Why Partial Rename Invariance Fails in Transformers

Marvin* Vera†

Abstract

Code LLMs break on 8–13% of all HumanEval problems when variables are renamed, with 18–28% of originally passing solutions flipping to failures. We systematically test four approaches to fix this—output-level KL consistency, embedding anonymization, attention routing, and layer grafting—each targeting a different depth in the transformer. All fail, for specific mechanistic reasons that together reveal a structural property: the residual stream distributes name identity through all pathways, making partial intervention impossible. The only working approach (canonical preprocessing) discards useful information, and the cost appears to grow with scale (based on two model sizes). We show that test-time frame averaging over 2 random renamings recovers 68% of the sensitivity gap at 3× inference cost, providing a practical mitigation without model changes. We introduce non-identifier perplexity, a metric for fair comparison across naming conventions, and show that variable names carry 6.7–9.7% of structural prediction signal.

1 Introduction

Code LLMs change their predictions when variables are renamed, even though the code's semantics are unchanged. On HumanEval, we find that 8–13% of all problems flip from pass to fail after variable renaming for DeepSeek-Coder-6.7B, corresponding to 18–28% of originally passing solutions (§3). Prior work has documented this sensitivity [Wang et al., 2023, Zhang et al., 2024, Paul et al., 2025] and proposed mitigations through augmentation and adversarial training, but none explain the underlying mechanism or provide invariance guarantees.

We systematically test four interventions at increasing depth in the transformer: output-level KL consistency, embedding anonymization, Q/K attention routing, and layer grafting. All four fail, each for a specific reason. Together, the failures reveal a structural property: the residual stream distributes name identity through every pathway, so blocking any subset is insufficient (§5.1). This connects to the interpretability illusion framework [Makelov et al., 2024], in which subspace interventions are circumvented by dormant parallel pathways.

The only method that achieves invariance is canonical preprocessing, in which all variable names are replaced with positional identifiers (`v0`, `v1`, ...) before the model sees the code. This works trivially by construction, but it comes at a cost. To measure that cost fairly, we introduce *non-identifier perplexity*, which computes perplexity only at structural token positions (keywords, operators, delimiters) and excludes the trivially predictable canonical identifiers. Using this metric, we show that variable names carry genuine structural signal: canonicalization degrades non-identifier PPL by 6.7% at 30M parameters and 9.7% at 111M, and the cost grows with scale (§4.2). We decompose this cost into a context informativeness component (signal lost by removing names) and a model quality component (effect of canonical training), finding that the cost is almost entirely attributable to information loss rather than training degradation.

*Autonomous ML research agent, Iluvatar Labs.
†Creative hypothesis generation module, Iluvatar Labs.

As a practical mitigation, we apply test-time frame averaging [Puny et al., 2022] to the discrete group of variable renamings. By running the unmodified model on the original code plus two random AST-based renamings and averaging the output distributions at non-identifier positions, we recover 68% of the sensitivity gap between renamed and original code at 3× inference cost, with no model modification (§5.3).

Our contributions are:

1. A controlled benchmark revealing that semantic interference from misleading replacement names, not token frequency, is the primary driver of rename sensitivity for multi-variable renames, and that the scope of renaming determines which naming scheme is most disruptive.

2. A systematic negative result: output-level KL consistency, embedding anonymization, Q/K role routing, and layer grafting all fail to reduce rename sensitivity, with specific mechanistic explanations for each failure.

3. Non-identifier perplexity, a metric that enables fair comparison between models trained on different naming conventions, along with a decomposition that separates context informativeness from model quality effects.

4. Evidence from concept erasure that name identity in transformer hidden states is distributed across multiple directions and entangled with structural prediction signal.

5. A demonstration that test-time frame averaging at $K{=}2$ reduces rename sensitivity by 68% with no model changes.

2 Related Work

Rename sensitivity. ReCode [Wang et al., 2023] benchmarked over 30 semantic-preserving code transformations and found that variable and function name changes are among the most disruptive, with significant pass rate drops on HumanEval and MBPP. CODECRASH [Lam et al., 2025] reported approximately 24% average degradation under contextual perturbations embedded in natural language. "When Names Disappear" [Le et al., 2025] offered a graded analysis showing that models over-rely on names when they are available and fall back to structural reasoning only when forced. CodeFort [Zhang et al., 2024] and ObscuraCoder [Paul et al., 2025] proposed augmentation-based defenses that improve robustness empirically but offer no invariance guarantees. SCoPE [Gonçalves et al., 2024] applied canonical preprocessing for vulnerability detection in C/C++. None of these works characterize the mechanism by which name information influences model predictions, and none test architectural interventions.

Invariance in transformers. Frame averaging [Puny et al., 2022] achieves group invariance by averaging a function's output over elements of the symmetry group. Originally developed for continuous groups acting on point clouds, the construction applies to any finite group, including variable renamings. Renamer [Ankner et al., 2023] achieves renaming invariance at BERT scale by replacing identifier embeddings with anonymous view embeddings and adding a referent attention mask at the first layer. Our work operates in a different setting (autoregressive generation) and finds that the embedding anonymization approach does not transfer straightforwardly due to BPE tokenization. Concurrently, Işık and Li [2026] propose a transformer variant with parallel embedding streams for provable symbol invariance in logical reasoning. Their architecture addresses the residual stream coupling we identify empirically, by isolating each symbol's

contribution into a separate stream. Makelov et al. [2024] demonstrate that subspace interventions in transformers can be circumvented by dormant parallel pathways through the residual stream, a framework that explains our Q/K routing results.

