

Auditing What Program Images Hide and Preserve

Anonymous Authors

Abstract—Reusing a reversible program image for masked code prediction requires hiding the target while preserving the available context. Recovery payloads and integrity metadata may still depend on the target; removing them can also discard needed context. We propose a task-specific audit with two obligations: invariance to hidden completions and preservation of the fields promised to a consumer. A paired test harness applies established noninterference and metamorphic-testing principles to the carrier-to-view adapter before training. In a bounded study of one pinned CodeImage deployment, fixed redactions expose a reference-class marker that leaves 36 differing pixels after the main panels, recovery regions, and digest stamp are hidden. A separate pair shows an information-loss failure: views become identical although an allowed operator differs. Exact preservation of designated non-target marks does not prevent this collision. These emerging results motivate construction of views from permitted partial records and evaluation of both obligations. No learned generator or model-accuracy gain is claimed.

Index Terms—program representation, information flow, masked modeling, evaluation, software testing

I. PROBLEM AND RESEARCH IDEA

Image-like program representations offer a possible substrate for predicting and refining several structural elements at once. AST-derived images have already been studied for code classification [1], and masked parallel image generation is established [2]. Neither result establishes that a reversible program image is a suitable input for a masked code-generation experiment. A carrier designed to recover a complete program has an intentionally different information contract from a view designed to hide part of that program.

Consider a program image in which a node is drawn using colored panels. A proposed learning pipeline masks the panels corresponding to an unknown reference. If the reference class is also drawn beside those panels, the target is still observable. Removing that marker may still leave a source-derived checksum elsewhere in the image. A checksum difference is not proof that a model can recover the target, but it refutes a claim that the view depends only on the declared visible information. Conversely, metadata used only by an evaluator is outside the model’s view and should not be counted as model access.

We ask: *how can we test whether a program-image adapter hides the declared target while preserving the program information promised to its consumer?* Our proposal is a task-specific, paired audit at the interface between the carrier and its prospective consumer. It combines an explicit available-information projection, valid counterfactual programs, fixed redaction rules, and two checks: hidden-target invariance and

visible-information preservation. The first detects residual dependence; the second prevents an empty, constant image from passing as a useful solution.

Our technical artifact is a task-specific adapter and paired test harness for CodeImage, a publicly available program-image codec [3]. It records how fixed redaction stages affect target dependence, retained records, and retained marks, and produces witnesses of both residual dependence and information loss. The adapters are research constructions for a prospective masked-learning task; the study evaluates their information boundary, not a deployed generation model. This makes the proposed conversion to a learning view falsifiable before model training.

II. TASK-SPECIFIC VIEW CONTRACTS

Let p be a valid program, $E_c(p)$ its carrier under fixed public settings c , and H a declared set of hidden structural fields. Let $\mathcal{A}_H(p)$ be a fixed, executable projection of the partial-program information promised to the consumer. Its schema is declared before comparing adapters; it is not defined by whatever survives a redaction. A view adapter M_H produces

$$V_H(p) = M_H(E_c(p)) \quad (1)$$

as the actual input to the model or a specified downstream stage. For a raster consumer this means decoded pixel values and any supplied side information, not every byte of a PNG container. A prediction-time decoder, candidate selector, or retry procedure that accesses additional information must have its own declared view.

A. Hidden-Target Invariance

For deterministic views, the desired relation is

$$\mathcal{A}_H(p_0) = \mathcal{A}_H(p_1) \implies V_H(p_0) = V_H(p_1). \quad (2)$$

The two programs must both be valid, and their hidden completions must differ. We hold the public profile, title, and mask selection fixed. If canvas size is provided by the task, it belongs in $\mathcal{A}_H$; programs with different provided sizes are not an admissible pair. If it is not provided, size computed from a hidden completion is a potential dependence to examine. The same qualification applies to node counts, offsets, and source locations. For randomized views the corresponding condition concerns distributions; a common-seed experiment alone is not a distributional proof.

Equation (2) specializes established information-flow reasoning. Paired testing of noninterference already generates indistinguishable inputs and searches for counterexamples [4];

our relation is not a new security definition. Its violation gives a *witness of undeclared target dependence*. It does not measure mutual information, demonstrate target recovery, or prove that a learned model will exploit the difference. Hashes are a useful example: they can distinguish two completed inputs without providing a practical predictor for unseen completions. An observed equality is also only evidence for the tested pair, not a universal guarantee.

