

False Proximity: Frozen Code Embeddings Invert Similarity on Behaviourally Divergent Mutant Pairs

Keletso J. Letsholo

Higher Colleges of Technology

Nourchene Elleuch Ben Ayed

`nbenayed@hct.ac.ae`

Higher Colleges of Technology

Achraf Ghorbel

Higher Higher Colleges of Technology

Muath Alrammal

University of Wollongong in Dubai

Research Article

Keywords: Code Representation Learning, CodeBERT, Cosine Similarity, Behavioural Divergence, Mutation Testing, Code Clone Detection

Posted Date: October 20th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-10188373/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

False Proximity: Frozen Code Embeddings Invert Similarity on Behaviourally Divergent Mutant Pairs

Keletso J. Letsholo^{1†}, Nourchène Elleuch Ben Ayed^{1*†},
Achraf Ghorbel², Muath Alrammal³

¹Computer Information Science, Higher Colleges of Technology, Abu Dhabi, UAE.

²Computer Information Science, Higher Colleges of Technology, Sharjah, UAE.

³School of Computer Science, University of Wollongong in Dubai, Dubai, UAE.

*Corresponding author(s). E-mail(s): nbenayed@hct.ac.ae;
Contributing authors: kletsholo@hct.ac.ae; aghorbel@hct.ac.ae;
MuathAlrammal@uowdubai.ac.ae;

[†]These authors contributed equally to this work.

Abstract

Embedding-based code search, clone detection, and defect-prediction pipelines routinely use cosine similarity over frozen CodeBERT embeddings as a lightweight proxy for behavioural equivalence. We ask whether that signal separates behaviourally divergent code from genuinely equivalent code, and find that it does not merely fail: it inverts. From Project_CodeNet_Python800 we construct a controlled benchmark of 800 execution-verified equivalent pairs and 1,833 I/O-verified mutant pairs whose single-operator edits (conditional negation, relational-operator replacement, and algorithmic inversion) change the output on every usable oracle input; the analysis set holds 2,617 pairs over 784 problems at Cohen’s $\kappa = \mathbf{0.925}$. Across eight embedding configurations spanning three frozen encoders (CodeBERT, with GraphCodeBERT and UniXcoder evaluated at their standard final-layer CLS setting), the discrimination area under the ROC curve (AUC) stays far below the 0.5 chance level (0.017 to 0.090) and the false-high-similarity rate remains at or above 0.87: a behaviourally divergent mutant is scored more similar to its original than two independently written correct

solutions are to each other, in more than nine comparisons out of ten. The inversion is uniform across the three mutation categories and only partly explained by token and structural overlap. It withstands the training-free corrections a practitioner would reach for: a normalised AST-distance signal, a contrastively trained encoder (CodeSage), and standard isotropy post-processing each leave it unchanged or worse, because single-site mutants are structurally near-identical to their originals. For polarity-sensitive equivalence decisions on Python code, frozen embedding similarity reflects surface proximity rather than behaviour; contrastive fine-tuning or execution-based verification is required instead.

Keywords: Code Representation Learning, CodeBERT, Cosine Similarity, Behavioural Divergence, Mutation Testing, Code Clone Detection

1 Introduction

Code search, clone detection, and many defect-prediction pipelines rely, at some point, on a similarity judgement: are these two functions doing the same thing? CodeBERT [1] is a widely used encoder-only reference model for that question. Trained on a bimodal corpus of natural language and code with masked language modelling and replaced-token detection objectives, it produces token-level and sequence-level representations that serve as a baseline across most code intelligence benchmarks. More recent code models (CodeLlama, StarCoder2, DeepSeek-Coder) are predominantly decoder-only and optimised for generation, not embedding-based comparison. The encoder-only family (CodeBERT, GraphCodeBERT [2], UniXcoder [3]) remains the appropriate reference set for representation analysis precisely because these models are designed to produce fixed-size embeddings for pairwise comparison. In practice, representations are extracted under different configurations: the choice of layer and pooling strategy both affect the resulting similarity scores. Whether any configuration reliably separates behaviourally divergent mutant pairs from equivalent pairs has not been established.

The standard way to compare two CodeBERT embeddings is cosine similarity. The intuition is straightforward: if two functions are semantically close, their vector representations should point in roughly the same direction. The problem is that this intuition was never validated for a semantically charged scenario that appears often in real software: the case where two code fragments compute *opposite* results. A function that returns the maximum of two numbers and a function that returns the minimum are not equivalent; they are, in a precise sense, inversions of each other. Yet at the token level, they share nearly every token. The only difference may be a single comparison operator.