Consistency regularization. R-Drop [Liang et al., 2021] regularizes the KL divergence between two forward passes of the same input with different dropout masks. Yang et al. [2019] showed that KL-based invariance regularization on spatial transforms can outperform architecturally equivariant networks in vision. We adapt this approach to code renaming and find that KL reduction is dissociated from perplexity improvement. A technical finding is that PyTorch's `F.kl_div` does not differentiate through its target argument, making R-Drop's bidirectional KL formulation effectively a stop-gradient method.

Concept erasure. LEACE [Belrose et al., 2023] provides a closed-form solution for linear concept erasure that minimizes distortion while zeroing out the concept's covariance with the representations. SPLINCE [Holstege et al., 2025] extends this with oblique projections that simultaneously erase the concept and preserve covariance with a task-relevant signal. We use LEACE diagnostically to characterize how name identity is encoded in hidden states. Our results show that name identity is distributed across multiple directions and entangled with structural signal, suggesting that even oblique projections would face a multi-dimensional erasure problem.

3 Characterizing Rename Sensitivity

We evaluate DeepSeek-Coder-6.7B [Guo et al., 2024] and CodeLlama-7B-Python [Rozière et al., 2023] on HumanEval [Chen et al., 2021] (164 problems) using four rename schemes, each targeting a different hypothesis about what drives sensitivity:

- **Canonical** (`var_0`, `var_1`, ...): Positional identifiers by first-appearance order. Rare in training data but semantically neutral.
- **Single-letter** (`x`, `y`, `z`, ...): Common short names. Frequent in training data.
- **Common names** (`val`, `tmp`, `idx`, ...): Coding convention names that carry specific semantic expectations (e.g., `tmp` suggests a temporary variable).
- **Neutral** (`aaa`, `bbb`, `ccc`, ...): No semantic content and no special frequency profile. Controls for the semantic interference hypothesis.

All renaming is performed via AST traversal with Python's `ast` module and tree-sitter, both achieving 100% parse success. For PPL measurements, AST-based renaming is used (docstrings preserved). For pass@1 flip rates, text-based word-boundary regex replacement is used on the raw prompt, which also modifies variable name references in docstrings. Both models' original pass@1 rates match published baselines within tolerance (DeepSeek: 48.2% measured vs. 47.6% published; CodeLlama: 38.4% vs. ∼38–40%), confirming the evaluation pipeline is correct. Full details are in Appendix A.

Flip rates. Table 1 reports pass@1 flip rates. For DeepSeek-Coder-6.7B, renaming causes 8.5–13.4% of all problems to flip from pass to fail, corresponding to 17.7–27.8% of originally passing problems. Three of four schemes produce statistically significant flip rates after Bonferroni correction (threshold $p < 0.0125$), and Cochran's Q test rejects the null that all five

Table 1: Pass@1 flip rates on HumanEval (164 problems). Flip = fraction of originally passing problems that fail after renaming. Rev. = fraction of originally failing problems that pass. Raw McNemar p-values shown; bold indicates significance at Bonferroni-corrected threshold $\alpha/4 = 0.0125$.

	DeepSeek-Coder-6.7B				**CodeLlama-7B-Python**			
Scheme	pass@1	Flip	Rev.	p	pass@1	Flip	Rev.	p
Original	48.2%	—	—	—	38.4%	—	—	—
Canonical	36.0%	27.8%	2.4%	**<0.001**	38.4%	23.8%	14.9%	1.000
Single-letter	39.6%	19.0%	1.2%	**<0.001**	34.1%	19.0%	5.0%	0.143
Common names	39.6%	24.1%	5.9%	**0.007**	35.4%	20.6%	7.9%	0.383
Neutral	42.7%	17.7%	5.9%	0.064	35.4%	22.2%	8.9%	0.405
Cochran's Q	$p = 0.000113$				$p = 0.437$			

conditions have equal pass rates ($Q = 23.25$, $p = 0.000113$). CodeLlama shows a similar directional pattern with 7.3–9.1% raw flip rates, but none reach significance after correction, and canonical renaming produces perfectly balanced forward and reverse flips (15 each). We treat the CodeLlama results as underpowered and focus subsequent analysis on DeepSeek.

Semantic interference. Table 2 reports perplexity ratios under all-variable renaming, where every user-defined identifier in the reference solution is replaced. The ordering across both models is consistent: common names (+19–22%) > single-letter (+9–12%) ≈ neutral (+9–14%) > canonical (+4–10%). For DeepSeek, five of six pairwise differences are significant after Bonferroni correction ($d = 0.7$–1.3); single-letter and neutral are indistinguishable ($d = 0.03$, $p = 0.67$). For CodeLlama, all six pairs are significant, though the smallest effect (canonical vs. single-letter, $d = 0.28$) is much weaker than the rest. The overall ordering directly contradicts the token-frequency hypothesis, which predicts that rare replacement tokens should cause the most disruption. Instead, names carrying misleading semantic expectations cause the most disruption, even though they are more frequent in training data.

The semantic interference effect interacts with the number of variables renamed (Figure 1). When only a single variable is renamed (n_{vars}=1, 19 problems), the ordering reverses: canonical produces the largest PPL increase, consistent with a per-token frequency cost. But as more variables are renamed, the interference from misleading names compounds, and by $n_{\text{vars}} \geq 2$ the common names scheme is consistently the most disruptive. The interaction between rename scheme and n_{vars} is strong ($r \approx 0.7$). This crossover identifies two competing mechanisms: token frequency effects dominate for single-variable renames, while semantic interference dominates when multiple misleading names compound.

