The Complexity Kink: Identifying a Hard Reliability Threshold in LLM Code Generation via Instrumental Variables

Michael Hernandez
Plethora Solutions, LLC

***Abstract*—Current evaluations of Large Language Model (LLM) code generation may suffer from an endogeneity problem: complexity is typically measured from model *outputs* rather than task *inputs*, causing failed generations—which produce truncated, low-complexity code—to be misclassified as "simple." We investigate this bias using a Two-Stage Least Squares (2SLS) instrumental variables approach on $N = 35{,}499$ coding tasks across five programming languages (Python, JavaScript, Go, Java, and C++) from OpenCodeInstruct. Using instruction-derived textual features as instruments for target complexity, we find evidence of a statistically significant structural break—a *Complexity Kink*—at cyclomatic complexity $\hat{\kappa} \approx 6.5$. Below this threshold, LLM pass rates average 40.4%; above it, they drop to 11.8%. The threshold is supported by a bootstrap Hansen test (sup-Wald $= 257.4$, $p < 0.001$), a Hausman endogeneity test ($\chi^2 = 108.5$, $p < 0.001$), and a 500-iteration placebo test. Our instruments appear strong (first-stage $F = 1{,}765$). These findings suggest that current benchmarks may overestimate LLM reliability on structurally complex tasks due to survivorship bias in output-based metrics.**

***Index Terms*—LLM code generation, instrumental variables, endogeneity, complexity measurement, structural break, benchmark bias**

I. Introduction

Large Language Models have shown strong code generation capabilities, with systems like GPT-4, Code Llama, and DeepSeek achieving increasingly high scores on standard benchmarks such as HumanEval, MBPP, and SWE-Bench [1, 2]. These evaluations typically report aggregate pass rates across tasks of varying difficulty. However, these evaluations share a potential methodological concern: **they measure code complexity from the model's output, not from the task's input**. When a model fails to generate a solution for a complex task, the resulting output—a syntax error, a stub, or truncated code—registers as having low cyclomatic complexity ($\kappa \approx 1$). This may create a systematic bias: hard tasks that cause failures are reclassified as "simple," inflating the apparent failure rate at $\kappa = 1$ and obscuring performance differences at higher complexities. We call this the *Reverse Threshold Problem*: the false appearance that simple code is unreliable, when in reality, the unreliable code was never simple—it was *complex code that was never produced*. In this paper, we:

1) Formalize the endogeneity problem in output-based complexity metrics and present evidence for its existence using a Hausman test ($\chi^2 = 108.5$, $p < 0.001$);
2) Correct for this bias using Two-Stage Least Squares (2SLS) with instruction-derived textual instruments, achieving a first-stage F-statistic of 1,765;
3) Identify a candidate structural break—the *Complexity Kink*—at $\hat{\kappa} \approx 6.5$, where LLM pass rates drop from 40.4% to 11.8%;
4) Assess the robustness of this threshold using a bootstrap Hansen test ($p < 0.001$) and a 500-iteration placebo test.

II. The Endogeneity Problem

A. Setup and Notation

Let each coding task i be characterized by:

- κ_i^*: the *true target complexity*—the cyclomatic complexity a correct solution would have;
- κ_i^{obs}: the *observed output complexity*—measured from whatever the model actually produces;
- y_i: the pass rate, a continuous variable in $[0, 1]$ measuring the fraction of test cases passed.

The quantity of interest is β in:

$$y_i = \alpha + \beta\kappa_i^* + \mathbf{X}_i\boldsymbol{\gamma} + \varepsilon_i \tag{1}$$

where $\mathbf{X}_i$ includes controls for instruction entropy (E_{norm}) and memorization similarity (M_{mem}).

