Two Kinds of Unreadable: What Obfuscating Code Actually Costs Language Models

Tony Salomone
Transformer Lab
tony.salomone@lab.cloud

Deep Gandhi
Transformer Lab

Ali Asaria
Transformer Lab

Abstract

A recent report claimed that making code unreadable, by replacing familiar syntax with alien glyphs, improves a frontier model's accuracy on hard programming problems by tens of points: a "decoupling" of capability from human legibility. We do not reproduce any such benefit on five models spanning 7B to frontier scale. Instead, holding each problem fixed and varying only its surface, we show that "code opacity" is two separable things with different costs. Obfuscating the *syntax* (glyph keywords and operators, readable identifiers) produces no statistically resolved drop in a model's ability to *read* the code (paired-bootstrap 95% intervals on the read cost include zero for all three models measured); the large apparent penalty is an output-channel effect, 80–95% of it attributable to being made to *write* the answer back in the opaque form (write-cost intervals exclude zero). Obfuscating the *identifiers* instead (glyph or misleading names) imposes a genuine comprehension cost of roughly 9–16 points that survives on plain, readable output (intervals exclude zero). We also find that the study's own pre-registered "shortcut-suppression" interaction reaches significance when computed the naive way but dissolves once the output-channel artifact is removed, and that the current version of the model family behind the original report refuses the opaque input outright. The practical lesson is a measurement one: score comprehension separately from generation, or an output-format effect will masquerade as a change in reasoning.

1 Introduction

A recent project [1] (which we refer to as the source study) reported a "decoupling" between model capability and human legibility on code: rendering programs in an opaque, glyph-based surface form improved a frontier model's accuracy on hard programming problems by roughly 36 percentage points (57% to 93%), while human readability collapsed. The authors offered no mechanism. The intuition on offer, which the project this paper reports was designed to test, is that opacity *suppresses surface shortcuts*: with familiar tokens removed, a model can no longer pattern-match to a memorized or superficially-cued answer and must compute the result.

We find no such benefit on any model we test. But the more useful result is a decomposition. The source protocol shows a model an opaque program and asks it to answer *in the same opaque form*, then decodes the answer before grading. This bundles two abilities that opacity affects very differently: *comprehension* (can the model understand the transformed code?) and *generation* (can the model emit the transformed surface?). Separating them, with a single change that lets

the model read the opaque input but answer in ordinary Python, splits "code opacity" into two channels:

- **Syntactic opacity** (glyph keywords and operators, readable identifiers): reading is cheap. Across three models the paired read cost is small and not statistically resolved, while the cost of writing the opaque surface is large and significant, accounting for 80–95% of the apparent penalty. This is what the source transform actually does to code.
- **Identifier opacity** (replacing the names with glyphs or with misleading names): a genuine comprehension cost of roughly 9–16 points, present even when the model answers in plain Python.

Neither channel produces the reported reasoning benefit. And the study's pre-registered primary hypothesis, that opacity helps most when identifier signals are misleading, reaches significance when computed on the naive glyph-output cells but dissolves once the output-channel artifact is removed (§5).

The work followed a two-stage pre-registration: a cheap Phase-1 replication gate (does obfuscation help at all?), with a more powered Phase-2 interaction test to run only if Phase 1 passed. Phase 1 did not pass, so the interaction is reported as a secondary, underpowered analysis rather than a confirmatory result.

Contributions.

1. A read/write decomposition of code opacity, with paired-bootstrap intervals, showing that the penalty of *syntactic* obfuscation is an output-channel artifact: no statistically resolved comprehension drop, 80–95% of the penalty in generation, across three models (§5).
2. A distinct second channel: *identifier* obfuscation (glyph or misleading names) imposes a real comprehension cost, roughly 9–16 points, that survives on plain output (intervals exclude zero) (§5).
3. A controlled non-replication of the reported opacity benefit across five models, and the finding that the pre-registered priming-suppression interaction passes naively but does not survive removing the output-channel artifact (§5).
4. A frontier-reproducibility observation: the current version of the source study's model refuses the opaque input at the guardrail level, making the original claim untestable on it today (§5).
5. A measurement protocol, score comprehension separately from generation, for any future study of transformed inputs (§3).

