Journal of Innovative Learning
Institute for Innovative Learning, Mahidol University
Volume 2 No. 2, page 1 - 11

Research Article

AI Audit Lab: A Phase 1 Design-and-Development Framework and Evaluation Protocol for Auditing AI-Assisted Machine-Learning Workflows

Phirawit JITNARONG[a*]
[a]*Department of Computer Science, Faculty of Informatics, Burapha University, Thailand*
*Corresponding author: nxme176@gmail.com

Abstract: Generative AI (GenAI) coding assistants let learners produce working machine-learning notebooks quickly. The code can contain easily missed methodological errors, such as data leakage, invalid validation splits, and unwarranted causal claims. This paper reports Phase 1 of a design and development research (DDR) study of the AI Audit Lab, an instructional framework for upper-level undergraduate and early postgraduate data-science learners. It treats GenAI output as an artifact to be audited rather than trusted. Development followed analysis, design, and development phases, all conducted by the author, and each material was executed or reviewed and revised until it worked as specified. Learners follow a five-step cycle (inspect, predict, test, repair, explain) on notebooks seeded with GenAI-style errors. A Socratic tutor policy withholds case-specific guidance until the learner commits to a hypothesis. The framework specifies eight error scenarios in three validity-failure categories, with feedback tiered by evidentiary strength. A reproducible computational demonstration shows that two leakage instances can have very different numerical consequences. A split-plot Phase 2 protocol is also specified, with seven prespecified threats to validity. A Phase 1 package for two scenarios is provided as supplementary material. The framework has not yet undergone expert review, usability testing, or learner try-out, and no claim of pedagogical effectiveness is made.

Keywords: generative AI, data leakage, reproducibility, machine learning education, productive failure, design and development research

1. Introduction

Generative AI (GenAI) coding assistants can shorten the path from a data-analysis question to a runnable answer. A learner can describe a prediction task in one sentence and receive a complete notebook with preprocessing, a fitted machine learning (ML) model, a metric, and often a confident recommendation. This widens access, but a learner can now produce a working notebook without being able to judge whether its conclusions hold.

Kapoor and Narayanan (2023) found data leakage, in which information from outside the legitimate training set influences training or evaluation, in at least 294 published studies by trained researchers. In some cases, leakage inflated accuracy enough that the claim did not survive correction. GenAI-authored code adds a complication: it is fluent, explains its steps, and can state that the model is ready for use, so a learner with limited domain grounding has little reason to question it.

Gerlich (2025) reported a negative correlation between frequent AI-tool use and critical thinking, mediated by cognitive offloading, in a cross-sectional mixed-methods study of 666 participants. The study was not specific to data science and does not show causation. It is treated here as indirect motivation for a design that tries to counter offloading.

Two responses are commonly discussed: restricting GenAI use in coursework, or teaching learners to interrogate its output directly (Lau & Guo, 2023). Restriction may be hard to enforce, and it does not build the skill of judging AI output. This paper develops the second approach and treats GenAI output as a claim to be verified.

This paper presents the AI Audit Lab, an instructional framework with four components: a five-step audit cycle, eight GenAI-style error scenarios in three validity-failure categories, feedback tiered by evidentiary strength, and a Socratic tutor protocol under which learners commit to a diagnosis before case-specific help. The framework is grounded in productive failure (Kapur, 2008) and in metacognitive monitoring and control (Nelson & Narens, 1990). No comparative human-subjects evaluation has been conducted.

The intended learners are upper-level undergraduate and early postgraduate students in data science, computer science, or related quantitative programs. They have completed introductory Python and supervised machine-learning courses and already use GenAI coding assistants. Novice programmers are not the target because their main difficulty is writing code rather than judging it. This profile is a design assumption not yet checked with a real cohort. Three design questions guided Phase 1. (DQ1) What design principles and components can lead such learners to audit GenAI-style ML workflows rather than accept them (Sections 3.2–3.6)? (DQ2) Can a reproducible demonstration illustrate, on one synthetic dataset, that two leakage instances differ in numerical consequence (Section 4)? (DQ3) How should Phase 2 evaluate effectiveness while controlling threats to validity (Section 3.8; Appendix A)?

