

Strategic Missingness in AI Help-Seeking Traces for Programming Education

Hanjing Shi¹, Dominic DiFranzo¹

¹*Lehigh University, Bethlehem, PA, United States*

Abstract

Generative AI has made programming homework harder to observe. A student can debug with a chatbot, revise generated code, and submit a final program that hides the help-seeking path. Detection and penalties address some integrity concerns, but they also create a data problem: students who expect sanction can move help seeking outside course tools. We name this problem strategic missingness, missing process evidence produced by policy, interface design, incentives, and trust. We argue for governed AI help-seeking traces as computing-education data infrastructure: linked records of code attempts, run failures, help requests, support states, student reactions, verification attempts, and follow-up practice. These traces make episodes of programming work and support visible for interpretation. The contribution is a design and research agenda for studying AI help seeking with process evidence governed for teaching, aggregate interpretation, and protection from automatic punishment.

Keywords

programming education, generative AI, help-seeking, process traces, learning analytics, academic integrity, strategic missingness

1. Introduction

Generative AI has changed what it means to observe programming homework. A student can write code, ask a model to explain an error, request a hint, paste in a generated solution, revise it, and submit final code that gives almost no trace of that path. The old record of work, final submission plus a few timestamps, is now too thin. It cannot distinguish independent debugging from a conceptual nudge, guided decomposition, or solution-level assistance.

Some course policies begin from uncertainty: detect AI use, then sanction it. That approach can be necessary in assessment settings with explicit rules about unauthorized assistance. The trouble starts when detection becomes the main design posture. Students who expect punishment have a practical reason to move help seeking away from course tools, to use private chatbots, or to paste results back into the assignment environment after the meaningful work has happened elsewhere. The course may still receive a submission. What disappears is the process.

For programming education, useful evidence lies in the movement among coding, debugging, help seeking, response evaluation, verification, and review. Strategic missingness names the loss of that evidence when the learning environment gives students a reason to hide the process. It differs from ordinary logging failure because course rules and interface choices can produce the missing data.

Unofficial help is not new in computing education. Students have long used friends, forums, office hours, tutors, search engines, and copied examples, and EDM researchers have always interpreted traces with that incompleteness in mind. Generative AI makes the measurement problem sharper because help can be immediate, private, executable, and hard to see in the final submission. A large share of interactive problem solving can move into an unobserved channel unless the course offers a trustworthy alternative.

Computing education therefore needs trustworthy infrastructures for AI help-seeking traces. Such infrastructures can make legitimate help available inside course contexts, represent AI help as a learning-process signal, begin instructor analytics from aggregate teaching needs, and govern trace

CSEDM 2026: 10th Educational Data Mining in Computer Science Education Workshop, June 28, 2026, Seoul, South Korea

✉ hasa23@lehigh.edu (H. Shi); djd219@lehigh.edu (D. DiFranzo)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

access carefully. The research object becomes the conditions under which help remains visible enough to interpret.

Trust is not a notice box. Students may still avoid a transparent system if they think the trace can later be used for grading, individual ranking, or misconduct investigation. A trace system must therefore align incentives: authorized help should be useful, the policy should state what is not used for punishment, and the default instructor view should support teaching rather than suspicion. Without those social conditions, even a technically careful logging system can create missingness.

AI-era programming data are partly an artifact of the environment that records them. A course that centers detection of final-product irregularities and a course that offers logged, bounded, course-grounded help may produce different datasets from the same underlying student needs. Both datasets can look orderly. The difference is whether the path remains visible: where students got stuck, what help they asked for, how they evaluated it, and when they returned to work.

The missingness is unlikely to be random. Missing-data theory offers a useful reminder: inference depends on the process that produced the observed record [1]. In this paper, missingness is a measurement frame for trace infrastructure: it asks how policy and design shape what becomes observable before any formal missing-data model can be trusted. Academic help seeking relates to students' broader learning strategies and achievement behavior [2]. Help avoidance can also be shaped by perceived threat, competence, and classroom norms [3]. The students most likely to hide AI use may therefore be those who are most anxious about policy, least confident in their own work, most dependent on outside help, or most aware that a detector can be wrong. Those are precisely the cases where process evidence would help instructors interpret need. A dataset shaped by fear can look cleaner while becoming less valid. Fewer visible AI events may mean less observable help, not less AI help.