2 Related Work

Surface form and identifiers. Identifier semantics measurably steer code models: counterfactual edits to variable names change predictions [2], and in BERT-class code models the surface form measurably affects predictions [3]. In reasoning more broadly, irrelevant or misleading context degrades accuracy [4, 5], and shortcut learning is a recognized failure mode [6]. These motivate the shortcut-suppression hypothesis but none crosses opacity with surface-signal quality on fixed problems.

Does opacity help or hurt? The evidence is mixed and setup-dependent. Meaningless or unfamiliar tokens have been reported to *improve* reasoning [7, 8], while semantics-preserving garbling *hurts* advanced math [9] and opacity hurts when the surface carries genuinely useful knowledge [10]. Reported reasoning gains also frequently dissolve under standardized re-evaluation [11], which frames our replication as a necessary check.

Degenerate outputs and tokenizer pathology. Heavy perturbation makes models fail to answer rather than answer wrong [5], and character-level or Unicode corruption can trip comprehension and provoke outright refusals in frontier models [12]. This literature directly predicts the output-channel failure and the guardrail refusal we observe, and is why we separate "wrong" from "did not produce a valid answer." Memorization is also procedural rather than verbatim, with models computing correct answers before overriding them [13], which bears on whether an opaque surface could plausibly defeat retrieval at all.

3 Method

The transform. The source study's opacity transform ("CodeSkin" [1]) maps a program's keywords, operators, and delimiters to glyphs while leaving identifiers, numbers, and string literals unchanged; it is a bijection, and outputs are un-skinned before execution. Their released tool ships no static problem set, so an exact rerun is impossible; we perform a *conceptual* replication, reimplementing the transform for Python via a tokenizer shim that is bijective and round-trip-verified (164/164 HumanEval and 257/257 MBPP programs skin then un-skin to a test-passing original).

Two-factor design. We vary two independent axes on each fixed problem. Factor B is *syntax opacity*: B0 (plain Python) versus B1 (glyph-skinned syntax). Factor A is *identifier signal*, of which we use three rungs of a five-rung ladder: A0 (faithful names), A2 (adversarially misleading names), and A4 (glyph identifiers). The source transform corresponds to B1 with faithful identifiers (A0B1). We extend the identifier axis to a maximally opaque cell of *our* ladder, A4B1, where identifiers are also glyphs; whether the source's own endpoint matches A0B1 or A4B1 is itself ambiguous, since its released code leaves identifiers unchanged at every level while its accompanying report describes glyphing them (see §7).

Reading versus writing. Under the source protocol a B1 problem is presented in glyphs and the model is instructed to answer in glyphs ("*glyph-out*"), which is then un-skinned and executed; this measures comprehension and generation jointly. We add one controlled variant, "*plain-out*": the model reads the glyph-encoded input (with the glyph key) but answers in ordinary Python. Plain-out isolates comprehension; glyph-out bundles it with generation, and their difference is the generation cost. We verify the variant at the record level: under plain-out the model's raw output equals its un-skinned output on every B1 record we checked (the open models and Sonnet; no glyph decoding was applied), and prompt-hash checks confirm the intended prompt was sent.

4 Experimental Setup

Data. Problems are drawn from HumanEval, MBPP, and the medium-and-hard split of LiveCodeBench, the last as a post-training-cutoff "novel" pole. After a frozen audit removing 18 harness-unsafe items, 132 problems remain per condition cell. Every experimental input is a deterministic surface transform of a fixed underlying problem.

Models and metric. We test five models: Qwen2.5-Coder-7B and Llama-3.1-8B (open weights, served locally); Claude Sonnet-4.5, GPT-5, and Claude Opus-4.8 (API). Four engage with the

Table 1: Syntactic opacity (A0B1), unit-test pass@1, $n = 132$, paired within problem. Read cost = readable − plain-out; write cost = plain-out − glyph-out; brackets are 95% paired-bootstrap intervals. Read-cost intervals include zero; write-cost intervals exclude it.