B. Visible-Information Preservation

Invariance alone is insufficient: the adapter that discards everything satisfies it. Full preservation requires an extraction function R such that

$$R(V_H(p), H, c) = \mathcal{A}_H(p). \quad (3)$$

For a structured view, retained fields can be extracted and compared exactly. For a compatibility raster adapter, unchanged non-target marks provide only a weaker check. In particular, erasing a recovery band may remove allowed scalar values as well as the hidden target. Equal views for two programs with different permitted fields would refute Equation (3), even if every retained mark were unchanged. Such a *preservation collision* is separate from an invariance violation: it deliberately changes available information.

A sufficient design for invariance is to construct a view as

$$V_H(p) = B(\mathcal{A}_H(p), H, c), \quad (4)$$

where B receives no hidden completion. Checksums, layout, and auxiliary features must be omitted or recomputed from permitted inputs. For fixed H, c and deterministic B , the invariance argument is immediate. Preservation remains separate: a constant B satisfies the factorization while discarding context. When a carrier lacks a partial-structure renderer, compatibility masking requires empirical checks for residual dependencies. Our coordinate redactions are not verified implementations of Equation (4).

C. Audit Procedure

The audit has four steps. (1) Declare H , a fixed available projection, and the exact consumer view; reject invariance candidates whose permitted fields differ. (2) Independently encode both valid completions under the same settings. (3) Apply a fixed adapter and check view equality and retained fields or marks. Separately search for preservation collisions by varying a permitted field. (4) Report the inputs, target fields, differing record paths or pixel coordinates, and adapter settings. Repeating the same input provides a determinism control. Source-guided stages localize a dependence without changing the task policy or enlarging masks in response to observed differences.

Dynamic counterfactual testing may miss dependencies, and a source-derived rule can be wrong. Its purpose is to falsify an overbroad information claim before model training, while making the boundary and surviving information reviewable. It does not provide complete taint tracking.

III. A SOURCE-GUIDED PROGRAM-IMAGE AUDIT

A. Deployment and Information Channels

We pinned nine public JavaScript assets and their embedded WASM by SHA-256. The runtime reported Topaz 5.9.6. The experiment uses a Node.js adapter with network access disabled inside the kernel context; it neither modifies product assets nor rebuilds the compiler. This is a gray-box study of one deployment, with no established identity between its binaries and a particular public source commit.

The inspected system contains structured node records, visual marks, protected recovery bytes, page metadata, and a detached source-recovery key. A prior carrier control on four constructed programs confirmed that a complete logical wire plus an explicit interpretation profile could reproduce the same normalized source as the image route. Matching detached keys reproduced the original UTF-8 strings. Those observations concern recovery and do not establish a masked-learning result. The raster consumer receives neither detached keys nor extracted wire as separate inputs. Recovery pixels remain visible until their declared redaction stage; differences in unused sidecar bytes do not count as pixel dependence.

The public renderer suggests two less obvious channels. Identifier-class marks are drawn beside the three main panels of a node. A small stamp in the page decoration depends on the page-set digest. These observations motivate fixed, staged masks rather than a target-panel mask alone. The codec uses these channels for legitimate representation and integrity purposes. Our question is what happens if its output is reused in a task that declares the corresponding target information unavailable.

B. Fixtures, Adapters, and Observations

The protocol fixes short paired JavaScript programs, hidden fields, and mask rectangles before execution. Four completion pairs change $+/-$, $< / >$, the numeric literal $1/2$, or a reference to parameter a/b . A same-width bound-name control distinguishes source spelling from normalized identity; a $1/10$ pair probes geometry. Before execution, we added a separate preservation control that changes an allowed operator while hiding the unchanged literal. A same-source repeat checks determinism. These fixtures share contexts and some inputs; they are not independent samples from a program distribution.

The fixed projection retains node kinds, topology, code-words, non-target scalar fields, identifier-class references, and profile fields. It excludes the target bundle, two instance digests, and source offsets and row/column coordinates. This structure-completion policy excludes source geometry throughout the experiment. Raster shape is supplied: all recorded images are 1304×199 pixels, with equal mask maps within each pair. A different shape would be reported separately, not resized to force equality.

Structured stages mask the target (S1), also remove the two digests (S2), and also remove geometry (S3, equal to the fixed projection). Raster stages mask target panels (V1), also the header and a fixed recovery-band envelope (V2), also the

TABLE I

DIFFERING RGB PIXELS AFTER CUMULATIVE MASKS. V1: TARGET PANELS; V2: ALSO HEADER AND RECOVERY BAND; V3: ALSO STAMP; V4: ALSO TARGET IDENTIFIER MARKER. THE LAST TWO ROWS ARE DISTINCT DIAGNOSTIC ROLES. ALL SIX PAIRS HAVE EQUAL DECLARED AVAILABLE STRUCTURE.

