code ↑	4.00	1.00	3.67	1.00
Understood code ↑	4.50	3.00	4.00	2.00
Felt frustrated ↓	1.67	1.00	1.83	1.00
Helped learn ↑	4.00	2.83	5.00	3.50
No. of retries	3.35	1.33	4.67	1.50

behavior, we ran a small exploratory pilot with 18 participants using GPT-5.2. Participants used a browser-based tool with a programming task, code editor, and embedded AI assistant. After one warm-up task, they completed two Python tasks: range formatting and meeting-conflict detection. Group A used GPT-5.2 with the invisible-character system prompt; Group B used the same model without the intervention. We logged clipboard events and submissions, and collected post-task 1–5 Likert responses. The study used informed consent, disclosed clipboard logging, and was not tied to grades.

Behavioral Observations.

The human pilot suggests that the soft barrier changed how participants engaged with AI-generated code without preventing task completion. Table 4 shows that Group A reported less direct copying and more code modification than Group B (4.00 vs. 1.00 for Task 1; 3.67 vs. 1.00 for Task 2), consistent with a shift toward editing and reconstruction. Group A also reported higher code understanding and greater learning benefit (4.00 vs. 2.83 for Task 1; 5.00 vs. 3.50 for Task 2). Frustration remained low (1.67 and 1.83), suggesting added friction without making the interaction unusable. Group A also required more submission attempts on both tasks, indicating a more iterative inspect–modify–resubmit workflow while still completing the tasks.

Open responses support this interpretation. One participant said the barrier required them to "fully understand what the assistant produced before modifying it," while another said "copy-paste problems" "forced me to read and fully understand the code before executing it." Overall, the pilot provides preliminary evidence that Unicode-based soft barriers can shift AI-code handoff from direct transfer toward active editing and productive friction.

6 Discussion

Broader Implications Beyond Education. Programming education is our main scenario, but the handoff problem generalizes. Campus-managed assistants such as UniGPT-style services could support explanation-first, test-gated, or soft-barrier modes. In professional software engineering, AI-generated code may enter a codebase without review, tests, or provenance tracking. The designs in Table 1 could form a policy layer between an LLM and an IDE, LMS, or repository, with different defaults for beginner exercises, production patches, security-sensitive code, and boilerplate generation.

Threats and Responsible Use. Any motivated user can bypass Unicode barriers by requesting clean output from the model, running a text normalizer, or retyping the code [14]. The claim is not that barriers are secure but that they change the default path for ordinary use. CPR measures a proximate behavioral property, not learning outcomes, and whether changed handoff behavior translates to better comprehension or retention is an open question. Deployment requires transparency. Unicode barriers operating without disclosure are deceptive. Therefore, deployments should be institutionally approved and tested for accessibility, since zero-width characters can affect screen readers.

Research Agenda. A study that logs every model interaction during the session would directly test whether the friction produced is cognitive engagement or workaround behavior. Therefore, a next step would be a more comprehensive human experiment. Beyond that, a systematic CPR measurement across the teaching environment stack (online judges, Jupyter, VS Code, CLI) is needed because Unicode normalization varies across environments, and extending the soft barrier techniques would test whether the approach depends on language-specific identifier rules.

7 Conclusion

AI coding assistants are evaluated on how well they generate code, but the moment that code enters a running program is equally consequential and currently unmanaged. This paper introduces the AI code handoff problem, proposes soft barriers as a class of output-level mechanisms for shaping the handoff boundary, and uses Unicode perturbations as a first technical probe. Copy-Paste Resistance provides a metric that isolates handoff friction from generation quality. Our automated results and exploratory pilot suggest that output-level barriers can shape the AI-code handoff, motivating broader research on handoff-aware programming assistants. The longer-term goal is handoff-aware AI coding assistants that treat the boundary between generated text and executable software as a design surface worth optimizing.

8 Acknowledgment

This research was funded in whole, or in part, by the Luxembourg National Research Fund (FNR), grant reference C25/IS/19639771/BRIDGE and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Project NATURAL-Grant agreement N° 949014).

Data Availability Statement

All data, code, and scripts required to reproduce the experiments in this paper are available in our anonymous repository: https://zenodo.org/records/21786962.

References

[1] Anthropic. 2025. Claude 4.5 Sonnet. https://www.anthropic.com/news/claude-sonnet-4-5. Accessed: March 14, 2025.

[2] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732* (2021). doi:10.48550/arXiv.2108.07732

[3] Mary Burns, Rebecca Winthrop, Natasha Luther, Emma Venetis, and Rida Karim. 2026. *A New Direction for Students in an AI World: Prosper, Prepare, Protect*. Report. Center for Universal Education at Brookings. https://www.brookings.edu/centers/center-for-universal-education/

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374* (2021). doi:10.48550/arXiv.2107.03374

[5] Albérick Euraste Djiré, Abdoul Kader Kaboré, Iyiola E Olatunji, Earl T Barr, Jacques Klein, and Tegawendé F Bissyandé. 2025. Memorization or interpolation? detecting llm memorization through input perturbation analysis. *arXiv preprint arXiv:2505.03019* (2025). doi:10.48550/arXiv.2505.03019

[6] Alberick Euraste Djire, Iyiola E. Olatunji, Melissa Tessa, Earl T. Barr, Jacques Klein, and Tegawendé F. Bissyandé. 2026. Evaluating Inference-Time Defenses Against Package Hallucination in LLM-Generated Code. In *Proceedings of the 41st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. doi:10.1145/3832783.3837555