2. Related Work

2.1 Data leakage as a technical problem

Leakage is a family of related failure modes. Kapoor and Narayanan (2023) propose a taxonomy of eight leakage types. Sasse et al. (2025) catalogue leakage scenarios across the supervised-learning pipeline. Some of these scenarios inflate performance dramatically, while others barely move the metric and still invalidate the workflow. Some GenAI notebook errors, such as metric misuse and causal overclaiming, are validity failures but not leakage. The taxonomy therefore extends beyond leakage (Section 3.4).

2.2 Automated detection tooling

Yang et al. (2022) detect overlap, multi-test, and preprocessing leakage in notebook code by static analysis. Truong et al. (2025) released LeakageDetector 2.0, a Visual Studio Code (VS Code) extension with AI-assisted quick fixes. Tier 1 checks (Section 3.6) could wrap such tools and add assertions for other leakage types. For preprocessing leakage, the standard prevention is scikit-learn's Pipeline, which fits preprocessing only on each fold's training data. Section 4 applies this split-then-fit rule.

2.3 Leakage in machine learning education

Data leakage receives little dedicated attention in ML curricula. The closest prior work identified is Fund et al. (2025). Their modules, deployed in two introductory courses, have students reproduce and then correct published findings affected by leakage. The Audit Lab shares this before-and-after mechanic but differs in two ways. First, its flawed artifacts are

notebooks built to imitate GenAI output, and the Section 2.5 search located little prior evaluation of such artifacts. Second, its interaction is structured by a Socratic tutor and a hypothesis-before-help rule. Section 3.8 proposes a protocol to test whether these differences change outcomes.

2.4 Pedagogical grounding and concurrent Socratic-tutor design work

Productive failure (Kapur, 2008) holds that learners who struggle before direct instruction can develop more durable understanding, provided the struggle is structured and followed by consolidation. A meta-analysis found this advantage for conceptual knowledge and transfer but not for procedural knowledge (Sinha & Kapur, 2021). Research on erroneous examples, in which learners find and fix errors in worked solutions, reports transfer benefits for learners with sufficient prior knowledge and gains on delayed tests (Große & Renkl, 2007; McLaren et al., 2015). Predict–Observe–Explain tasks ask learners to predict an outcome before observing it (White & Gunstone, 1992). These strands motivate the Section 3.3 cycle. Learners audit a flawed artifact, predict before tool assistance, and receive pass/fail feedback.

Socratic sequencing of AI-tutor responses is a current design target. Lucas et al. (2026) report SocratiCode, a generative-AI programming tutor for K-12 students. Its four-week participatory design study moved it toward guided questioning and mandatory pauses for learner input. The study is small and formative, and its population differs from the one targeted here. Sun et al. (2026) compared Socratic and direct-answer generative-AI tutoring for programming. They found differences in learner behavior and interaction quality, but their task was writing new code rather than auditing a flawed artifact. The Section 2.5 search located no evaluation in a code-auditing context.

2.5 Search methodology and scope

The related-work search was targeted rather than systematic. It used Google Scholar, the ACM Digital Library, IEEE Xplore, arXiv, and the general web between August and September 2026. The closest finds were Fund et al. (2025), Lucas et al. (2026), and Sun et al. (2026), and the novelty claims are scoped to them.

3. Materials and Methods

3.1 Design and development research approach

This study follows the product-and-tool strand of design and development research (DDR; Richey & Klein, 2007). Phase 1 covers the analysis, design, and development of an instructional product. Phase 2 covers its evaluation with learners. Only Phase 1 is reported. Phase 1 involved no external experts or learners.

Analysis. Three bodies of evidence were reviewed: leakage taxonomies and detection tools (Sections 2.1 and 2.2), learning theory and related evidence (Section 2.4 and Section 1), and the closest instructional designs (Sections 2.3 and 2.4). The outputs were a problem statement (learners can produce GenAI notebooks whose validity they cannot judge), the learner profile (Section 1), and candidate failure types.

Design. A candidate failure type was retained if it is an error documented in human-authored ML code (in the leakage literature or in standard applied guidance) that GenAI

output can reproduce. Each retained type was assigned the strongest feasible feedback mechanism. This was a Tier 1 executable check where a deterministic property can be tested, and otherwise Tier 2 rubric judgment. Theory was translated into operational rules. For example, productive failure became the requirement to predict before testing and to commit to a hypothesis before case-specific help. The outputs were the design principles, audit cycle, taxonomy, tutor policy, and feedback tiers (Sections 3.2–3.6; Table 1).