This makes trace completeness a design outcome. In ordinary process mining, missing events are often treated as a technical limitation. In AI-supported homework, missing events can also be produced by course language, interface defaults, grade incentives, and students' trust in the institution. For a field that studies programming learning through data, those conditions belong in the data-generating process.

The position leads to commitments that appear throughout the paper: preserve situated help, read silence as informative, keep support states non-punitive, and start analytics from teaching questions. These commitments make the argument operational by specifying what must be captured, what cannot be inferred from silence, and where governance has to be designed into the data system.

2. What AI changes in unofficial help

Programming courses already live with partial observability. A trace rarely captures a roommate's hint, a copied Stack Overflow pattern, a lab assistant's explanation, or a student's private reasoning. Prior EDM practice has handled this by treating traces as evidence of interaction with a system, not as a complete record of learning. Generative AI makes the gap larger and more structured. It can join the work session at the point of error, explain the bug, rewrite the code, and suggest tests. If that work happens outside the course environment, the missing events are no longer peripheral. They may be the core help-seeking episode.

By enforcement-first policies, we mean systems or course routines whose primary purpose is to identify unauthorized AI use and attach consequences. The framing is understandable. It draws a boundary between acceptable and unacceptable work, and it matches the familiar academic-integrity logic of detection, evidence, and penalty. AI use in programming homework, however, often appears throughout the process of solving a problem, long before final submission.

The limits of detection are clearest in text settings, where model-output detection is vulnerable to paraphrasing and attacks [4]. Tool performance also varies across systems [5]. Detectors can misclassify non-native English writers [6]. Programming differs from prose, but the lesson carries over: a detector is a weak proxy for learning process. Even a correct AI-use flag says little about whether a student asked for a syntax explanation, used a hint to locate an error, requested a full solution, or verified the

answer afterward.

Surveillance also changes behavior. Dawson's study of institutional surveillance technologies found that students reported changes in browsing behavior, discussion topics, and writing style when they perceived online learning activities as surveilled [7]. Academic-integrity work likewise points to institutional culture alongside the threat of punishment [8]. Recent survey work on generative AI in higher education finds that students navigate mixed messages, ambiguous use cases, and false accusations when deciding how to use AI tools [9]. In AI-supported programming, enforcement may push AI use elsewhere while leaving the amount of use unresolved. Once help seeking leaves the platform, the remaining trace is a partial record shaped by fear, convenience, and policy interpretation.

For learning analytics, the loss is pedagogical as well as statistical. Programming education has long benefited from process evidence and has an established data-mining and learning-analytics tradition around programming traces [10]. Compilation histories help researchers reconstruct novice trajectories from more than the final program [11]. Programming-process data formats make code snapshots and events easier to export and compare [12]. AI assistance makes that trajectory more complex. If the environment logs only final code or an AI-use suspicion score, it compresses a rich help-seeking sequence into a compliance variable. Learning analytics needs context and purpose [13]. It also needs privacy principles for collection and interpretation [14]. Trace-data reviews show that ethical risks extend across the data lifecycle [15].

Assessment rules and process interpretation need separate interfaces. A system designed only for enforcement makes students legible as possible violators. A system designed for governed help seeking makes student work legible as a sequence that instructors can interpret.

High-stakes assessment still needs academic-integrity rules, restricted tool-use conditions, and ways to investigate clear violations of stated policy. The focus here is ordinary programming homework treated as learning evidence. The measurement problem appears when enforcement becomes the default infrastructure for understanding that work. Evidence collected for sanction answers a different question from evidence collected for teaching. A healthy course can keep those questions separate: what counts as authorized help, what process evidence supports instruction, and what exceptional evidence belongs in an integrity case.

3. Help seeking as the unit of analysis

Help seeking is already a learning process. Classic help-design work distinguishes productive help seeking from unproductive patterns such as avoiding help when needed or seeking answers without engaging the underlying task [16]. Metacognitive tutoring work models help seeking as a process that can itself be supported [17]. Generative AI moves that distinction into a new interface. It changes how help is requested and how much support a student can receive before another human sees the struggle.

Programming is a useful setting for this reframing because its work is already traceable. A session may include edits, run events, compiler messages, runtime errors, pasted code, test failures, and revisions. The question is how to connect those traces with AI help. A useful help-seeking trace links the code state before help, the student's description of being stuck, the kind of help requested, the support state, the model response, the student's reaction, later execution, and any follow-up practice.