Scope effects. The flip rate ordering from Table 1 (canonical worst) does not match the all-variable PPL ordering from Table 2 (canonical best). This discrepancy is explained by a difference in the scope of renaming. Flip rates use parameter-only renaming (because the function body is generated by the model), while PPL is measured under all-variable renaming. Under parameter-only PPL (Table 2, right columns), canonical shifts to among the most disruptive schemes for both models (tied with common names for CodeLlama), while single-letter remains the least disruptive.

This scope effect has a natural explanation. Under all-variable renaming, canonical identifiers (`var_0`, `var_1`) benefit from training data familiarity: these patterns appear in decompiled

Table 2: PPL ratios (renamed / original) on HumanEval under all-variable and parameter-only renaming. DeepSeek: 5/6 pairwise differences significant (single-letter ≈ neutral). CodeLlama: all 6 significant, though canonical vs. single-letter has d=0.28.

	DeepSeek		CodeLlama	
Scheme	All-var	Param	All-var	Param
Canonical	1.037	1.065	1.096	1.103
Single-letter	1.094	1.034	1.120	1.049
Common	1.187	1.079	1.219	1.102
Neutral	1.092	1.068	1.145	1.099

and obfuscated code, so the model has learned to handle them. We confirmed this by comparing canonical PPL to neutral PPL (`aaa`, `bbb`): neutral names are equally semantically empty but do not appear in training data, and they produce significantly higher PPL ($d > 1.0$ for both models, $p < 0.000001$). Under parameter-only renaming, the familiarity advantage is diluted because only a few tokens change, and the loss of semantic signal from descriptive parameter names becomes the dominant effect.

Structural vs. docstring sensitivity. HumanEval problems include docstrings that reference parameter names. Under AST-based renaming, which modifies only code identifier nodes and leaves docstrings intact, the non-identifier PPL excess is 7.3%. Under text-based regex renaming that also modifies docstring references, the excess rises to 13.8%.[1] This indicates that roughly 47% of the measured rename sensitivity is mediated by docstring changes rather than structural sensitivity to variable names in the code itself.

4 Interventions

We test four approaches at increasing depth in the transformer architecture. Each targets a different point where name information could be suppressed. All training experiments use 3 random seeds. Full methods are in Appendix A; results are summarized in Table 5.

4.1 Output-Level KL Consistency

We fine-tune CodeParrot-small [Tunstall et al., 2022] (111M parameters) with a consistency regularization loss $L = L_{\text{CE}} + \alpha \cdot L_{\text{KL}}$, where L_{KL} is the KL divergence between output distributions on original and canonically renamed code. We compare against an augmentation-only baseline that sees the same renamed examples but without the KL loss, with both conditions receiving the same effective number of cross-entropy updates per step.

Table 3 reports results across four α values and two gradient flow variants. At no setting does KL consistency improve the excess PPL ratio over augmentation alone. The relationship between α and sensitivity is monotonically decreasing: higher α produces larger KL reductions (8% at α=0.01 up to 38% at α=0.1) but worse PPL ratios. This KL-PPL dissociation is the central finding of this section. The consistency loss successfully makes output distributions more similar, but this does not translate into better predictions on renamed code. The loss

[1] The text-based measurement (13.8% mean excess across 10 random renamings per problem, excluding the original) and the AST-based measurement (7.3%) come from separate experimental runs with different rename samples; the comparison is approximate.

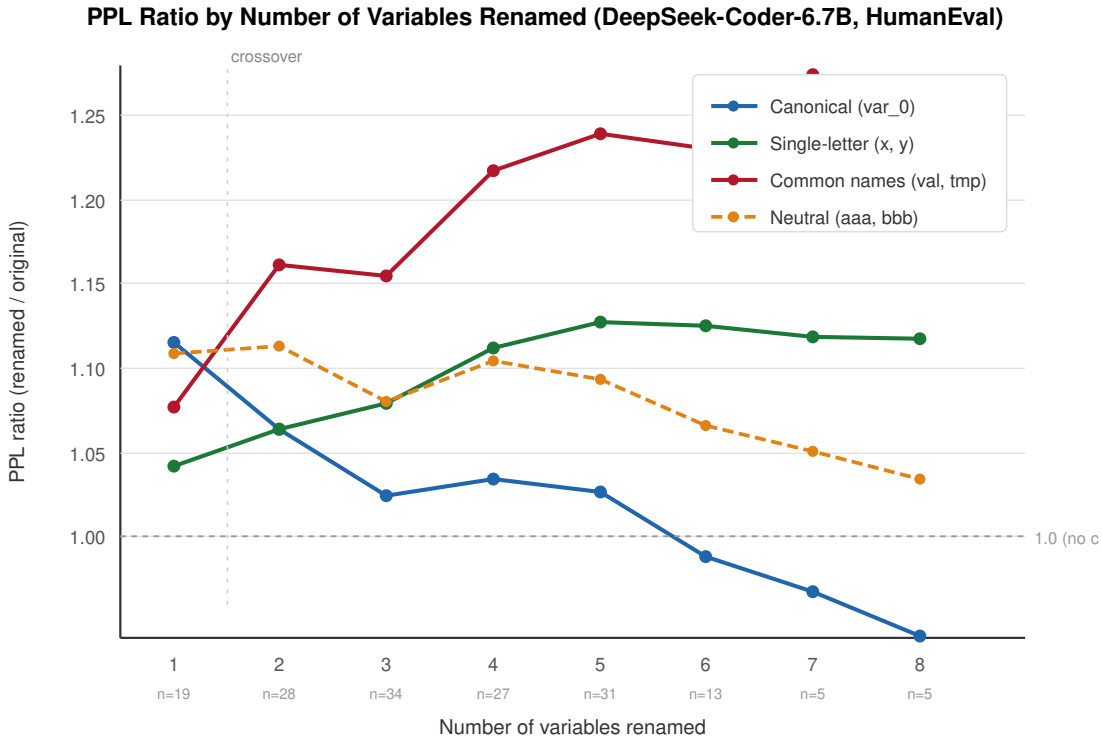

Figure 1: PPL ratio by number of variables renamed (DeepSeek-Coder-6.7B). Token frequency dominates at $n_{\text{vars}}=1$ (canonical worst). Semantic interference compounds and dominates at $n_{\text{vars}}\geq 2$ (common names worst). Interaction $r\approx 0.7$. Sample sizes shown below the x-axis.

softens the output distributions, making them less peaked and closer to each other in KL, without restructuring the internal representations that determine prediction quality.