This situation is not exotic. It arises when reviewing patches that accidentally invert a condition, when testing libraries of sorting and searching functions, when analysing programs produced by mutation testing frameworks [4], and when building code-change summarisers that need to characterise what a diff actually *does*. In all these cases, a similarity model that assigns high cosine similarity to behaviourally divergent mutant pairs is worse than useless: it actively misleads.

There is a theoretical reason to expect this failure. Ethayarajh [5] showed that BERT-family models produce anisotropic embeddings: representations that cluster in a narrow cone of the high-dimensional space, making cosine similarity between any two representations relatively high regardless of semantic content. Gao et al. [6] confirmed this and showed that contrastive fine-tuning can correct it for natural language. Code embeddings face an additional structural challenge: behaviourally divergent mutant programs often share vocabulary and similar syntactic structure, which may further reduce the separability of their embeddings even for a model with isotropic representations. Contrastive training frameworks such as ContraCode [7] demonstrate that functionality-aware representations *can* be learned, but only with access to large quantities of augmented pairs during pre-training. No such polarity-distinguishing signal is present in CodeBERT’s original training objectives, making it unlikely that the model learns to encode behavioural divergence in a geometrically separable way.

The result is a measurable gap: frozen CodeBERT embeddings are unlikely to discriminate behaviourally divergent mutant pairs from equivalent pairs reliably, and the magnitude of that gap is unknown. The existence of some failure is predictable from theory; what is not known is how severe it is, whether it varies across mutation operator classes, whether it is uniform across embedding configurations and model variants, and whether a lightweight structural correction can reduce it without fine-tuning. These are empirical questions, and their answers have direct consequences for practitioners. A near-chance area under the receiver operating characteristic curve (AUC) across all configurations would imply that no embedding choice mitigates the problem. Because mutant pairs are lexically near-identical to their originals while equivalent pairs are independently written, an AUC *below* 0.5 is also plausible a priori; that outcome would indicate systematic inversion (mutant pairs ranked above equivalent pairs), a stronger failure than noise. A partial failure concentrated in specific categories would instead guide practitioners to restrict cosine similarity to the categories where it is reliable. Without measurements, neither recommendation can be made. To our knowledge, no prior work has measured this gap systematically, categorised it by mutant category, or evaluated training-free mitigations at inference time. The distinction from prior geometry work is sharp: anisotropy and surface-sensitivity are established for code embeddings [5, 8, 9], but those results predict an *uninformative* similarity, whereas on pairs controlled for behaviour we find an *inverted* one, with the behaviourally divergent pair ranked above the equivalent. We address these questions directly:

- **RQ1:** To what extent does CodeBERT cosine similarity assign falsely high scores to mutant code pairs across embedding configurations, and does this failure persist in GraphCodeBERT [2] and UniXcoder [3]?
- **RQ2:** Across the three mutant categories that produce behaviourally divergent pairs (conditional negation (COD), relational-operator replacement (ROR), and algorithmic inversion (AI); defined in Section 3.1), which categories exhibit the highest cosine-similarity confusion rates, and what token-level or structural features explain these confusions?
- **RQ3:** Can a training-free hybrid that adds a normalised abstract syntax tree (AST) structural signal to CodeBERT cosine similarity repair the discrimination failure

on the mutant-pair dataset, or do structural similarity and behavioural divergence dissociate, given that mutants are structurally near-identical to their originals?

RQ1 and RQ2 treat the pre-trained models as fixed probes: we do not modify weights or fine-tune. RQ1 uses equivalent pairs as a reference distribution to establish what legitimate high similarity looks like for each model; the failure under study is mutant pairs scoring within that same range. The deliberate choice not to fine-tune is principled: contrastive fine-tuning [6, 7] can repair anisotropic embedding spaces, but it requires a labelled corpus of contrast pairs and modifies model weights. Neither requirement is met in many deployment settings where CodeBERT is used as a frozen backbone in a larger pipeline. A tree-edit distance over the functions’ abstract syntax trees can be computed directly from source code at inference time and combined with cosine similarity as a lightweight structural signal. RQ3 asks whether that training-free signal can repair the failure on the mutant-pair dataset. It cannot, and the reason is structural: the mutant pairs studied here are near-identical to their originals in syntax, so the AST term raises their scores rather than lowering them (Section 3.5.1). RQ3 makes this dissociation between structural similarity and behavioural divergence precise under the no-fine-tuning constraint.