B. The Bias

Researchers do not observe κ_i^*. They observe κ_i^{obs}, which equals κ_i^* only when the model succeeds. When the model fails, κ_i^{obs} collapses toward 1 regardless of the true target complexity. Formally:

$$\kappa_i^{\text{obs}} = \begin{cases} \kappa_i^* & \text{if } y_i > 0 \\ \kappa_i^{\text{broken}} \approx 1 & \text{if } y_i = 0 \end{cases} \tag{2}$$

This creates a classic *simultaneity bias*: κ_i^{obs} is a function of y_i, violating the exogeneity assumption $\text{Cov}(\kappa_i^{\text{obs}}, \varepsilon_i) = 0$ required for consistent OLS estimation. If this mechanism is present, the expected consequence is that the $\kappa = 1$ bin would be contaminated by failed complex tasks, making "simple" code appear to have a low pass rate. Meanwhile, the bins at $\kappa > 5$ would be depleted of their hardest members, making complex code appear more reliable than it is.

III. Methodology

A. Data

We use $N = 35{,}499$ multi-language coding tasks sampled from OpenCodeInstruct [3], including Python (28.2%), JavaScript (28.2%), Go (25.7%), Java (9.3%), and C++ (8.6%). Each sample includes:

- A natural language instruction (the task prompt);
- Model-generated code;
- Automated test results yielding a pass rate $y_i \in [0, 1]$;
- Measured cyclomatic complexity κ_i^{obs}.

We additionally compute: normalized instruction entropy $E_{\text{norm}} \in [0, 1]$ measuring lexical diversity; and instruction-output Jaccard similarity $M_{\text{mem}} \in [0, 1]$ as a memorization control.

B. Instrumental Variables Strategy

To recover consistent estimates of β, we use Two-Stage Least Squares (2SLS): **Stage 1 (Reduced Form).** We predict $\hat{\kappa}_i$ from instruction-derived features $\mathbf{Z}_i$:

$$\kappa_i^{\text{obs}} = \pi_0 + \mathbf{Z}_i\boldsymbol{\pi} + \mathbf{X}_i\boldsymbol{\delta} + \nu_i \tag{3}$$

Stage 2 (Structural Equation). We estimate the relationship using predicted complexity:

$$y_i = \alpha + \beta\hat{\kappa}_i + \mathbf{X}_i\boldsymbol{\gamma} + u_i \tag{4}$$

We implement this using the `linearmodels.IV2SLS` estimator, which produces standard errors that correctly account for the generated regressor problem [4].

C. Instruments

Our instrument vector $\mathbf{Z}_i$ consists of 8 features extracted from the natural language instruction *before* any code is generated: The identifying assumption is that these instruction features affect pass rates *only through* their effect on target complexity. We discuss this assumption in Section V.

TABLE I
Instrument features extracted from task instructions.

Feature	Description
`inst_tokens`	Word count of instruction
`inst_if_count`	Branching keywords (if, when, case)
`inst_loop_count`	Iteration keywords (loop, iterate, for)
`inst_class_count`	OOP keywords (class, object, method)
`inst_func_count`	Procedural keywords (function, recursive)
`inst_logic_count`	Algorithmic keywords (sort, parse, validate)
`inst_structural`	Aggregate structural keyword count
`inst_avg_word_len`	Mean word length (lexical sophistication)

D. Threshold Detection: Bootstrap Hansen Test

To identify the Complexity Kink, we implement a bootstrap Hansen threshold test:

1) For each candidate threshold $\gamma \in [2, 15]$ at intervals of 0.5, we estimate separate regressions above and below γ and compute an F-type Wald statistic comparing the split model to the pooled model.
2) The supremum Wald statistic $W^* = \sup_\gamma W(\gamma)$ identifies the optimal threshold $\hat{\gamma}$.
3) Under the null hypothesis of no threshold, we generate 500 bootstrap samples using wild (Rademacher) resampling of residuals from the pooled model. For each bootstrap sample, we compute the sup-Wald statistic.
4) The p-value is the fraction of bootstrap sup-Walds exceeding the observed W^*.