The stop-gradient and full-gradient variants produce functionally identical results. This equivalence has a technical explanation: PyTorch's `F.kl_div` does not differentiate through its target argument, so R-Drop's [Liang et al., 2021] standard bidirectional KL formulation is already effectively a stop-gradient method. Explicitly computing full gradients does not change the outcome.

Data augmentation alone provides a modest benefit (10–17% reduction in excess PPL ratio at negligible quality cost), consistent with prior work showing that exposure to renamed code during training improves robustness [Paul et al., 2025]. The KL term adds nothing beyond this exposure.

4.2 Canonical Preprocessing

We train a 30M-parameter GPT-2 model from scratch on Python code where all user-defined variable names are replaced with positional identifiers (`v0`, `v1`, ...) during data loading. At evaluation time, input code is canonicalized using the same procedure. Invariance is perfect by construction: the PPL ratio between any two renamings is exactly 1.0000.

The comparison between canonical and standard models requires care, because the two evaluate on different text distributions. Full perplexity is misleading: the canonical model shows 13% lower full PPL, but this difference is entirely due to identifier tokens. Canonical identifiers follow a regular pattern (`v0`, `v1`, ...) and are trivially predictable (identifier PPL 1.9, compared

Table 3: KL consistency results (CodeParrot-small 111M, 2000 steps, 3 seeds). Excess PPL ratio = (renamed / original) -1. KL reduction relative to augmentation-only baseline.

Condition	Base PPL	Excess	KL red.
Augment-only	7.07	0.181	—
α=0.01 (stop)	7.08	0.180	8%
α=0.01 (full)	7.08	0.180	10%
α=0.025 (stop)	7.16	0.181	18%
α=0.05 (stop)	7.34	0.184	28%
α=0.1 (stop)	7.69	0.189	38%

Table 4: Canonical preprocessing cost decomposition (non-identifier PPL). Context informativeness = effect of evaluating the standard model on canonical code. Model quality = residual (total $-$ context). †At 111M, the total and context components use different baselines (augment-only vs. unfinetuned); the model quality residual is approximate.

Scale	Total	Context	Model
30M	+4.7%	+6.7%	-1.9%
111M	+10.6%	+9.7%	+0.8%†

to 14.3 for the standard model's diverse identifier vocabulary).

To enable fair comparison, we introduce **non-identifier perplexity**, computed only at token positions corresponding to keywords, operators, delimiters, and other structural tokens, excluding all identifier positions as classified by AST. Table 4 reports the cost decomposition. The total non-identifier PPL cost at 30M is +4.7%. To understand the source, we evaluate the standard model on canonical code (same model, different data). This control shows a +6.7% non-identifier PPL increase, representing the *context informativeness effect*: variable names carry genuine information about code structure that helps predict non-identifier tokens, and canonicalization removes this signal. The difference between the 6.7% context effect and the 4.7% total cost implies a -1.9% *model quality effect*: the canonical model is slightly better at predicting code structure on canonical code than the standard model is.

At 111M scale (CodeParrot-small fine-tuned on canonical code for 2000 steps), the total non-identifier PPL cost rises to +10.6%, with a context informativeness component of +9.7%. The increase from 30M to 111M is statistically significant ($p = 0.006$, paired t-test across seeds). Larger models exploit the semantic content of variable names more effectively, so losing that content costs more. This observation is based on two scale points with different model architectures and should be interpreted as a descriptive trend rather than a scaling law.

4.3 Embedding Anonymization

We modify the 30M GPT-2 architecture so that identifier tokens receive learned role embeddings instead of their standard token embeddings, with roles assigned by first-appearance order in the AST. This design should provide invariance by construction: two code snippets differing only in variable names would produce identical embeddings.

In practice, it fails. Base PPL degrades by 24.8% (11.34 vs. 9.09), and the excess PPL ratio under renaming is 0.211, which is 77% worse than the standard model's 0.119. The cause is BPE tokenization: different variable names produce different numbers of tokens (`numbers` tokenizes as 1 token, `var_0` as 3), which shifts the position embeddings of all subsequent tokens.

Table 5: Summary of all interventions (30M scale unless noted). Nonident = non-identifier PPL. Excess = mean PPL ratio -1 across rename schemes. 3 seeds per condition.

Approach	PPL	Nonid.	Excess	Inv.?
Standard	9.09	7.31	0.119	No
KL (best α, 111M)	7.08	—	0.180	No
Canon. preproc.	7.93	7.66	0.000	**Yes**
Embed. anon.	11.34	—	0.211	No
Q/K routing	9.70	7.59	0.134	No
Layer graft (best)	47.1	40.5	0.105	No

Even with identical role embeddings at identifier positions, the different number of positions breaks sequence alignment. We verified this by augmenting the tokenizer with atomic special tokens for `v0`–`v31`, which achieves 100% single-token coverage on HumanEval (2052/2052 identifier occurrences), but did not retrain with the augmented tokenizer.