Development. Preprocessing leakage and target leakage were developed first. They contrast a numerically subtle failure with a dramatic one, and they exercise both the Tier 1 route and the Tier 3-to-Tier 2 route. Each notebook and script was executed and revised until it ran as specified. Each document was revised over successive author design reviews. The Phase 2 threat model served as a review lens. For example, the Tier 1 check reports PASS or FAIL without naming the cause, to limit hidden-test leakage. The outputs were the Phase 1 package (Appendix B), the walkthrough (Section 4), and the Phase 2 protocol (Section 3.8).

3.2 Design principles

The framework follows five principles. First, GenAI output is an object of scrutiny, not an oracle. Second, seeded errors are modeled on documented leakage patterns (Sasse et al., 2025) rather than contrived puzzles, to support transfer. Third, feedback is procedural where possible: a Tier 1 test passes or fails instead of a tutor judging the learner. Fourth, the tutor asks rather than tells until a hypothesis is committed. This constraint is to be enforced by a policy layer between the chat interface and the model, not by a prompt alone. Fifth, each scenario pairs the plausible but invalid result with the corrected one, which makes the cost of the error visible.

3.3 The audit cycle

Learners work through five steps per scenario. Inspect: the learner receives a GenAI-style notebook with its own summary and recommendation. Predict: before tool assistance, the learner writes a hypothesis about which pipeline step is suspect and how the metric is biased. Test: the learner runs the Tier 1 checks, which report pass or fail without stating the cause. Repair: the learner fixes the pipeline and reruns both versions. Explain: the learner writes an audit memo (Appendix B) on what was wrong, why it biased the result, what changed numerically, and whether the corrected result still supports the recommendation. The memo is scored with the Tier 2 rubric. Interpretively, Inspect and Test supply monitoring evidence, Predict supplies a prospective judgment, Repair supplies control, and Explain supplies retrospective reflection (Nelson & Narens, 1990). These processes are not measured.

3.4 Validity-failure-mode taxonomy

The eight scenarios (Table 1) fall into three categories by the validity claim violated. The first category is data and pipeline leakage (six scenarios; Kapoor & Narayanan, 2023; Sasse et al., 2025). The second is evaluation and validation errors (imbalanced-metric misuse). The third is interpretation and reporting errors (correlation-to-causation overclaiming). Preprocessing and target leakage are implemented in Section 4 and have an artifact-complete Phase 1 package: notebooks, a Tier 1 check or Tier 3 triage script, task sheets, a memo template, an instructor guide, a rubric, a tutor policy, and a consolidation activity. Each planned file exists and runs or reads as described, but none has been piloted or validated. The other six scenarios are specified at design level only.

Table 1 AI Audit Lab validity-failure-mode taxonomy

#	Category	Failure type	Typical GenAI presentation	Primary feedback mechanism
1	Data / pipeline leakage	Feature-availability leakage	Future-dated feature used to predict an earlier event	Tier 1: feature timestamp precedes prediction timestamp
2	Data / pipeline leakage	Temporally invalid split	Random (non-chronological) train/test split applied to time-ordered data	Tier 1: all training-set timestamps precede all test-set timestamps
3	Data / pipeline leakage	Target leakage†	Feature is a near-proxy for the label, recorded after the outcome	Tier 2: provenance and temporal-availability review; mutual-information analysis may serve only as a preliminary Tier 3 triage signal
4	Data / pipeline leakage	Preprocessing leakage†	Scaler/encoder fit on full dataset before splitting	Tier 1: fit-object parameters independent of test-set values
5	Data / pipeline leakage	Group / entity leakage	Same customer or user appears in train and test sets	Tier 1: zero entity-ID overlap between splits
6	Data / pipeline leakage	Test-set reuse	Hyperparameters tuned repeatedly against the test set	Tier 1: validation set distinct from final test set across tuning runs
7	Evaluation and validation errors	Imbalanced-metric misuse	Accuracy reported on a highly imbalanced label	Tier 2: rubric-based review of whether the reported metric matches class balance, decision costs, and the stated evaluation objective
8	Interpretation and reporting errors	Correlation-to-causation overclaim	Predictive association described as a causal driver	Tier 2 + Tier 3 (routes to Tier 2): causal-language pattern in model summary text (specified; not yet implemented)

Note. Tiers 1–3 are defined in Section 3.6. † Implemented in the Phase 1 package; the other six items are design-level only. Item 1 requires feature timestamps, and item 6 requires logging of evaluation calls across tuning runs.