A help-seeking trace is a linked event record. The unit of analysis is an episode in which a student encounters a programming problem, seeks or receives support, reacts to that support, and either returns to the task or moves elsewhere. That episode can include human help, AI help, platform feedback, and silence where help likely happened outside the system. The design task is to represent help as situated action rather than as an after-the-fact AI-use label.

A linked view lets researchers ask different questions. Did repeated run failures precede escalation to stronger support? Did students verify AI feedback by running edited code later? Were solution-level requests followed by conceptual review, or did they end the session? Did aggregate traces reveal a class-wide misconception before office hours? A detector score leaves these questions out.

Recent AI systems in computing education already move toward course-grounded support and

Table 1

Strategic missingness as a measurement problem.

Design condition	Missing process evidence	Analytic distortion	Trace response
AI use is framed mainly as a detectable violation	Help requests move to private tools before code returns to the course system	Low visible AI use can be mistaken for low AI use	Measure trace completeness and ask where help seeking occurred
AI logs are stored apart from code work	The code state before and after help is hard to recover	Chat quality is interpreted without task trajectory	Link request, response, snapshot, reaction, and later run events
Support level is treated as suspicion	Students avoid weaker help until they are already stuck or use outside help instead	Escalation rates reflect policy fear as well as need	Interpret support state as a process signal, not a score
Dashboards open on individuals	Instructor attention begins with who looks risky	Class-wide misconceptions remain secondary	Start with anonymous aggregate concepts, errors, and support patterns

instructor visibility. CS50's AI tools show how generative AI can be grounded in course context [18]. HiTA centers educators in the instructional loop through retrieval-augmented course support [19]. Instructor-in-the-loop systems add escalation paths when AI help is insufficient [20]. AI code generators can support novices while changing the learning task [21]. Prompt-based problem solving with Copilot shows how natural-language interaction can enter CS1 work [22]. A computing-education working group maps broader opportunities and risks from generative AI [23]. Visualization systems such as Editrail show that instructors want to understand student-AI interaction during coding, before the record collapses into after-the-fact inference [24]. These systems make governed process data a practical design object. The frame here adds a measurement question to that line of work: why help-seeking data disappear, and how code, help, reaction, verification, and aggregate teaching signals can be recorded together.

4. Design commitments for governed traces

When trace infrastructure becomes a quiet monitoring layer, it inherits the same trust problem as detection. Process traces are sensitive educational data, especially when they include code snapshots, stuck descriptions, model responses, and timestamps. A governed trace system needs a design stance before it needs a dashboard. Table 2 summarizes commitments that follow from the position.

These commitments recast tiered AI support. A support tier records how much help a student requested and received at a point in time. It has no standing as an automatic score. A student who moves from independent work to stronger help may be struggling productively, using the available supports as intended, or seeking a shortcut. The trace only begins the interpretation. It does not complete it.

The aggregate-first commitment matters because it changes how the data first become visible. Learning analytics ethics research warns that educational data can be misread or over-classify learners [13]. Privacy principles warn that analytics can become surveillance when institutional access is treated as self-justifying [14]. Trace-data reviews show the same concern across digital education datasets [15]. Data-protection work asks analytics designs to specify purpose, access, and proportionality, with collection carrying a specific justification [25]. Instructors often need class-level signals first: which concept is breaking down, which assignment causes repeated escalation, which error pattern appears across many students. Individual traces may still matter, but the entry point is a teaching question, not a default suspicion queue.

Aggregate-first does not mean aggregate-only. Class-level signals are computed from individual

Table 2

Design commitments for AI help-seeking trace infrastructure.

Commitment	Trace implication	Risk reduced
Make legitimate help usable	Provide course-grounded help inside the assignment workflow and record help requests alongside code work.	Fewer incentives to move learning-relevant help seeking off platform.
Represent support as process state	Track independent work, hints, guided scaffolding, and stronger rescue support as states in a session timeline.	Less pressure to treat support level as a grade or misconduct score.
Link code, help, reaction, and verification	Use assignment, session, snapshot, request, response, reaction, and run identifiers to connect events.	Chat logs and code histories do not remain separate fragments.
Start analytics from aggregates	Show class-level concepts, repeated errors, support distributions, and assignment pain points before individual drill-down.	Instructor attention begins with teaching needs before individual suspicion.
Treat off-platform AI as missingness	Ask about and model outside AI use as a threat to trace completeness.	Missing data stays tied to design and policy.
Govern access and retention	Give notice about logging, viewers, retention, deletion, de-identification, and limits of pseudonymization.	Trace collection is less likely to become opaque surveillance.

event records, but the system can delay or restrict individual drill-down until there is a teaching or support reason to inspect it. For example, an instructor might first see that many students ask for rescue-level help on loop bounds, then open representative de-identified sessions, and only later inspect a named student’s trace for advising. This order matters because the same underlying data can support instruction or surveillance depending on the default view.