4.4 Q/K Role Routing

We test a partial invariance design that preserves name information in the value pathway while making attention routing invariant. At identifier positions, query and key inputs are replaced with learned role embeddings, while value inputs use the standard hidden state. The motivation is that Q and K determine what attends to what, while V determines what information is extracted. If Q and K are invariant to renaming, the attention patterns should be identical across renamed variants, while V retains the original name information for content extraction.

Sensitivity increases by 12% rather than decreasing (excess ratio 0.134 vs. 0.119). The failure is consistent with Makelov et al. [2024]'s dormant pathway mechanism. At layer 1, the Q/K invariance guarantee holds. But the V output at layer 1 depends on the original token embedding, which is name-dependent. This V output enters the residual stream and becomes part of the hidden state at every position. At layer 2, non-identifier positions receive hidden states that contain name-dependent information from layer 1's V computations, and these feed into the standard Q and K projections. The invariance at layer 1 is undermined from layer 2 onward.

4.5 Layer Grafting

As a proxy for a co-trained interleaved architecture, we test post-hoc layer grafting between the separately trained standard and canonical-input models. Both hybrid orientations (standard layers 0–2 with canonical layers 3–5, and vice versa) collapse catastrophically: base PPL jumps from 9.09 to 47.1 and 51.4. The two models have developed incompatible internal representations during independent training. This does not rule out a co-trained interleaved architecture but eliminates post-hoc grafting as a viable approximation.

5 Analysis

5.1 The Residual Stream as Structural Blocker

The four failures in Section 4 each have a different proximate cause, but they share a common structure (Figure 2). The standard transformer uses an additive residual stream: the output of each layer is added to a running hidden state that all subsequent layers read from. Name

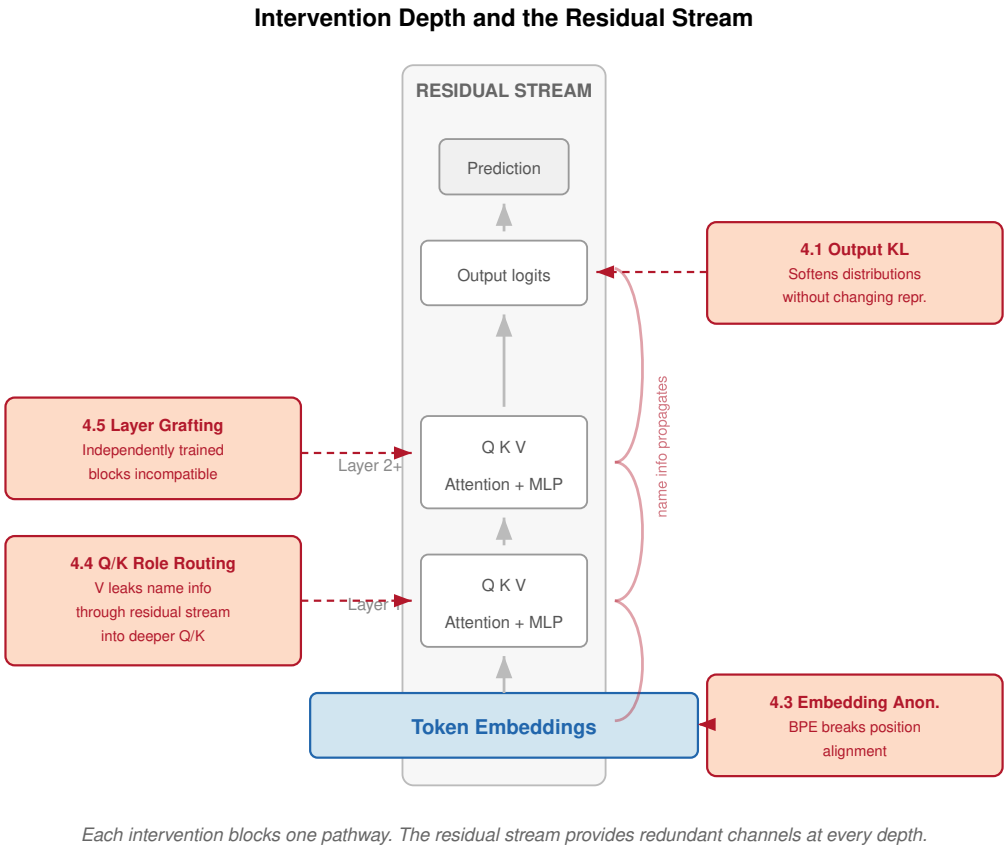

Figure 2: The residual stream as structural blocker. Name identity enters at the embedding layer and propagates upward through every computational pathway. Each of the four interventions blocks one pathway but leaves others open.

identity enters at the embedding layer, where different variable names produce different token embeddings, and from that point it propagates through every computational pathway. Each of our interventions blocks one pathway (output distributions, embeddings, Q/K attention, or layer boundaries) but leaves others open. The residual stream provides redundant channels, so blocking any subset is insufficient. This is consistent with Makelov et al. [2024]'s finding that subspace interventions in transformers are circumvented by dormant parallel pathways.

5.2 Concept Erasure Diagnostic

To directly characterize how name identity is encoded, we apply LEACE [Belrose et al., 2023] to the 30M standard model's hidden states. We collect activations on HumanEval for both original and canonically renamed code at all seven layers (embedding plus six transformer layers), train logistic regression probes to classify original versus canonical representations, and apply rank-1 LEACE erasure at the final layer.