Model	Readable	Plain-out	Glyph-out	Read cost [95%]	Write cost [95%]
Qwen2.5-Coder-7B	0.295	0.280	0.106	+0.015 [−.05,+.08]	+0.174 [+.11,+.25]
Llama-3.1-8B	0.182	0.152	0.030	+0.030 [−.03,+.09]	+0.121 [+.06,+.18]
Sonnet-4.5	0.705	0.667	0.015	+0.038 [−.03,+.11]	+0.652 [+.57,+.74]

GPT-5 (provider-default sampling, so reported apart): readable 0.841, plain-out 0.864 at $n = 132$ (reading no worse than readable); on its $n = 25$ cells the glyph-out rate is 0.36, a large write cost consistent with the greedy models but on a small, non-deterministic sample.

opaque input and yield the read/write decomposition; the fifth, Opus-4.8, refuses it (§5). The metric is unit-test pass@1. Decoding is greedy (temperature 0) for the two open models; the API models (Sonnet, GPT-5, Opus) ran at provider-default sampling (the Anthropic and OpenAI endpoints we used do not accept a fixed temperature 0), so their cells are single stochastic draws. The paired bootstrap resamples over problems, not over sampling noise, so it is valid regardless. B1 outputs are un-skinned before the fixed evaluator runs them, matching the source study's decode-then-execute loop. GPT-5's figure cells use its $n = 25$ probe set, so its figure values (readable 0.88, plain-out 0.92) differ from its $n = 132$ table values (0.841, 0.864). On a legend-independent control the greedy models reproduce prompt hashes and per-problem verdicts exactly across runs.

Inference. All contrasts are paired within-problem. Reported intervals are 95% percentile paired bootstraps over problems (5000 resamples), the pairing unit being the problem. We do not fit the pre-registered mixed-effects model; the paired bootstrap is the inferential report.

5 Results

Syntactic opacity: reading is cheap, writing is the tax. Table 1 reports the same faithful-identifier syntactic opacity (A0B1) three ways, with paired-bootstrap intervals on the read cost (readable minus plain-out) and write cost (plain-out minus glyph-out). For all three greedy models the read cost is small and its interval includes zero, so we cannot resolve a comprehension drop at this sample size; the write cost is large and its interval excludes zero. Generation accounts for 80–95% of the total penalty (Figure 1). We stop short of claiming equivalence: at $n = 132$ the read-cost intervals are wide (upper bounds near 8–11 points), so "no resolved difference" is the correct reading, not "provably identical."

Identifier opacity: a real comprehension cost. The picture flips when the *identifiers* are obfuscated rather than the syntax. Reading code with glyph identifiers (A4, plain syntax, plain output) costs Qwen +15.9 points (95% [+9.8, +22.7]) and Llama +10.6 ([+4.5, +17.4]); replacing names with *misleading* ones (A2, plain code) costs +10.6 ([+3.8, +17.4]) and +9.1 ([+3.0, +15.9]). All four intervals exclude zero. So opacity of the names a model reasons with is a genuine comprehension cost, and it is separable from the syntactic surface, which is not.

We do not reproduce the benefit. The pre-registered Phase-1 gate required a pooled readable-to-opaque delta of at least +5 points with an interval excluding zero, directionally positive in at least two of the open models. No model meets it: the naive (glyph-out) contrast is strongly negative, and the comprehension-isolated (plain-out) contrast is small with an interval spanning

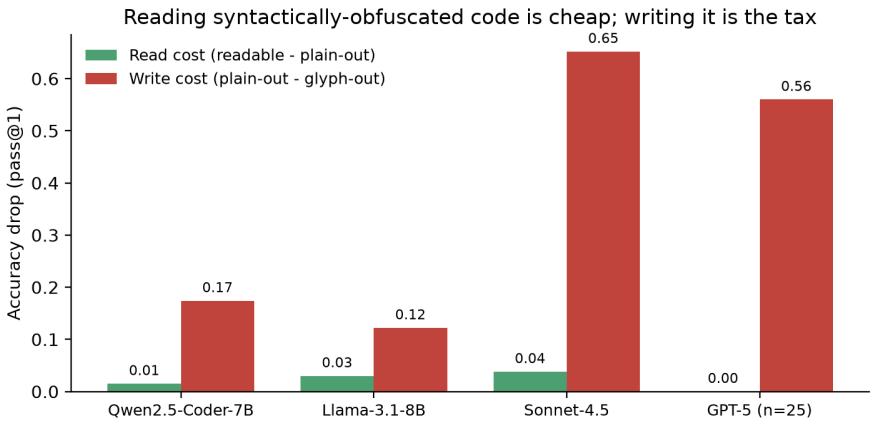

Figure 1: Read cost versus write cost per model, for syntactic opacity. Read cost (readable minus plain-out) is small everywhere; write cost (plain-out minus glyph-out) is large. GPT-5 is shown on its own $n = 25$ cells to avoid mixing sample sizes; its read cost is floored at zero (plain-out slightly exceeded readable).

zero. This is the pre-registered clean-negative outcome. (The gate named three open models; we ran two, so it is assessed on the two available.)