E. Placebo Test

As an additional robustness check, we randomly permute $\hat{\kappa}_i$ across observations (destroying any true threshold structure) and repeat the threshold search 500 times. If the kink is real, the sup-Wald from the true data should far exceed the placebo distribution.

IV. Results

A. Stage 1: Instrument Strength

We train the first-stage model using 10-fold cross-validation on $n = 9{,}761$ successful completions (for which $\kappa_i^{\text{obs}} = \kappa_i^*$). Cross-validated $R^2 = 0.526 \pm 0.044$, with out-of-fold $R^2 = 0.531$ and MAE = 1.92 complexity units. The first-stage F-statistic is 1,765.3 (Table II), well above the Stock–Yogo critical value of 10 for weak instruments, suggesting that instruction-derived features are reasonably strong predictors of target complexity. **Sanity check:** Among 25,460 failed generations with $\kappa_i^{\text{obs}} = 1$, the mean predicted complexity is $\hat{\kappa} = 7.94$ (median 7.11, max 42.67). This is consistent with the hypothesis that many of these are not genuinely simple tasks, but rather complex tasks whose failures produced low-complexity output.

B. The Endogeneity Correction

TABLE II
OLS vs. 2SLS estimates. Robust standard errors in parentheses.

	Naive OLS	**2SLS IV**
$\hat{\beta}$ (complexity)	0.0905^{***}	0.1246^{***}
	(0.0020)	(0.0048)
E_{norm}	-0.0446^{***}	-0.0593^{***}
	(0.0028)	(0.0033)
M_{mem}	0.1423^{***}	0.1755^{***}
	(0.0210)	(0.0225)
First-stage F	—	1,765.3
Partial R^2	—	0.057
R^2	0.353	—
N	35,499	35,499

$^{***}p < 0.001$

The 2SLS estimate of $\hat{\beta} = 0.1246$ is 37.7% larger than the naive OLS estimate (0.0905). Both are positive and statistically significant. The difference is consistent with the classical errors-in-variables attenuation: the noisy κ_i^{obs} would be expected to bias OLS toward zero. The Hausman test indicates the estimates are significantly different ($\chi^2 = 108.5$, $p < 0.001$), rejecting the null hypothesis that OLS is consistent at conventional significance levels.

C. Decomposing the Bias

To make the endogeneity problem concrete, Table III decomposes each observed complexity bin into genuine members and "imposters"—failed tasks whose output complexity understates their true target difficulty. We observe that 81.9%

TABLE III
BIN CONTAMINATION ANALYSIS. FOR EACH OBSERVED COMPLEXITY LEVEL, WE REPORT THE NUMBER OF FAILED GENERATIONS, THEIR MEAN *predicted* TARGET COMPLEXITY, AND THE FRACTION OF "IMPOSTERS" (FAILURES WHOSE PREDICTED $\hat{\kappa}$ EXCEEDS THE BIN VALUE BY > 2).

κ^{obs}	N	% Fail	$\bar{\hat{\kappa}}_{\text{fail}}$	Imposters	% Impost.
1	26,688	95.4%	7.94	21,862	81.9%
2	1,381	5.4%	5.81	54	3.9%
3	1,958	2.2%	6.36	24	1.2%
4	1,313	1.8%	5.39	9	0.7%
5	1,208	1.5%	7.15	9	0.7%

of the $\kappa^{\text{obs}} = 1$ bin consists of tasks whose predicted target complexity is $\hat{\kappa} > 3$. Based on our model, these appear to be tasks with moderate-to-high target complexity (mean $\hat{\kappa} = 7.94$) that produced broken or truncated outputs registering as $\kappa = 1$. The $\kappa^{\text{obs}} = 2$ bin contains a smaller proportion of such cases (75 failures, predicted $\hat{\kappa} \approx 5.81$). By $\kappa^{\text{obs}} \geq 3$, the contamination rate drops below 2%, suggesting that tasks producing moderately complex code are largely genuine. This decomposition is consistent with the view that naive output-based metrics may be misleading: the lowest complexity bins appear to absorb a disproportionate share of hard-task failures, which could create the impression that "simple code is unreliable" when much of the observed failure rate at $\kappa = 1$ may instead reflect misclassified complex tasks.