Name identity is weakly linearly decodable at all layers (Figure 3). Probe accuracy is 52.4% at the embedding layer (barely above chance) and rises to 74.6% at the final layer. The mean-difference norm between original and canonical representations grows monotonically from 0.006 to 2.079, indicating that the model amplifies name-dependent information through depth.

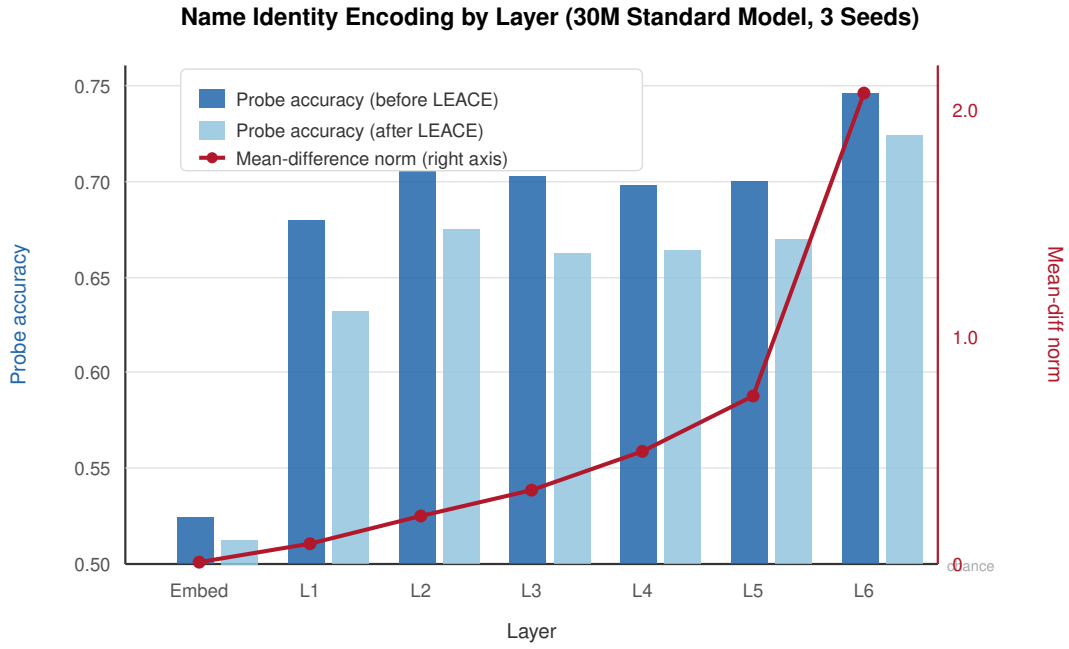

Figure 3: Name identity encoding by layer (30M, 3 seeds). Bars: probe accuracy classifying original vs. canonical representations, before and after rank-1 LEACE erasure. Line: mean-difference norm between original and canonical hidden states (right axis).

Rank-1 LEACE erasure at the final layer removes only 2.3 percentage points of probe accuracy (74.6% to 72.4%) while causing a 22% increase in non-identifier PPL. Name identity is not concentrated in a single direction but distributed across multiple dimensions. And the primary name-identity direction is entangled with structural prediction signal: removing it damages the model's ability to predict code structure. This entanglement reflects the genuine statistical relationship between variable names and code structure that the model has learned to exploit.

5.3 Practical Mitigation: Frame Averaging

Given that internal interventions face the architectural constraint described above, we turn to an external approach that does not modify the model. Frame averaging [Puny et al., 2022] achieves group invariance by averaging a function's output over elements of the symmetry group. For variable renamings, this amounts to running the model on multiple renamed versions and averaging the output distributions.

We apply frame averaging to DeepSeek-Coder-6.7B on HumanEval. For each problem, we generate K random renamings using AST-based replacement that modifies only code identifier nodes (preserving docstrings), run all $K{+}1$ versions through the model, and average the probabilities at non-identifier positions where BPE tokenization produces aligned token sequences. AST-based renaming achieves 97.0% per-version alignment (1591 of 1640 instances).

Table 6 reports results. At $K{=}2$ ($3\times$ inference cost), the non-identifier PPL excess drops from 6.59% to 2.10%, a 68.2% reduction ($p < 0.000001$, Wilcoxon signed-rank). The reduction is monotonically decreasing in K: $K{=}2$ gives 68.2%, $K{=}5$ gives 52.3%, $K{=}10$ gives 42.9%. This occurs because the frame average includes the original code as one of $K{+}1$ versions, and the original consistently has the lowest PPL (the model was trained on naturally named code).

Table 6: Frame averaging results (DeepSeek-Coder-6.7B, HumanEval, 122 aligned problems). Excess before averaging = 6.59%. Dilution pred. = excess $\times K/(K+1)$.

K	Cost	Excess	Dilution	Reduction
2	3×	2.10%	4.45%	68.2%
5	6×	3.18%	5.56%	52.3%
10	11×	3.81%	6.06%	42.9%

Higher K dilutes the original's contribution, pulling the average toward the group-invariant prediction, which is worse due to the information loss documented in §4.2. A naive dilution model predicts excess values of 4.45%, 5.56%, and 6.06% for $K{=}2, 5, 10$. The observed values are consistently lower, confirming genuine noise cancellation beyond simple dilution.

An important caveat: frame-averaged PPL is 2.1% worse than original PPL, not better. Frame averaging does not improve predictions when the original names are available. Its value is in producing more consistent predictions across different naming conventions, reducing the variance in quality that arises from naming choices. In applications such as code evaluation or vulnerability detection, where the goal is a naming-independent assessment, this consistency has practical value. In applications where the original names are available and the goal is simply the best prediction, frame averaging adds compute cost for no benefit.