The pre-registered interaction passes naively, then dissolves. The study's pre-registered primary metric was the opacity-by-misleadingness interaction. Computed as originally specified, on the glyph-output cells, it reaches significance: pooled +0.057, 95% [+0.011, +0.106], positive in both pooled open models (Qwen +0.053, Llama +0.061). This clears the interval threshold but not the pre-registered *power*: it was run at $n = 132$ against a specified $n \geq 300$. We disclose it because it is the registered primary, but it is computed on glyph-output accuracies, which the decomposition above shows are dominated by the output-channel artifact and floored near zero; the "interaction" is consistent with a floor effect of the collapsed cells. Recomputed on the comprehension-isolated plain-out cells it is small and sign-inconsistent across models. We therefore do not read it as evidence of shortcut suppression.

Frontier reproducibility: Opus refuses the input. The source study's headline was on Claude Opus. The current Opus-4.8 returns `finish_reason: refusal` on all 12 glyph-encoded inputs we sampled (Wilson 95% lower bound on the refusal rate 0.74), even for the mildest A0B1 form and even when asked to answer in plain Python, while solving the readable baseline on all 12. The refusal is triggered by the opaque input itself. On our sample the current guardrail deterministically refuses the input the method depends on, so the original claim is not testable on that model today.

6 Discussion

Two things travel under the name "unreadable," and they behave oppositely. Scrambling the syntax is cheap for a model to see through; the dramatic penalty a naive setup records is an *output-channel* effect, the cost of emitting an unfamiliar surface (or of a correct answer failing an un-skin/format gate), not a change in reasoning. Scrambling the names a model reasons with is not cheap: it carries a real, significant comprehension cost. Only the first is a measurement artifact.

This also explains the source result's fragility. A capable model that can both read and write the opaque surface shows no benefit; a model that cannot emit it shows a large artifactual penalty; and a model whose guardrails reject the surface cannot be measured at all. The apparent shortcut-suppression signal in the naive interaction is best explained the same way: glyph-output floors the cells, and a floored cell that started lower falls less far, which reads as a positive interaction without any reasoning being suppressed.

7 Limitations

An upstream paper/code discrepancy. The source study's released code and its accompanying report disagree on whether identifiers are obfuscated: the shipped transform leaves identifiers unchanged at every level, while the report describes a ladder that glyphs them. We adapted the code, so our A0B1 matches the shipped transform; our factorial spans both readings (A0B1 and A4B1), but which the source actually ran for its headline number is undetermined from the public artifacts. This is why we do not attribute A4B1 to the source as its endpoint.

Construct and external validity. This is a conceptual replication: the source ships no fixed problem set, so we reimplemented its transform for Python rather than rerunning its pipeline. Our glyph vocabulary and tokenization differ from the original, and our problems are public benchmarks. We could not run the source's own model (current Opus refuses the input), which bounds how directly we speak to the original number.

Output-channel vs. format-gate. Glyph-output is un-skinned before execution, so a logically-correct answer with a malformed glyph surface fails the format gate and scores zero. We did not separately re-score glyph-output leniently to split "malformed but correct" from "genuinely wrong." We therefore attribute the syntactic penalty to the *output channel* (emission or decode gate) rather than to model generation specifically.

Mechanism not established. The source's hypothesized benefit rests on suppressing priming and memorization shortcuts; we did not establish that our readable baselines actually ride those shortcuts (the contamination-controlled set was gated behind Phase 1 and never built). A null is therefore partly expected if the shortcuts are not present to suppress; our result bounds the effect's generality rather than adjudicating the mechanism.

Power and provenance. At $n = 132$ the read-cost intervals are wide, so "no resolved difference" is not equivalence. The pre-registered interaction was run only at $n = 132$ against a specified $n \geq 300$. GPT-5 used provider-default sampling and its glyph-output cell is $n = 25$; it is reported apart. The Opus finding is a 12-item sample, framed as such.