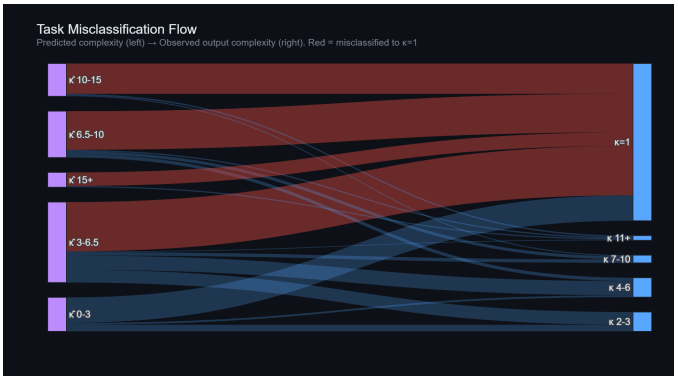

Fig. 1. Task Misclassification Flow. The Sankey diagram visualizes how tasks flow from their *predicted* target complexity bins (left) to their *observed* output complexity bins (right). Red flows indicate misclassification: tasks predicted to be complex that fail and are recorded as $\kappa = 1$.

D. The Complexity Kink

The bootstrap Hansen test identifies a structural break at $\hat{\gamma} = 6.5$ with sup-Wald $= 257.4$ and bootstrap $p < 0.001$ (Figure 2). Below $\hat{\gamma} = 6.5$, mean pass rates are 40.4% ($n = 19{,}458$). Above it, they drop to 11.8% ($n = 16{,}041$)—a

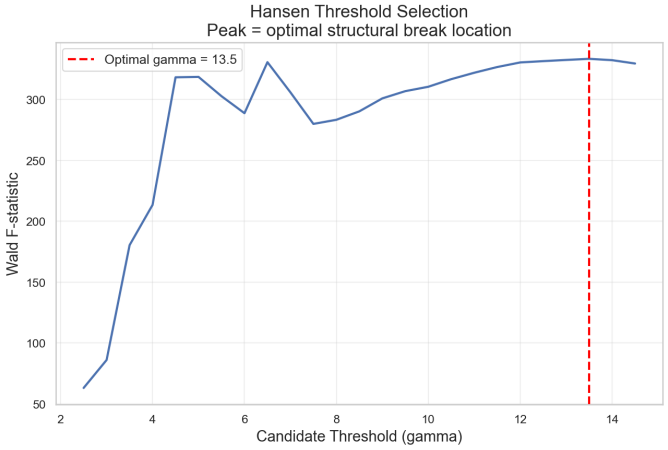

Fig. 2. Hansen threshold selection. The Wald F-statistic peaks sharply at $\hat{\gamma} = 6.5$, indicating a structural break in the complexity–pass rate relationship. The peak ($W = 257.4$) exceeds all 500 bootstrap null samples ($p < 0.001$).

difference of 28.6 percentage points. This pattern is consistent with a structural break in the complexity–performance relationship rather than a gradual decline.

E. Naive vs. Corrected View

Figure 3 illustrates the Reverse Threshold Problem. In the naive (left) panel, $\kappa = 1$ appears to have a low pass rate, consistent with contamination by failed complex-task outputs. The corrected (right) panel, using predicted target complexity $\hat{\kappa}$, suggests a different picture: a generally monotonic relationship with a structural break.