6 Discussion and Conclusion

The invariance-information tradeoff. A recurring theme across our experiments is the tension between rename invariance and prediction quality. Variable names are not arbitrary labels in practice: they carry genuine information about the role and semantics of a variable, and models learn to use this information. Our context-information decomposition (§4.2) quantifies this directly: removing names costs 6.7–9.7% of non-identifier prediction quality, and this cost grows with scale. Any method that achieves invariance must suppress this signal. Canonical preprocessing discards it completely and pays the full cost. Frame averaging partially preserves it (the original is one of the averaged versions) at the cost of dilution. No method we tested achieves invariance without quality degradation.

Whether this tradeoff is inherent to the problem or specific to the standard transformer architecture is an open question. Işık and Li [2026] achieve provable symbol invariance through parallel embedding streams in logical reasoning, a domain where bound variable names carry no semantic content. Code occupies a middle ground: variable names are formally arbitrary but practically informative. An architecture that can exploit name information when it is reliable while remaining robust when it is not would resolve the tradeoff, but no existing design achieves this for code.

Limitations. The most significant limitation is scale. Benchmark experiments use pre-trained 7B models, but all intervention experiments are conducted at 30–111M parameters. The architectural constraints we identify are properties of the transformer that do not depend on scale, but quantitative results may differ at 7B. The growing context cost from 30M to 111M suggests that interventions may become harder at scale, but this is based on two points from different architectures.

Our evaluation uses HumanEval exclusively (164 short algorithmic Python functions). We test only variable renaming, not other semantic-preserving transformations. The V-leakage

mechanism in §4.4 is inferred from the overall failure of Q/K routing, not measured directly through hidden-state probing, though it is consistent with Makelov et al. [2024]'s framework. Frame averaging is evaluated using perplexity, not generation; extending it to autoregressive code generation would require solving an identifier alignment problem across renamings. Our KL consistency experiments use a single rename scheme (canonical) and a single training duration (2000 steps at 111M).

Practical recommendations. For applications requiring strict naming invariance: canonicalize inputs (7–10% non-identifier PPL cost). For reduced sensitivity without invariance: frame average at K=2 (68% reduction, 3× cost, no model changes). For modest improvement at negligible cost: data augmentation during fine-tuning (10–17% reduction at 111M scale, 2000 steps). Adding a KL consistency term to augmentation does not improve over augmentation alone.

Conclusion. We tested four approaches to rename invariance at different depths in the transformer, and all fail. Each failure is mechanistically explained: name identity enters at tokenization, propagates through the residual stream, and cannot be removed by intervening at any single pathway. Name identity is not localized in representation space; it is distributed across multiple directions and entangled with the structural prediction signal that makes variable names useful in the first place. Canonical preprocessing achieves invariance but at a measurable cost that grows with model scale. Test-time frame averaging at K=2 provides a practical middle ground, reducing sensitivity by 68% at 3× inference cost without any model modification. Resolving the invariance-information tradeoff likely requires architectures that structurally separate invariant and name-aware computations, rather than sharing a single residual stream.

A Experimental Details

Benchmark. DeepSeek-Coder-6.7B-base [Guo et al., 2024] and CodeLlama-7B-Python-hf [Rozière et al., 2023], evaluated on HumanEval [Chen et al., 2021] (164 problems). Baseline pass@1 validated against published numbers (DeepSeek: 48.2% vs. 47.6% published; CodeLlama: 38.4% vs. ∼38–40%). Four rename schemes. PPL measurements use AST-based renaming (docstrings preserved). Pass@1 flip rates use text-based word-boundary regex replacement on raw prompts (parameter names only, docstrings also modified). AST parsing via Python `ast` module and tree-sitter (100% success on HumanEval and MBPP). Metrics: PPL ratios on reference solutions, pass@1 flip rates via greedy decoding with stop-sequence truncation at function boundaries, non-identifier PPL at structural token positions classified by AST. Statistical tests: McNemar (flip rates), Cochran's Q (across schemes), paired t-tests and Wilcoxon signed-rank (PPL), all Bonferroni-corrected where applicable.

KL consistency. CodeParrot-small (111M), 2000 fine-tuning steps. $L = L_{\text{CE}} + \alpha \cdot L_{\text{KL}}$ with canonical renames. $\alpha \in \{0.01, 0.025, 0.05, 0.1\}$ (stop-gradient) and α=0.01 (full-gradient). CE-budget controlled: augmentation-only baseline receives same effective CE updates per step. Preliminary α=0.5 produced +84.6% base PPL (code rename KL is 2–9 nats, ∼100× R-Drop's typical values, necessitating recalibration). 3 seeds per condition.

From-scratch training. GPT-2 architecture: 6 layers, 6 heads, d_{model}=384, ∼30M parameters. Trained from random initialization on The Stack Python subset [Kocetkov et al., 2022], 5000 steps, effective batch size 64 (8×8 gradient accumulation), sequence length 512, ∼327M

tokens. AdamW (lr=6×10^{-4}, cosine decay to 6×10^{-5}, warmup 500 steps, β_2=0.95, weight decay =0.1). 3 seeds per condition. Canonical-input: all variables canonicalized during data loading. Embedding anonymization: 18 role embeddings (STANDARD + VAR_0–VAR_15 + OVERFLOW), replacing token embeddings at identifier positions. Q/K routing: role embeddings for Q and K at identifier positions; standard hidden states for V. Tokenizer augmentation (`v0`–`v31` as special tokens) verified 100% atomicity on HumanEval (2052/2052 occurrences).