Items 1–6 draw on the leakage subtypes of Kapoor and Narayanan (2023) and Sasse et al. (2025). The mapping is an instructional synthesis, not a verified crosswalk. Items 7 and 8 reflect standard metric-selection guidance and the correlation–causation distinction. The contribution is packaging these failure types into GenAI-facing audit scenarios with tiered feedback, not the taxonomy itself.

3.5 Socratic AI tutor protocol

Before a hypothesis is submitted, the tutor may only ask a clarifying question, confirm in general terms whether the learner's line of inquiry is productive, or explain a general concept without applying it to the notebook. After a hypothesis is submitted and at least one Tier 1 check has run, the tutor may discuss the specific pipeline. This sequencing is the operational difference from standard AI tutoring (Section 3.8). The policy exists as a written specification, and no enforcement layer has been built.

3.6 Feedback tiers

Tier 1 (executable invariants) checks deterministic properties, such as no train–test index overlap or preprocessing parameters that do not depend on test-fold values. It is ground truth only for the property instrumented, not for overall workflow validity. Tier 2 (instructor-scored reasoning checks) applies a rubric to the audit memo (Appendix B). It is not claimed to be automatable without loss of validity. Tier 3 (automated heuristic flags) uses no large language model (LLM) and currently implements one heuristic, a feature–label mutual-information ranking. The ranking reports how many standard deviations (SD) the top feature's score lies above the mean of the others, called the SD gap. A causal-language check for item 8 is specified but not implemented. A flag does not by itself establish leakage or an unwarranted claim, so Tier 3 flags always route to Tier 2 review. Their precision and recall are unknown.

3.7 Computational walkthrough procedure

To illustrate the audit cycle and that failure severity can differ by error type, Section 4 uses a synthetic churn-style dataset with 4,000 records, 20 features (8 informative and 4 redundant), and a 20% positive class. It was generated with scikit-learn make_classification (random_state = 42). Both demonstrations use logistic regression (lbfgs, max_iter = 1000) and ROC-AUC (area under the receiver operating characteristic curve, called AUC below). AUC is averaged over 20 stratified 75/25 splits (seeds 42–61), so only the audited error differs. The learner notebooks use one such split (random_state = 42). The Demonstration B inflation is set by the chosen proxy strength (3% flip probability) and is illustrative, not an estimate for real target leakage. Results come from one run of audit_lab_worked_example_v2.py. They describe two pipelines on one synthetic dataset, not a pedagogical effect.

3.8 Proposed Phase 2 evaluation protocol

A mixed, split-plot design is proposed. No human-subjects data have been collected. Participants would be individually randomized to standard AI tutoring or the Audit Lab. Both conditions use the same model, scenarios, audit cycle, checks, and memo. The explanation-only tutors differ only in sequencing. The standard tutor may hint at any time and state the cause if asked, whereas the Audit Lab tutor withholds case-specific help until a hypothesis is committed (Section 3.5).

After the tutorial scenarios, learners complete a delayed, AI-free transfer task. The task is drawn from a held-out pool of unseen error instances of the failure types that have a Tier 1 check (Table 1, items 1, 2, 4, 5, and 6), so that the primary outcome can be computed. Condition is between subjects so that the transfer skill cannot carry over. The primary outcome is the Tier 1 pass rate after the first proposed repair on the transfer task, computed by the scorer and not shown to the learner during transfer. It would be analyzed with logistic regression (condition fixed), adding participant and item random effects only if participants complete several transfer items. Failure types without a Tier 1 check (items 3, 7, and 8) enter only the Tier 2 secondary outcome.

Design review identified seven threats to internal validity, each paired with a control: scenario-reuse contamination, hidden-test leakage, AI-model context leakage, assistance asymmetry, time-on-task confound, scenario-difficulty calibration, and between-participant contamination. Section 3 of the supplementary protocol gives each pairing, and Appendix A summarizes the protocol.

4. Reproducible Computational Demonstration

4.1 Demonstration A: preprocessing leakage

A GenAI-style notebook fits a StandardScaler on the full feature matrix before splitting. The corrected pipeline splits first, fits the scaler on the training fold, and transforms the test fold without refitting. Across 20 splits, the leaky and corrected pipelines both had a mean AUC of 0.7808 (SD = 0.0167), with a naive 95% confidence interval (CI; mean ± 1.96 standard errors across splits) of [0.7735, 0.7882]. The intervals are naive and descriptive only, because the splits share one dataset. The means are equal to four decimals, but the individual-split AUCs are not identical (Section 4.3).