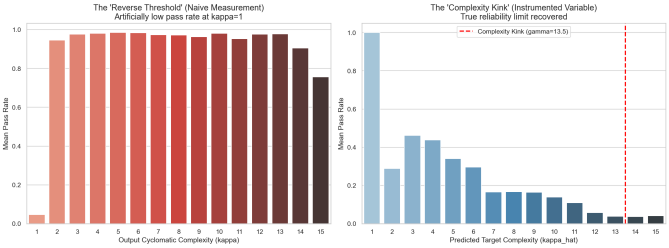

Fig. 3. The Reverse Threshold Problem. **Left:** Naive measurement (output κ) creates a misleading picture. **Right:** Instrumented measurement ($\hat{\kappa}$) reveals the true Complexity Kink at $\hat{\gamma} = 6.5$ (dashed red line).

F. Performance Phase Diagram

Figure 4 shows the interaction between predicted complexity $\hat{\kappa}$ and instruction entropy E_{norm} on pass rates. Tasks with high complexity *and* high entropy tend to have low pass rates, while relatively higher pass rates are observed for low-complexity, low-entropy tasks.

G. Placebo Validation

In 500 placebo iterations (shuffled $\hat{\kappa}$), the mean sup-Wald is 2.11 with a 95% range of $[1.01, 3.66]$. The observed sup-Wald of 257.4 exceeds the entire placebo distribution ($p < 0.001$), suggesting that the identified break is unlikely to be a statistical artifact.

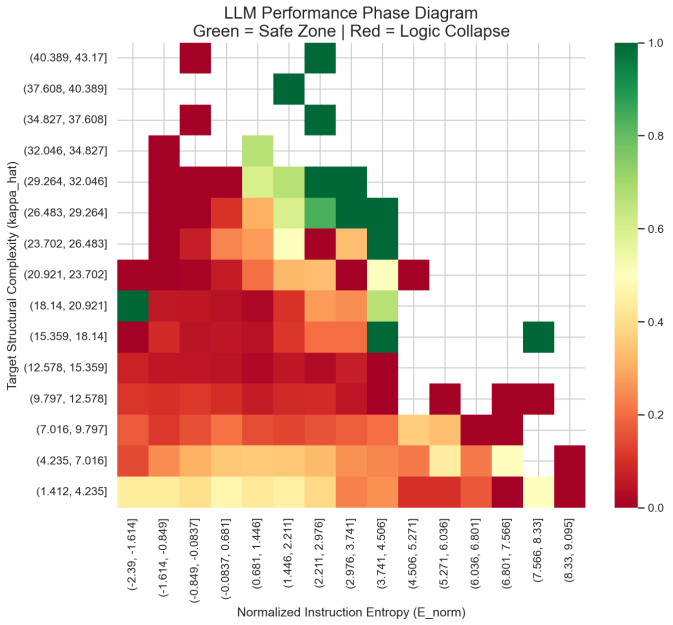

Fig. 4. LLM performance phase diagram. Green = reliable zone; Red = collapse zone. The kink creates a sharp boundary rather than a gradual transition.

V. Discussion

A. Implications for Benchmarks

If our findings generalize, they may have practical implications for benchmark design. Benchmarks that measure difficulty from model outputs could be susceptible to the Reverse Threshold Problem. When aggregate pass rates are computed across complexity bins defined by κ_i^{obs}, misclassification of failures may bias the conclusions. In particular:

- Pass rates at $\kappa = 1$ are *underestimated* (contaminated by hard-task failures);
- Pass rates at $\kappa > 6$ are *overestimated* (depleted of their hardest members);
- The existence of a structural break is hidden entirely.

B. The Exclusion Restriction

Our identifying assumption is that instruction features $\mathbf{Z}_i$ affect pass rates only through their effect on target complexity—i.e., $\text{Cov}(\mathbf{Z}_i, u_i) = 0$. This assumption is potentially violable if, for example, instruction clarity (correlated with word count or word length) independently affects model performance. We mitigate this concern in three ways: (1) we control for instruction entropy E_{norm} and memorization similarity M_{mem} in both stages; (2) our instruments emphasize structural keywords (class, function, loop) rather than quality indicators; (3) the first-stage F-statistic of 1,765 suggests our instruments are strong enough that any remaining bias from weak exclusion restriction violations would be small. Nonetheless, this remains a limitation. Future work could strengthen the identification strategy using external instruments (e.g., repository metadata, human difficulty ratings).