Pair	V1	V2	V3	V4
Operator $+/-$	3,489	225	0	0
Operator $</>$	3,376	100	0	0
Literal $1/2$	3,011	175	0	0
Reference a/b	3,140	236	36	0
Bound-name control	2,967	175	0	0
Length diagnostic	3,095	275	0	0

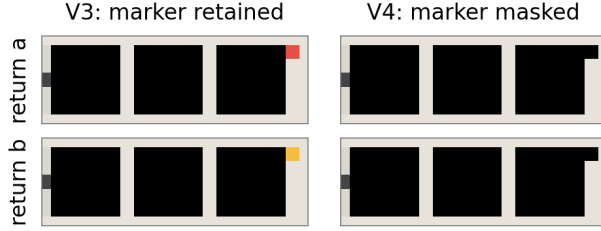


Fig. 1. Exact crops of the reference pair’s observed RGB views. At V3 the target panels are black but the class marker still differs. V4 also masks that marker. Crops are enlarged with nearest-neighbor sampling; the audit compares full tensors.

digest stamp (V3), and also the target identifier marker (V4). We record RGB differences, bounding boxes, mask areas, and retained marks. The source-derived rectangles are fixed before execution; observed differences never select them. Original images must pass the unchanged product decoder. A separate Python/Pillow checker reconstructs the RGB buffers and masks from the recorded inputs and independently recomputes structure and pixel comparisons. This verifies measurements, not program semantics. Masked tensors are prospective inputs, not valid replacement carrier files.

C. Emerging Results

All six completion/diagnostic pairs satisfy the fixed projection premise. The repeat control has identical structured and RGB observations at every stage. At S1, the four completion pairs still differ in two instance digests. S2 removes those differences. The length diagnostic retains 12 differing source-position fields at S2 and becomes equal at S3. All retained structured fields are unchanged by their declared redactions.

Table I shows the raster results. Main-panel masking alone changes none of the pairwise difference counts: the inspected panels encode node kind, not operator or numeric value. After removing header and recovery bytes, the $+/-$ pair still differs in 225 pixels inside the decorative digest stamp. Masking the stamp removes these differences. The reference pair has an additional channel: 236 pixels differ at V2, 36 remain at V3, and none at V4. The last 36 form the 6×6 identifier-class marker outside the three target panels (Fig. 1).

All six pairs become pixel-identical at V4. The final masks cover 11.01–12.08% of the image, retaining the specified panel and marker pixels of all 8–10 non-target nodes without overlap. This bounded mark-preservation result is weaker than preserving every required field.

The preservation control demonstrates the distinction directly. Its sources are function `f(a){return a+1;}` and the corresponding `a-1` variant, both followed by LF. The hidden field is the literal value 1; the operator remains in $\mathcal{A}_H$. S3 differs at exactly that operator field, yet V3 and V4 produce identical RGB views, shapes, and mask maps. All specified non-target marks remain unchanged. Thus no reader receiving only that view and the fixed public settings can recover the different promised operators for both inputs. The compatibility raster adapter fails field preservation, despite passing the completion-pair invariance checks. These observations identify two separate failure modes rather than establish a completed masking solution.

IV. RELATION TO PRIOR WORK

Villmow et al. control shared-identifier, indentation, and syntax cues in contextualized code retrieval [5]. We examine additional dependencies created when a reversible carrier becomes a partial-program observation.

Noninterference testing [4], metamorphic testing of machine-learning systems [6], and dependence analysis supply the underlying principles. Drobnjaković et al. derive a sound static analysis of data leakage from dependency semantics [7]; our dynamic audit has no corresponding soundness guarantee. Janisiów et al. retain common molecular subgraphs, mask differing fragments, and compare predictions [8]. We inspect the consumer input before learning and separately test loss of fields promised by a program-representation adapter. Our contribution is an executable audit that exposes both residual target dependence and loss of required context in program-image views.

V. FUTURE PLANS

Generality and automation. First, we will freeze the rules and evaluate held-out contexts from a small typed grammar, including a second, independently implemented representation. Seeded target-dependent channels will test detection; score-only metadata and permitted-context changes will test correct treatment of nonviolations. We will measure pair-premise errors, seeded-channel detection, preservation failures, and cost separately. A renderer built from permitted partial records must also provide a reader satisfying Equation (3); independence alone is insufficient. Field-to-channel instrumentation is a prospective automation strategy. Passing generated pairs remains distinct from a sound noninterference guarantee.