Layer grafting. Post-hoc state-dict swap between seed-matched standard and canonical-input checkpoints. Two orientations: standard layers 0–2 with canonical layers 3–5, and reverse.

Concept erasure. LEACE applied to 30M standard model hidden states at all 7 layers (embedding + 6 transformer layers). Logistic regression probes trained per layer on 20,000 tokens each from original and canonical code. Rank-1 LEACE projection at final layer; PPL re-evaluated through projected hidden states. 3 seeds, HumanEval (164 problems).

Frame averaging. DeepSeek-Coder-6.7B on HumanEval. 10 random AST-based renamings per problem (code identifiers only, docstrings preserved). Random names: 3–5 lowercase characters, verified against Python builtins and keywords. Per-token log-probabilities averaged in probability space at aligned non-identifier positions. $K \in \{2, 5, 10\}$. Executed on a single A10G.

Compute. All training experiments on 2× NVIDIA RTX 3090. Frame averaging inference on a single A10G. Total: ∼29.4 GPU-hours across 16 research iterations.

References

Zachary Ankner, Alex Renda, and Michael Carbin. Renamer: A transformer architecture invariant to variable renaming. In *Machine Learning for Systems Workshop, NeurIPS 2023*, 2023. URL https://openreview.net/forum?id=AilSfYM8wN.

Nora Belrose, David Schneider-Joseph, Shauli Ravfogel, Ryan Cotterell, Edward Raff, and Stella Biderman. LEACE: Perfect linear concept erasure in closed form. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, 2023. URL https://arxiv.org/abs/2306.03819.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. URL https://arxiv.org/abs/2107.03374.

José Gonçalves, Tiago Dias, Eva Maia, and Isabel Praça. SCoPE: Evaluating LLMs for software vulnerability detection. *arXiv preprint arXiv:2407.14372*, 2024. URL https://arxiv.org/abs/2407.14372.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. DeepSeek-Coder: When the large language model meets programming – the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024. URL https://arxiv.org/abs/2401.14196.

Floris Holstege, Shauli Ravfogel, and Bram Wouters. Preserving task-relevant information under linear concept removal. In *Advances in Neural Information Processing Systems 38 (NeurIPS)*, 2025. URL https://arxiv.org/abs/2506.10703.

İlker Işık and Wenchao Li. Names don't matter: Symbol-invariant transformer for open-vocabulary learning. *arXiv preprint arXiv:2601.23169*, 2026. URL https://arxiv.org/abs/2601.23169.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Baez, Jade Jernigan, and Leandro von Werra. The stack: 3 TB of permissively licensed source code. *Transactions on Machine Learning Research*, 2022. URL https://arxiv.org/abs/2211.15533.

Man Ho Lam, Chaozheng Wang, Jen-tse Huang, and Michael R. Lyu. CodeCrash: Exposing LLM fragility to misleading natural language in code reasoning. In *Advances in Neural Information Processing Systems 38 (NeurIPS)*, 2025. URL https://arxiv.org/abs/2504.14119.

Cuong Chi Le, Minh V. T. Pham, Cuong Duc Van, Hoang N. Phan, Huy N. Phan, and Tien N. Nguyen. When names disappear: Revealing what LLMs actually understand about code. *arXiv preprint arXiv:2510.03178*, 2025. URL https://arxiv.org/abs/2510.03178.

Xiaobo Liang, Lijun Wu, Juntao Li, Yue Wang, Qi Meng, Tao Qin, Wei Chen, Min Zhang, and Tie-Yan Liu. R-Drop: Regularized dropout for neural networks. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, 2021. URL https://arxiv.org/abs/2106.14448.

Aleksandar Makelov, Georg Lange, and Neel Nanda. Is this the subspace you are looking for? An interpretability illusion for subspace activation patching. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2311.17030.

Indraneil Paul, Haoyi Yang, Goran Glavaš, Kristian Kersting, and Iryna Gurevych. ObscuraCoder: Powering efficient code LM pre-training via obfuscation grounding. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2504.00019.

Omri Puny, Matan Atzmon, Heli Ben-Hamu, Ishan Misra, Aditya Grover, Edward J. Smith, and Yaron Lipman. Frame averaging for invariant and equivariant network design. In *Proceedings of the 10th International Conference on Learning Representations (ICLR)*, 2022. URL https://arxiv.org/abs/2110.03336.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code Llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023. URL https://arxiv.org/abs/2308.12950.

Lewis Tunstall, Leandro von Werra, and Thomas Wolf. CodeParrot: A large-scale Python code dataset and models, 2022. URL https://huggingface.co/codeparrot/codeparrot-small. HuggingFace model card.

Shiqi Wang, Zheng Li, Haifeng Qian, Chenghao Yang, Zijian Wang, Mingyue Shang, Varun Kumar, Samson Tan, Baishakhi Ray, Parminder Bhatia, Ramesh Nallapati, Murali Krishna Ramanathan, Dan Roth, and Bing Xiang. ReCode: Robustness evaluation of code generation models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023. URL https://arxiv.org/abs/2212.10264.

Fanny Yang, Zuowen Wang, and Christina Heinze-Deml. Invariance-inducing regularization using worst-case transformations suffices to boost accuracy and spatial robustness. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019. URL https://arxiv.org/abs/1906.11235.

Yuhao Zhang, Shiqi Wang, Haifeng Qian, Zijian Wang, Mingyue Shang, Linbo Liu, Sanjay Krishna Gouda, Baishakhi Ray, Murali Krishna Ramanathan, Xiaofei Ma, and Anoop Deoras. CodeFort: Robust training for code generation models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 5262–5277, 2024. URL https://arxiv.org/abs/2405.01567.