4.2 Demonstration B: target leakage

A second notebook includes a noisy proxy of the outcome (3% synthetic flip probability plus small continuous jitter; realized match rate 96.975%, 3,879 of 4,000 rows). This is analogous to a field recorded after the outcome (Kapoor & Narayanan, 2023). Across the same splits, the leaky pipeline's mean AUC was 0.9869 (SD = 0.0028; naive 95% CI [0.9857, 0.9882]). With the proxy removed, the mean AUC was 0.7808 (SD = 0.0167; naive 95% CI [0.7735, 0.7882]). This baseline is identical to Demonstration A's by construction.

4.3 Interpretation of the demonstration

The Demonstration A result is not exact affine invariance. For an unregularized linear classifier, per-feature rescaling would be absorbed into the coefficients, leaving AUC unchanged. The default logistic regression in scikit-learn applies L2 regularization (C = 1.0), which is not scale-invariant. Per-split AUCs differ by up to 6.8×10^{-5} in this run, although for one split (seed 48) they agree to the eight decimal places recorded. The difference is probably small because regularization is weak at this strength and training-fold statistics (3,000 of 4,000 rows) closely approximate full-data statistics. This explanation was not tested, for example by varying C. The workflow is still invalid. With a non-linear model or a calibration-sensitive metric such as log-loss, the same leakage could matter more.

Demonstration B shows the large inflation usually associated with leakage. Together, the demonstrations counter the misconception that a small AUC change means no problem.

5. Discussion

The AI Audit Lab rests on three mechanisms that theory suggests may support learning. First, the predict step and the hypothesis-before-help rule require commitment to an explanation before guidance, as productive failure prescribes (Kapur, 2008; Section 2.4). Second, the Tier 1 check returns only PASS or FAIL, and the memo asks learners to compare prediction with result. In Nelson and Narens's (1990) terms, this supplies an external monitoring signal while leaving control with the learner. Because the tutor does not diagnose, the design targets the offloading pattern reported by Gerlich (2025). Third, the paired flawed-versus-corrected comparison, especially Demonstration A, is intended to teach that an unchanged metric does not establish validity.

Whether the auditing habit transfers to unfamiliar GenAI output is the central open question, hence the delayed transfer outcome. Offloading may be relocated rather than

reduced, for example onto repeated Tier 1 runs or an outside assistant. The tutor policy is untested against circumvention, and the workload may exceed what learners below the intended profile can manage. Productive-failure and erroneous-example evidence comes from domains such as mathematics rather than code auditing (Große & Renkl, 2007; McLaren et al., 2015; Sinha & Kapur, 2021), so the expected mechanisms may not carry over.

5.1 Relation to prior work and design implications

Compared with Fund et al. (2025) and SocratiCode (Lucas et al., 2026), the Audit Lab combines GenAI-style artifacts with a hypothesis-before-help sequence. This is a design rationale, not a claim of superiority. Because failure severity can differ by error type, assessment that rewards only large metric changes would under-train detection of numerically silent failures. Tier 1 checks therefore test directly the specific property they instrument, although passing them does not establish that the whole workflow is valid.

5.2 Limitations

This paper makes no claim of pedagogical effectiveness. The framework has not undergone expert review, usability testing, learner try-out, or formative evaluation, which should precede Phase 2. All design decisions were made by a single author, and the learner profile is an unchecked assumption. The premise that GenAI assistants frequently produce these errors is untested. The scenarios model errors documented in human-authored code that GenAI output can reproduce, and the seeded notebooks were constructed by design rather than sampled from GenAI output. The walkthrough uses synthetic data, and the Section 4.3 explanation is specific to linear classifiers with a ranking-based metric. Tier 1 invariants can miss failures outside their coverage. The tutor policy exists only as a specification and needs red-team testing. Tier 3 flags and the Appendix B instruments have no reliability or validity evidence. The related-work search was targeted rather than systematic, and the package is specific to Python and Jupyter.