To answer these questions, we make the following contributions:

1. We construct a labelled dataset drawn entirely from **Project CodeNet Python800** [10]: 800 equivalent pairs (one per problem, selected from judge-accepted submissions and verified by execution on the available oracle inputs) and 1,833 input/output (I/O)-verified behaviourally divergent mutant pairs (COD: 779, ROR: 775, AI: 279), for a total of 2,633 pairs; the 784 problems contributing both pair types form the 2,617-pair analysis set used in all RQ analyses (Section 3.2.6). Each mutant pair is produced by a single-site AST transformation and retained only when execution confirms output divergence from the original on *every* usable oracle input; this is stricter than the standard kill criterion of mutation testing, under which a single distinguishing test suffices [4], and the label still asserts divergence on tested inputs only, not logical complementarity over the full input domain. Drawing both pair types from one corpus removes the domain-shift confound of mixing sources. Both subsets undergo inter-rater reliability (IRR) validation (target Cohen’s $\kappa \geq 0.80$; Section 3.2.5). Construction details appear in Section 3.2.
2. We conduct a systematic empirical analysis (RQ1) measuring the rate at which frozen CodeBERT embeddings assign falsely high cosine similarity scores to mutant pairs (scores indistinguishable from those on equivalent pairs) across eight embedding configurations: six CodeBERT variants and two baseline models (GraphCodeBERT, UniXcoder). We further check that the failure is not an artefact of lexical near-duplication (a token-overlap-matched comparison), is not specific to masked-language-model pre-training (a frozen contrastively trained encoder, CodeSage), and is not removed by two standard training-free isotropy corrections (whitening and all-but-the-top).

3. We provide a category-level breakdown (RQ2) of which mutant categories produce the highest confusion rates, with explanations grounded in token-level and AST-level overlap analysis.
4. We propose a hybrid similarity model (RQ3) that combines CodeBERT cosine similarity with normalised AST edit distance as a plug-in, training-free correction, and use it as a diagnostic: on the mutant-pair dataset we show that structural similarity dissociates from behavioural equivalence, so that no convex combination of cosine and AST similarity repairs the failure measured in RQ1. The deployment-oriented evaluation of this correction on inter-project clone benchmarks is left to future work.

The failure documented here is not inherent to CodeBERT but arises from the specific practice of applying cosine similarity over frozen embeddings for polarity-sensitive decisions. The model performs well on the tasks it was designed for; the problem arises when sequence-level representations trained without any polarity-distinguishing signal are used to judge whether two programs are functionally equivalent. High cosine similarity in that setting reflects surface proximity, not semantic equivalence, and systems that treat the two as interchangeable will produce false positives on exactly the class of pairs studied here. The empirical scope of this paper is Python; its limits and the open cross-language question are discussed in Sections 5 and 7.

The rest of the paper is organised as follows. Section 2 surveys related work. Section 3 describes our dataset construction, experimental design, and hybrid model. Section 4 presents results for all three RQs. Section 5 interprets findings in terms of their practical and theoretical implications. Section 6 discusses threats to validity. Section 7 concludes the paper.

2 Related Work

The problem studied in this paper sits at the intersection of four research streams: pre-trained encoder models for code, the geometry of contextual embeddings, code clone detection and semantic similarity evaluation, and mutation-based program analysis. We survey each stream, identify the most closely related contributions, and conclude each subsection with a paragraph that explicitly positions the present work.

2.1 Pre-trained Encoder Models for Code and Their Representations

Pre-trained language models for source code trace their lineage to BERT [11], whose masked language modelling objective transferred effectively to new domains. CuBERT [12] applied masked language modelling directly to Python; CodeBERT [1] extended this by using a bimodal training corpus of natural language and six programming languages, adding a replaced-token detection objective. The resulting model produces token-level and sequence-level representations that serve as a frozen backbone in code search, clone detection, and defect prediction pipelines. GraphCodeBERT [2] incorporated data-flow graphs extracted from source code as an additional structural pre-training signal through two structure-aware tasks (variable-origin prediction and

data-flow edge prediction), consistently outperforming CodeBERT on code search, clone detection, code translation, and code refinement. UniXcoder [3] adopted a unified cross-modal architecture that also consumes code comments and AST-derived content, extending the encoder to generative tasks via a decoder attached to a shared encoder. More recent decoder-only models, such as CodeLlama, StarCoder2, and DeepSeek-Coder, are optimised for token generation rather than for producing fixed-size embeddings suitable for pairwise cosine comparison; the encoder-only family therefore remains the appropriate reference set for representation analysis, and CodeBERT is among the most widely studied members.

Probing studies have examined what semantic content these representations actually encode. Naik et al. [8] applied Representational Similarity Analysis (RSA) to CodeBERT and found that current pre-training methods do not induce semantic grounding: the model captures form-based statistical patterns, and even modest fine-tuning on semantically relevant tasks substantially improves the correspondence between embeddings and functional content. Zhang et al. [9] demonstrated that anonymising all programmer-defined names simultaneously (variable names, method definitions, and method invocations) reduces GraphCodeBERT-based code search accuracy in Java from approximately 70% to 17%, confirming that encoder-only code representations are dominated by surface lexical cues rather than control-flow or algorithmic structure. These findings establish a principled expectation that such embeddings will be insufficient for tasks requiring fine-grained functional discrimination, but they do not examine the specific failure mode of mutant pairs.