Exploitability. The current endpoint is view equality. A follow-up will compare raw, partially masked, and observation-only views on held-out program families to test whether residual channels change prediction accuracy. Decoder-only and simple non-neural controls will establish whether recovery is possible without learning. If a varying

checksum offers no useful prediction advantage, the result should be reported as dependence without demonstrated practical exploitation. Metadata used only after prediction for scoring will remain outside this comparison.

Representation value. Only after fixing the information boundary will we compare linear and grid placements using common node records, masking units, model capacity, and declared training and inference budgets. A random-layout control will test whether structural neighborhoods matter. Executability and independent behavioral tests will remain separate from syntactic validity; stronger execution tests can materially change code-generation conclusions [9]. Any benefit limited to extra side information, deterministic repair, or additional compute will support that narrower explanation.

VI. CONCLUSION

The audit distinguishes remaining dependence on hidden completions from loss of information promised as available. Pixel invariance and unchanged non-target marks alone do not establish the field-level contract. The current evidence is limited to one pinned codec and selected short programs; source-guided rules may overfit its renderer, and source inspection does not prove binary equivalence. The projection must be justified for each consumer. Our prototype supplies reproducible witnesses and scoped preservation checks. A renderer that satisfies both contracts and evidence of downstream model effects remain future work.

DATA AVAILABILITY

We have prepared an anonymized replication package containing the inputs, pinned asset hashes, adapters, raw observations, and independent verification scripts. Public deployment assets are identified by URL and SHA-256; a source rebuild is not included. The package is not yet publicly archived.

ACKNOWLEDGMENT OF AI ASSISTANCE

OpenAI ChatGPT [10] assisted with the abstract and all sections, reference checking, and implementation of the research harness and analysis scripts. Measurements were computed from recorded executions; no learned-model results are claimed.

REFERENCES

- [1] Z. Bilgin, “Code2Image: Intelligent code analysis by computer vision techniques and application to vulnerability prediction,” arXiv preprint arXiv:2105.03131, 2021. [Online]. Available: <https://arxiv.org/abs/2105.03131>
- [2] H. Chang, H. Zhang, L. Jiang, C. Liu, and W. T. Freeman, “MaskGIT: Masked generative image transformer,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 11 315–11 325. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2022/html/Chang_MaskGIT_Masked_Generative_Image_Transformer_CVPR_2022_paper.html
- [3] CodeImage, “CodeImage source browser,” 2026, accessed: Oct. 22, 2026. [Online]. Available: <https://codeimage.io/source/>
- [4] C. Hrițcu, J. Hughes, B. C. Pierce, A. Spector-Zabusky, D. Vytiniotis, A. A. de Amorim, and L. Lampropoulos, “Testing noninterference, quickly,” in *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, 2013. [Online]. Available: <https://catalin-hritcu.github.io/publications/testing-noninterference-icfp2013.pdf>
- [5] J. Villmow, V. Campos, A. Ulges, and U. Schwanecke, “Addressing leakage in self-supervised contextualized code retrieval,” in *Proceedings of the 29th International Conference on Computational Linguistics*. International Committee on Computational Linguistics, 2022, pp. 1006–1013. [Online]. Available: <https://aclanthology.org/2022.coling-1.84/>
- [6] X. Xie, J. W. K. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, “Testing and validating machine learning classifiers by metamorphic testing,” *Journal of Systems and Software*, vol. 84, no. 4, pp. 544–558, 2011. [Online]. Available: <https://www.cs.columbia.edu/wp-content/uploads/sites/7/2016/08/jss2011.pdf>
- [7] F. Drobñjaković, P. Subotić, and C. Urban, “An abstract interpretation-based data leakage static analysis,” in *Theoretical Aspects of Software Engineering*, ser. Lecture Notes in Computer Science, vol. 14777, 2024, pp. 109–126. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-64626-3_7
- [8] Ł. Janisiów, M. Kochańczyk, B. M. Zieliński, and T. Danel, “Enhancing chemical explainability through counterfactual masking,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 40, no. 26, pp. 22 155–22 163, 2026. [Online]. Available: <https://ojs.aaai.org/index.php/AAAI/article/view/39371>
- [9] J. Liu, C. S. Xia, Y. Wang, and L. Zhang, “Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation,” in *Advances in Neural Information Processing Systems*, vol. 36, 2023. [Online]. Available: https://papers.nips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html
- [10] OpenAI, “ChatGPT,” 2026, used for drafting and research-code assistance, Oct. 22–23, 2026. [Online]. Available: <https://chatgpt.com/>