5. An illustrative infrastructure pattern

A programming homework environment can instantiate this position without presenting itself as an unrestricted chatbot. The basic pattern is an integrated coding workspace with run feedback, course-material grounding, structured help requests, support states, student reactions, follow-up practice, and instructor summaries. The substantive move is the event model that connects AI help to the programming process.

Consider a student working on a list-indexing problem. The student runs code several times and receives the same runtime error. They request a low-support hint and write, “I think my loop goes one item too far.” The system records the code snapshot, the error category, the help request, the support state, the course-material excerpt retrieved for the response, and the model response. The student marks the response as still unclear, asks for stronger guidance, edits the loop bound, runs the code again, and later opens a short practice item on indexing. A linked trace can represent this as a sequence of learning events. A detector would collapse the same episode into an after-the-fact judgment about AI use.

A minimal exported trace for the episode might contain four linked records: a run event with session id, snapshot id, error category, and outcome; a `help_request` event with the same snapshot id, stuck text, support state, and retrieved course excerpt; a `student_reaction` event linked to the response id; and a `practice_opened` event linked to the concept tag. The important feature is the join across records. Each event keeps enough identifiers to connect help with the prior code state and later verification.

Concretely, a system designer can start with a small schema: `session_id`, `assignment_id`, `snapshot_id`, `event_type`, `support_state`, `concept_tag`, `student_reaction`, `verification_event`, `viewer_role`, and `retention_class`. These fields make the com-

mitments testable. They show whether help is linked to code, whether support state is treated as process information, whether instructor views begin from aggregates, and whether access and retention are explicit.

For research, the linked trace supports sequence-level analysis. Researchers can examine whether repeated errors predict escalation, whether students evaluate AI feedback before editing, whether later runs verify an AI suggestion, and whether targeted practice follows specific error categories. For teaching, the same data can be aggregated into signals such as common stuck points, concepts needing review, assignments generating rescue-level support, and patterns of unresolved confusion.

The pattern also handles students who use tools outside the platform. Off-platform use is not a moral residue that invalidates all analysis. It is a missing-data mechanism. Surveys, interviews, and lightweight self-report can help estimate when traces are incomplete, but they should not be treated as exact ground truth. Self-report will be shaped by memory, policy fear, perceived anonymity, and whether students believe the answer can affect grades. A better design treats self-report as one weak signal, triangulates it with aggregate platform patterns, and reports the uncertainty. Perfect visibility is the wrong target and carries its own ethical risks. A better target is enough trustworthy visibility for instructors to teach from process evidence.

6. Research agenda

The first empirical question is **trace completeness**. When course-grounded AI help is available inside the assignment environment, what kinds of help seeking remain on platform, and what still moves outside it? Direct evidence about policy-driven displacement in programming homework is still emerging, so displacement should be treated as a measurement target rather than a settled fact. The answer will depend on policy, trust, task type, perceived usefulness, and the cost of asking for help.

Once some of that trace remains visible, **sequence validity** asks whether the event links support meaningful interpretation. Researchers need to test whether linked code snapshots, run failures, help requests, reactions, later executions, and practice records can distinguish different help-seeking patterns. The point is to understand when AI support participates in debugging, when it replaces problem solving, and when it leads to review.

Those links are useful only if instructors can read them carefully, so **instructor interpretability** needs its own evaluation. Dashboards often present metrics before they establish what an instructor can safely infer from them. Aggregate support distributions, common concepts, and repeated errors may be useful for lecture planning and office hours. Individual drill-down may be useful for advising or targeted support. These uses need separate evaluation because they carry different risks.

The same logic puts **governance** inside the measurement model rather than after the dashboard is built. Trace infrastructures are socio-technical arrangements, not neutral pipes. Students need to know what is logged, who can view it, how long it is retained, and what role it plays in grading or support. Researchers also need to test how institutional language affects help-seeking behavior. A course that says “AI use will be detected and punished” may produce a different dataset from one that says “course-grounded AI help is available, logged, and interpreted as a support process.”