6. Conclusion

As GenAI assistants become a common first step in data analysis, producing a working notebook is increasingly separate from judging whether its conclusions hold. The AI Audit Lab makes GenAI output the object of a structured audit. Until expert review, learner try-out, and Phase 2 are completed, its contribution is the Phase 1 design, its documented development, and the protocol for testing it.

Acknowledgments

The author thanks the maintainers of the open-source scikit-learn and NumPy projects used in Section 4.

The generative AI tool Perplexity (Perplexity AI) was used during manuscript preparation for deep research and literature searching, structural organization, language editing, and programming assistance (drafting and debugging audit_lab_worked_example_v2.py). A generative AI assistant was also used for final language editing and formatting to the journal template. The author personally inspected and executed all code, checked all numerical results, and checked all references against their original sources. The author takes full responsibility for the accuracy, originality, and integrity of the manuscript. Generative AI was not used to fabricate or alter data, results, or references.

Declaration of Interest

The author declares no conflict of interest.

Data Availability Statement

The code, seeds, pinned dependencies (numpy==2.2.6, scikit-learn==1.7.1), and outputs for the Section 4 demonstration are supplementary material under the MIT license (audit_lab_worked_example_v2.py). Reproduction in other environments has not been verified. The Phase 2 protocol (AI_Audit_Lab_Phase2_Protocol_v6) and the Phase 1 package for the two implemented scenarios are also supplementary material. None of these materials has been piloted with learners.

References

Fund, F., Saeed, M., Malik, S., & Ishak, K. (2025). Learning from irreproducibility: Introducing data leakage case studies for machine learning education. In *Proceedings of the 3rd ACM Conference on Reproducibility and Replicability* (pp. 224–228). Association for Computing Machinery. https://doi.org/10.1145/3736731.3746153

Gerlich, M. (2025). AI tools in society: Impacts on cognitive offloading and the future of critical thinking. *Societies, 15*(1), Article 6. https://doi.org/10.3390/soc15010006

Große, C. S., & Renkl, A. (2007). Finding and fixing errors in worked examples: Can this foster learning outcomes? *Learning and Instruction, 17*(6), 612–634. https://doi.org/10.1016/j.learninstruc.2007.09.008

Kapoor, S., & Narayanan, A. (2023). Leakage and the reproducibility crisis in machine-learning-based science. *Patterns, 4*(9), Article 100804. https://doi.org/10.1016/j.patter.2023.100804

Kapur, M. (2008). Productive failure. *Cognition and Instruction, 26*(3), 379–424. https://doi.org/10.1080/07370000802212669

Lau, S., & Guo, P. (2023). From "Ban it till we understand it" to "Resistance is futile": How university programming instructors plan to adapt as more students use AI code generation and explanation tools such as ChatGPT and GitHub Copilot. In *Proceedings of the 2023 ACM Conference on International Computing Education Research V.1* (pp. 106–121). Association for Computing Machinery. https://doi.org/10.1145/3568813.3600138

Lucas, C., Bihani, A., Kukka, R., Tsai, C.-H., Sarker, J., & Imran, M. M. (2026). *Towards SocratiCode: Designing a generative AI-based programming tutor for K-12 students through a 4-week participatory design study* (arXiv:2605.17857). arXiv. https://doi.org/10.48550/arXiv.2605.17857

McLaren, B. M., Adams, D. M., & Mayer, R. E. (2015). Delayed learning effects with erroneous examples: A study of learning decimals with a web-based tutor. *International Journal of Artificial Intelligence in Education, 25*(4), 520–542. https://doi.org/10.1007/s40593-015-0064-x

Nelson, T. O., & Narens, L. (1990). Metamemory: A theoretical framework and new findings. In G. H. Bower (Ed.), *The psychology of learning and motivation* (Vol. 26, pp. 125–173). Academic Press. https://doi.org/10.1016/S0079-7421(08)60053-5

Richey, R. C., & Klein, J. D. (2007). *Design and development research: Methods, strategies, and issues*. Lawrence Erlbaum Associates.