8 Availability

Code and transformed data are available from the authors on request. The transform is adapted from the source study's released CodeSkin tool (AGPL-3.0); our reimplementation, condition definitions, run artifacts, and evaluation recipes are retained in the project record.

9 Conclusion and Future Work

Making code unreadable does not make models reason better. Separating reading from writing shows that syntactic obfuscation is an output-channel effect, while name obfuscation is a genuine but separate comprehension cost, and neither buys a reasoning gain. The reusable lesson is to

score comprehension apart from generation, and to report answer-rate (refusals and degenerate outputs) apart from correctness. Natural next steps are a lenient re-score that splits malformed-but-correct from wrong on the output channel, a properly powered ($n \geq 300$) test of the name-comprehension cost, and a broader survey of which frontier guardrails reject transformed inputs, since that increasingly determines what such studies can measure at all.

References

[1] elder_plinius. Lingua Ex Machina: A Procedural Xenolinguistics Engine Reveals Zero-Shot Language Acquisition, Human-Unreadable Coding Systems, and Exploitable Covert Channels in Frontier AI. https://github.com/elder-plinius/GLOSSOPETRAE, 2026. Self-published technical report and software.

[2] Ashish Hooda, Mihai Christodorescu, Miltiadis Allamanis, Aaron Wilson, Kassem Fawaz, and Somesh Jha. Do Large Code Models Understand Programming Concepts? Counterfactual Analysis for Code Predicates. 2024. URL https://arxiv.org/abs/2402.05980.

[3] Toufique Ahmed, Dian Yu, Chengxuan Huang, Cathy Wang, Prem Devanbu, and Kenji Sagae. Towards Understanding What Code Language Models Learned. 2023. URL https://arxiv.org/abs/2306.11943.

[4] Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models. 2024. URL https://arxiv.org/abs/2410.05229.

[5] Giannis Chatziveroglou, Richard Yun, and Maura Kelleher. Exploring LLM Reasoning Through Controlled Prompt Variations. 2025. URL https://arxiv.org/abs/2504.02111.

[6] Geetanjali Bihani and Julia Rayz. Learning Shortcuts: On the Misleading Promise of NLU in Language Models. 2024. URL https://arxiv.org/abs/2401.09615.

[7] Zeru Shi, Yingjia Wan, Zhenting Wang, Qifan Wang, Fan Yang, Elisa Kreiss, and Ruixiang Tang. Meaningless Tokens, Meaningful Gains: How Activation Shifts Enhance LLM Reasoning. 2025. URL https://arxiv.org/abs/2510.01032.

[8] Xinyi Wang, Antonis Antoniades, Yanai Elazar, Alfonso Amayuelas, Alon Albalak, Kexun Zhang, and William Yang Wang. Generalization v.s. Memorization: Tracing Language Models' Capabilities Back to Pretraining Data. 2024. URL https://arxiv.org/abs/2407.14985.

[9] Yuren Hao, Xiang Wan, and Ce Zhang. An Investigation of Robustness of LLMs in Mathematical Reasoning: Benchmarking with Mathematically-Equivalent Transformation. 2025. URL https://arxiv.org/abs/2508.08833.

[10] Yu-Fei Shih, Zheng-Lin Lin, and Shu-Kai Hsieh. Reasoning Over the Glyphs: Evaluation of LLM's Decipherment of Rare Scripts. 2025. URL https://arxiv.org/abs/2501.17785.

[11] Andreas Hochlehnert, Hardik Bhatnagar, Vishaal Udandarao, Samuel Albanie, Ameya Prabhu, and Matthias Bethge. A Sober Look at Progress in Language Model Reasoning: Pitfalls and Paths to Reproducibility. 2025. URL https://arxiv.org/abs/2504.07086.

[12] Jiawen Wen, Bangshuo Zhu, and Huaming Chen. What You See Is Not Always What You Get: Evaluating GPT's Comprehension of Source Code. 2024. URL https://arxiv.org/abs/2412.08098.

[13] Yupei Du, Philipp Mondorf, Silvia Casola, Yuekun Yao, Robert Litschko, and Barbara Plank. Reason to Rote: Rethinking Memorization in Reasoning. 2025. URL https://arxiv.org/abs/2507.04782.