The agenda also needs **interoperability**, because trace designs will otherwise stay tied to single tools. Programming-process datasets already have useful formats and conventions [12]. AI-help traces can build on that work and avoid isolated chatbot logs. A useful schema would represent code states, run events, help requests, support states, reactions, and follow-up activities in ways that can be exported, inspected, and compared across tools.

These questions require more than log analysis. Studies need to report the course AI policy, the wording students saw, the stakes attached to platform traces, and the alternatives students had outside the system. Without that context, a low rate of visible AI help could mean compliance, distrust, poor usability, or successful displacement. Policy language and interface defaults are part of the evidence behind AI-era programming datasets.

A cumulative research program will need dataset context as well as event records. At minimum,

studies of AI help-seeking traces should report metadata about institutional rules, interface affordances, interpretation, and missingness. Institutional context covers which uses of AI were permitted, discouraged, required, or punishable. Interface context describes what help was available in the course system, how easy it was to access, and whether students could see how requests were logged. Interpretive context identifies who could view individual traces, whether dashboards began from aggregate patterns, and whether support states affected grades. Missingness context reports what off-platform tools students were likely to use, how outside use was measured, and what the study cannot observe. Without this layer, two datasets with similar event fields may describe very different classrooms.

7. Conclusion

Generative AI has made final-code records even thinner than they used to be. A course may receive a correct submission and still know little about the path that produced it. Treating AI use mainly as a violation to detect makes that path harder to see. Treating AI help seeking as a governed process trace gives the field a better object: incomplete, sensitive, and interpretable only in context, but far richer than final code or suspicion scores. When course policy teaches students to hide help seeking, computing education loses the evidence it needs. The next task is to design environments where help can remain visible enough to teach from and carefully governed enough to avoid automatic judgment.

Declaration on Generative AI

During the preparation of this work, the authors used large language model tools for grammar checking and paraphrasing author-written content. After using these tools, the authors reviewed and edited the manuscript and take full responsibility for the publication's content.

References

- [1] D. B. Rubin, Inference and missing data, *Biometrika* 63 (1976) 581–592. doi:10.1093/biomet/63.3.581.
- [2] S. A. Karabenick, J. R. Knapp, Relationship of academic help seeking to the use of learning strategies and other instrumental achievement behavior in college students, *Journal of Educational Psychology* 83 (1991) 221–230. doi:10.1037/0022-0663.83.2.221.
- [3] A. M. Ryan, P. R. Pintrich, C. Midgley, Avoiding seeking help in the classroom: Who and why?, *Educational Psychology Review* 13 (2001) 93–114. doi:10.1023/A:1009013420053.
- [4] V. S. Sadasivan, A. Kumar, S. Balasubramanian, W. Wang, S. Feizi, Can AI-generated text be reliably detected?, 2023. URL: <https://arxiv.org/abs/2303.11156>. arXiv:2303.11156.
- [5] D. Weber-Wulff, A. Anohina-Naumeca, S. Bjelobaba, T. Foltýnek, J. Guerrero-Dib, O. Popoola, P. Šigut, L. Waddington, Testing of detection tools for AI-generated text, *International Journal for Educational Integrity* 19 (2023). doi:10.1007/s40979-023-00146-z.
- [6] W. Liang, M. Yuksekgonul, Y. Mao, E. Wu, J. Zou, GPT detectors are biased against non-native English writers, *Patterns* 4 (2023) 100779. doi:10.1016/j.patter.2023.100779.
- [7] S. Dawson, The impact of institutional surveillance technologies on student behaviour, *Surveillance & Society* 4 (2002) 69–84. doi:10.24908/ss.v4i1/2.3455.
- [8] D. L. McCabe, L. K. Trevino, K. D. Butterfield, Dishonesty in academic environments, *The Journal of Higher Education* 72 (2001) 29–45. doi:10.1080/00221546.2001.11778863.
- [9] B. W. Stone, Generative AI in higher education: Uncertain students, ambiguous use cases, and mercenary perspectives, *Teaching of Psychology* 52 (2025) 347–356. doi:10.1177/00986283241305398.
- [10] P. Ihantola, A. Vihavainen, A. Ahadi, M. Butler, J. Börstler, S. H. Edwards, E. Isohanni, A. Korhonen, A. Petersen, K. Rivers, M. À. Rubio, J. Sheard, B. Skupas, J. Spacco, C. Szabo, D. Toll, Educational data mining and learning analytics in programming, in: *Proceedings of the 2015 ITiCSE on Working Group Reports*, ACM, 2015, pp. 41–63. doi:10.1145/2858796.2858798.