[7] Djiré Albérick Euraste, Kaboré Abdoul Kader, Jordan Samhi, Earl T Barr, Jacques Klein, and Tegawendé F Bissyandé. 2026. Learned or Memorized? Quantifying Memorization Advantage in Code LLMs. *arXiv preprint arXiv:2604.13997* (2026). doi:10.48550/arXiv.2604.13997

[8] Dren Fazlija, Iyiola E Olatunji, Daniel Kudenko, and Sandipan Sikdar. 2026. Towards Sensitivity-Aware Language Models. *arXiv preprint arXiv:2601.20901* (2026). doi:10.48550/arXiv.2601.20901

[9] James Finnie-Ansley, Paul Denny, Andrew Luxton-Reilly, Eddie Antonio Santos, James Prather, and Brett A Becker. 2023. My AI wants to know if this will be on the exam: Testing openai's codex on cs2 programming exercises. In *Proceedings of the 25th Australasian Computing Education Conference*. 97–104. doi:10.1145/3576123.3576134

[10] Michael Gerlich. 2025. AI tools in society: Impacts on cognitive offloading and the future of critical thinking. *Societies* 15, 1 (2025), 6. doi:10.3390/soc15010006

[11] Nataliya Kosmyna, Eugene Hauptmann, Ye Tong Yuan, Jessica Situ, Xian-Hao Liao, Ashly Vivian Beresnitzky, Iris Braunstein, and Pattie Maes. 2025. Your brain on ChatGPT: Accumulation of cognitive debt when using an AI assistant for essay writing task. *arXiv preprint arXiv:2506.08872* 4 (2025). doi:10.48550/arXiv.2506.08872

[12] Hao-Ping Lee, Advait Sarkar, Lev Tankelevitch, Ian Drosos, Sean Rintel, Richard Banks, and Nicholas Wilson. 2025. The impact of generative AI on critical thinking: Self-reported reductions in cognitive effort and confidence effects from a survey of knowledge workers. In *Proceedings of the 2025 CHI conference on human factors in computing systems*. 1–22. doi:10.1145/3706598.3713778

[13] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437* (2024). doi:10.48550/arXiv.2412.19437

[14] Iyiola E Olatunji, Franziska Boenisch, Jing Xu, and Adam Dziedzic. 2025. Adversarial attacks and defenses on graph-aware large language models (llms). *arXiv preprint arXiv:2508.04894* (2025). doi:10.48550/arXiv.2508.04894

[15] OpenAI. 2025. GPT-5.2. https://openai.com/index/introducing-gpt-5-2/. Accessed: December 31, 2025.

[16] Wei Hung Pan, Ming Jie Chok, Jonathan Leong Shan Wong, Yung Xin Shin, Yeong Shian Poon, Zhou Yang, Chun Yong Chong, David Lo, and Mei Kuan Lim. 2024. Assessing AI detectors in identifying AI-generated code: Implications for education. In *Proceedings of the 46th international conference on software engineering: software engineering education and training*. 1–11. doi:10.1145/3639474.3640068

[17] Christian Rahe and Walid Maalej. 2025. How Do Programming Students Use Generative AI? *Proceedings of the ACM on Software Engineering* 2, FSE (2025), 978–1000. doi:10.1145/3715762

[18] Prateek Kumar Rajput, Yewei Song, Abdoul Aziz Bonkoungou, Iyiola E Olatunji, Abdoul Kader Kabore, Jacques Klein, and Tegawendé F Bissyandé. 2026. Correctness isn't Efficiency: Runtime Memory Divergence in LLM-Generated Code. In *Proceedings of the IEEE/ACM 48th International Conference on Software Engineering: Software Engineering in Practice*. 693–703. doi:10.1145/3786583.3786908

[19] Reuters. 2025. OpenAI targets higher education in the U.S. with ChatGPT rollout at California State University. https://www.reuters.com/technology/openai-targets-higher-education-us-with-chatgpt-rollout-california-state-2025-02-04/. Accessed: 2026-05-13.

[20] Evan F Risko and Sam J Gilbert. 2016. Cognitive offloading. *Trends in cognitive sciences* 20, 9 (2016), 676–688. doi:10.1016/j.tics.2016.07.002

[21] Anjali Singh, Karan Taneja, Zhitong Guan, and Avijit Ghosh. 2025. Protecting human cognition in the age of AI. *arXiv preprint arXiv:2502.12447* (2025). doi:10.48550/arXiv.2502.12447

[22] Melissa Tessa, Iyiola E Olatunji, Jacques Klein, and Tegawendé F Bissyandé. 2026. Position: The Iceberg of Pitfalls in LLM-Based Secure Code Generation. In *Proceedings of the AAAI Symposium Series*, Vol. 9. 162–165. doi:10.1609/aaaiss.v9i1.42920

[23] Melissa Tessa, Iyiola E Olatunji, Aicha War, Jacques Klein, and Tegawendé F Bissyandé. 2026. How Secure is Secure Code Generation? Adversarial Prompts Put LLM Defenses to the Test. *arXiv preprint arXiv:2601.07084* (2026). doi:10.48550/arXiv.2601.07084

[24] Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F Bissyandé. 2023. Is ChatGPT the ultimate programming assistant–how far is it? *arXiv preprint arXiv:2304.11938* (2023). doi:10.48550/arXiv.2304.11938

[25] University of Graz, IDea_Lab. 2025. uniGPT and studiGPT: The Data Privacy-Friendly Chatbots of the University of Graz. https://idea-lab.uni-graz.at/en/learn-apply-connect/unigpt-and-studigpt/. Accessed: 2026-05-13.

[26] Yale University. 2026. Yale's AI Tools and Resources. https://ai.yale.edu/yales-ai-tools-and-resources. Accessed: 2026-05-13.

Received 2026-05-13; accepted 2026-07-02