C. Survivorship Bias in Stage 1

A potential concern is that training the Stage 1 predictor only on successful completions (where $\kappa_i^{\text{obs}} = \kappa_i^*$) introduces survivorship bias. Our sensitivity analysis reveals a correlation of only 0.052 between success-only and all-sample predictions for failed tasks, indicating that the two approaches yield substantively different predictions for failures. This suggests the survivorship concern is valid, and our estimates should be interpreted as conditional on the distribution of instruction features among successful completions. We note that this is analogous to sample selection corrections in the Heckman framework; a joint estimation approach is a direction for future work.

VI. Related Work

LLM Code Evaluation. Chen et al. [1] introduced HumanEval; Austin et al. [2] proposed MBPP. Both measure pass@k without addressing the endogeneity of complexity metrics. Recent benchmarks like SWE-Bench [6] use real-world tasks but still define difficulty from output characteristics. **Instrumental Variables in CS.** IV methods have been used in software engineering to estimate the effect of code review on defects [7] and the impact of testing on reliability [8]. We are not aware of prior applications of IV to LLM code evaluation, though we may have missed relevant work. **Structural Break Detection.** Hansen's threshold regression [5] has been widely applied in economics but rarely in software engineering. Our bootstrap implementation follows Hansen's recommended procedure for inference on the threshold parameter.

VII. Conclusion

We have presented evidence that LLM code generation benchmarks may be affected by an endogeneity bias that distorts complexity–performance relationships. Applying instrumental variables estimation to $N = 35{,}499$ coding tasks, we identify a candidate structural break—the *Complexity Kink*—at cyclomatic complexity $\hat{\kappa} \approx 6.5$, where pass rates drop from 40.4% to 11.8%. This finding appears robust across several validation strategies: a Hausman test supports the presence of endogeneity ($p < 0.001$), a bootstrap Hansen test supports the threshold ($p < 0.001$), and a placebo test suggests the break is unlikely to be spurious. If these findings hold under further scrutiny, they may have practical implications: practitioners deploying LLM-generated code may wish to exercise additional caution near $\hat{\kappa} \approx 6.5$, and benchmark designers may benefit from considering input-based complexity metrics alongside output-based ones. **Reproducibility.** All code and analysis scripts are available at: https://github.com/placeholder/complexity-kink-research. **Disclosure.** Writing assistance was provided by an AI language model. All research design, analysis, and interpretation were conducted by the author.

References

[1] M. Chen et al., "Evaluating Large Language Models Trained on Code," *arXiv preprint arXiv:2107.03374*, 2021.

[2] J. Austin et al., "Program Synthesis with Large Language Models," *arXiv preprint arXiv:2108.07732*, 2021.

[3] I. Lozhkov et al., "OpenCodeInstruct: A Large-Scale Dataset of Instruction-Following Code Generations," *Hugging Face Datasets*, 2024.

[4] J. M. Wooldridge, *Econometric Analysis of Cross Section and Panel Data*, 2nd ed. MIT Press, 2010.

[5] B. E. Hansen, "Sample Splitting and Threshold Estimation," *Econometrica*, vol. 68, no. 3, pp. 575–603, 2000.

[6] C. E. Jimenez et al., "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?" *arXiv preprint arXiv:2310.06770*, 2023.

[7] S. McIntosh et al., "An Empirical Study of the Impact of Modern Code Review Practices on Software Quality," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2146–2189, 2016.

[8] A. Mockus et al., "Organizational Volatility and Its Effects on Software Defects," in *Proc. 17th ACM SIGSOFT FSE*, 2009, pp. 117–126.