- [11] M. C. Jadud, Methods and tools for exploring novice compilation behaviour, in: *Proceedings of the Second International Workshop on Computing Education Research*, ACM, 2006, pp. 73–84. doi:10.1145/1151588.1151600.
- [12] T. W. Price, D. Hovemeyer, K. Rivers, G. Gao, A. C. Bart, A. M. Kazerouni, B. A. Becker, A. Petersen, L. Gusukuma, S. H. Edwards, D. Babcock, Progsnap2: A flexible format for programming process data, in: *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ACM, 2020, pp. 356–362. doi:10.1145/3341525.3387373.
- [13] S. Slade, P. Prinsloo, Learning analytics: Ethical issues and dilemmas, *American Behavioral Scientist* 57 (2013) 1510–1529. doi:10.1177/0002764213479366.
- [14] A. Pardo, G. Siemens, Ethical and privacy principles for learning analytics, *British Journal of Educational Technology* 45 (2014) 438–450. doi:10.1111/BJET.12152.
- [15] L. Hakimi, R. Eynon, V. A. Murphy, The ethics of using digital trace data in education: A thematic review of the research landscape, *Review of Educational Research* 91 (2021) 671–717. doi:10.3102/00346543211020116.
- [16] V. Aleven, E. Stahl, S. Schworm, F. Fischer, R. Wallace, Help seeking and help design in interactive learning environments, *Review of Educational Research* 73 (2003) 277–320. doi:10.3102/00346543073003277.
- [17] V. Aleven, B. McLaren, I. Roll, K. Koedinger, Toward meta-cognitive tutoring: A model of help seeking with a cognitive tutor, *International Journal of Artificial Intelligence in Education* 16 (2006) 101–128. doi:10.3233/IRG-2006-16(2)02.
- [18] R. Liu, C. Zenke, C. Liu, A. Holmes, P. Thornton, D. J. Malan, Teaching CS50 with AI: Leveraging generative artificial intelligence in computer science education, in: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, ACM, 2024, pp. 750–756. doi:10.1145/3626252.3630938.
- [19] C. Liu, L. Hoang, A. Stolman, B. Wu, HiTA: A RAG-based educational platform that centers educators in the instructional loop, in: *Artificial Intelligence in Education*, Springer, 2024, pp. 405–412. doi:10.1007/978-3-031-64299-9_37.
- [20] T. Phung, H. Choi, M. Wu, C. Brooks, S. Gulwani, A. Singla, Closing the loop: An instructor-in-the-loop AI assistance system for supporting student help-seeking in programming education, in: *Proceedings of the 57th ACM Technical Symposium on Computer Science Education V.1*, ACM, 2026, pp. 852–858. doi:10.1145/3770762.3772612.
- [21] M. Kazemitabaar, J. Chow, C. K. T. Ma, B. J. Ericson, D. Weintrop, T. Grossman, Studying the effect of AI code generators on supporting novice learners in introductory programming, in: *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, ACM, 2023, pp. 1–23. doi:10.1145/3544548.3580919.
- [22] P. Denny, V. Kumar, N. Giacaman, Conversing with copilot: Exploring prompt engineering for solving CS1 problems using natural language, in: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, ACM, 2023, pp. 1136–1142. doi:10.1145/3545945.3569823.
- [23] J. Prather, P. Denny, J. Leinonen, B. A. Becker, I. Albluwi, M. Craig, H. Keuning, N. Kiesler, T. Kohn, A. Luxton-Reilly, S. MacNeil, A. Petersen, R. Pettit, B. N. Reeves, J. Savelka, The robots are here: Navigating the generative AI revolution in computing education, in: *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education*, ACM, 2023, pp. 108–159. doi:10.1145/3623762.3633499.
- [24] A. G. Zhang, Y.-R. Jhou, Y. Yang, S. Rao, M. Arab, Y. Chen, S. Oney, Editrail: Understanding AI usage by visualizing student-AI interaction in code, 2026. URL: <https://arxiv.org/abs/2601.20085>. doi:10.48550/arXiv.2601.20085. arXiv:2601.20085.
- [25] A. N. Cormack, A data protection framework for learning analytics, *Journal of Learning Analytics* 3 (2016) 91–106. doi:10.18608/jla.2016.31.6.