Sasse, L., Nicolaisen-Sobesky, E., Dukart, J., Eickhoff, S. B., Götz, M., Hamdan, S., Komeyer, V., Kulkarni, A., Lahnakoski, J. M., Love, B. C., Raimondo, F., & Patil, K. R. (2025). Overview of leakage scenarios in supervised machine learning. *Journal of Big Data, 12*(1), Article 135. https://doi.org/10.1186/s40537-025-01193-8

Sinha, T., & Kapur, M. (2021). When problem solving followed by instruction works: Evidence for productive failure. *Review of Educational Research, 91*(5), 761–798. https://doi.org/10.3102/00346543211019105

Sun, D., Zheng, Y., Xu, J., & Yang, Z. (2026). When generative AI meets Socratic method: Investigating programming learning dynamics through behaviours, interaction qualities and perceptions. *Journal of Computer Assisted Learning, 42*(2), Article e70210. https://doi.org/10.1002/jcal.70210

Truong, O., Zhang, T., Marchareddy, A., Lee, R., Busold, J., Socas, M., & AlOmar, E. A. (2025). LeakageDetector 2.0: Analyzing data leakage in Jupyter-driven machine learning pipelines. In *2025 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (pp. 895–899). IEEE. https://doi.org/10.1109/ICSME64153.2025.00102

White, R., & Gunstone, R. (1992). *Probing understanding*. Falmer Press.

Yang, C., Brower-Sinning, R. A., Lewis, G. A., & Kästner, C. (2022). Data leakage in notebooks: Static detection and better processes. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering* (pp. 1–12). Association for Computing Machinery. https://doi.org/10.1145/3551349.3556918

Appendix A. Phase 2 Evaluation Protocol Summary

The supplementary protocol also specifies the following. Both conditions share the same language model, scenario corpus, session time, and calibrated task difficulty. Rubric-scored diagnosis is a secondary outcome, and tutorial-phase performance is exploratory. Process measures such as time-on-task and hint requests are logged. Before implementation, the protocol requires scenario calibration, red-team testing for leakage, verification of assistance symmetry, and a power simulation from pilot data.

Appendix B. Example Research Instruments

Both instruments are condensed from the supplementary Phase1_Learner_Memo_Template and Phase1_Tier2_Rubric for Scenario A (preprocessing leakage) and Scenario B (target leakage). Neither has been piloted, and no reliability or validity evidence exists for either.

B.1 Learner audit memo (Explain step)

One memo is completed per scenario audited, with eight prompts:

1. Assistant's original claim: quote or paraphrase the assistant's recommendation and the AUC it reported.
2. Your prediction: copy the Predict-step hypothesis exactly as written before the Test step, without revision.
3. Test result: for Scenario A, the Tier 1 PASS/FAIL result before and after repair; for Scenario B, the Tier 3 ranking and the top feature's mutual-information score and SD gap.
4. What was wrong: name the specific pipeline step (Scenario A) or feature (Scenario B), not just the general category.
5. Why it biased the result: explain the mechanism and the direction of the bias.

6. Numerical comparison: report the flawed and repaired AUC and interpret the size and direction of the difference.
7. Does the corrected performance still support the original recommendation? Answer yes or no, justify with the corrected number, and note limits on confidence (for example, one split of one synthetic dataset).
8. Provenance argument (Scenario B only): explain when the feature's value would realistically be available relative to the label and why it was judged illegitimate or legitimate. A high mutual-information score alone is not an acceptable justification.

B.2 Tier 2 scoring rubric

Each dimension is scored 0–3 against the anchors in Table 2.

Table 2 Tier 2 scoring rubric for the audit memo (condensed)

Dimension	0	1	2	3
1. Diagnostic accuracy	Misidentifies the cause	General category only	Specific step or feature, minor imprecision	Precise identification of the actual cause
2. Causal explanation	None, or factually incorrect	Vague or generic (e.g., "leakage is bad")	Correct mechanism but incomplete (e.g., no direction of bias)	Complete mechanism, including why the bias runs in that direction
3. Numerical interpretation	Numbers missing or misreported	Reported but not interpreted	Correctly interpreted directionally	Also contextualized (e.g., a near-zero change in Scenario A does not make leakage harmless)
4. Recommendation judgment	Not addressed	Yes/no without justification	Justified with the corrected number	Also appropriately hedged (e.g., one split of one synthetic dataset)
5. Provenance reasoning (Scenario B only)	None, or relies only on high mutual information	Asserts (il)legitimacy without reasoning about timing	Reasons whether the feature could be known before the label	Specific reasoning about real-world provenance and timing beyond the Tier 3 signal

Note. Scenario A total = Dimensions 1–4 (maximum 12); Scenario B total = Dimensions 1–5 (maximum 15). No pass/fail threshold is set. It is left to instructor judgment pending reliability and validity evidence.