We use CodeBERT [1], GraphCodeBERT [2], and UniXcoder [3] as fixed probes, with no changes to architecture or weights. Unlike probing studies that perturb programs while *preserving* semantics (variable renaming, statement reordering), we study semantics-*inverting* transformations: not whether the model generalises across stylistic variants, but whether it assigns *lower* cosine similarity to a program that inverts the original’s behaviour.

2.2 Contextual Embedding Geometry and the Limits of Cosine Similarity

Ethayarajh [5] showed formally and empirically that BERT, ELMo, and GPT-2 produce anisotropic representations: the average cosine similarity between randomly sampled representations is substantially above zero, and representations in higher layers exhibit stronger anisotropy than those in lower layers. Li et al. [13] confirmed that BERT sentence embeddings are semantically non-smooth at the sequence level and that direct cosine comparison is a less discriminative signal between close and arbitrary pairs. This line of work identified the root cause: representations cluster in a narrow cone of the high-dimensional space, so cosine similarity cannot reliably distinguish genuinely close pairs from arbitrary ones.

Gao et al. [6] showed that contrastive training via SimCSE regularizes the embedding space toward isotropy for natural language, improving semantic textual similarity and, by their analysis, both aligning positive pairs and spreading representations more uniformly over the hypersphere. SimCSE operates on natural language, however, with no mechanism for the structural properties of programs. Diera et al. [14] confirm that

code-LM embeddings (including CodeBERT) are strongly anisotropic and that ZCA whitening improves semantic code search, while contrastive fine-tuning raises isotropy only marginally; their post-processing corrections are the code-domain analogue of the whitening and all-but-the-top transforms we evaluate in RQ1 (Section 3.3.1).

For code specifically, Jain et al. [7] showed that reconstruction-based pre-training (masked language modelling) leaves code representations sensitive to surface-level edits even when those edits preserve semantics: adversarially selected semantics-preserving transformations cause RoBERTa’s performance to fall below chance on a zero-shot clone detection benchmark. ContraCode addresses this by pre-training on augmented pairs drawn from compiler transformations that preserve functionality, producing representations more invariant to code form. Liu et al. [15] proposed ContraBERT, which applies a similar contrastive pre-training strategy to enhance encoder-based code models such as CodeBERT, reporting gains on code search and clone detection tasks. Both contributions confirm that contrastive training can repair surface sensitivity for semantic equivalents. Neither, however, explores the inverse direction: whether programs that invert the behaviour of the original are correctly assigned low similarity. By design, contrastive objectives trained on positive pairs and random negatives have no exposure to pairs that are syntactically close yet behaviourally divergent.

We begin from the same diagnosis as Ethayarajh (2019) and Li et al. (2020) but operationalise it in a code-specific, polarity-sensitive setting. Where Gao et al. (2021), Jain et al. (2021), and Liu et al. (2023) address the problem through contrastive training, we evaluate the failure under the no-fine-tuning constraint common in deployment and propose a training-free structural correction at inference time. Because a single operator change produces near-complete token overlap, the failure is unlikely to be correctable by isotropising the space alone: it arises from two compounding factors, geometric compression and the near-identical surface form of the input pairs.

2.3 Code Clone Detection and Semantic Similarity Evaluation

Code clone detection has been studied for over two decades, with approaches spanning token-level comparison, AST-based subtree matching, program dependency graph analysis, and embedding-based methods [16]. The taxonomy introduced by Roy et al. [16] classifies clones into four types: Type-1 (identical), Type-2 (syntactically near-identical with renamed identifiers), Type-3 (syntactically similar with added or modified statements), and Type-4 (semantically equivalent but structurally dissimilar). Deep learning methods have become dominant for the more challenging Type-3 and Type-4 cases. BigCloneBench [17], a manually validated inter-project Java repository of over six million clone pairs across ten functionalities, remains a de facto standard evaluation resource for clone detection tools, and CodeBERT fine-tuned on its binary clone classification task achieves very high recall on Type-1 and Type-2 clones [1].

Sonnekalb et al. [18] examined the generalisation of CodeBERT-based clone detection on BigCloneBench and found that measured performance is sensitive to how the evaluation set is constructed, so that strong results on one distribution of pairs do not necessarily transfer to more challenging near-miss cases. Ahmed et al. [19] proposed a scalable architecture that fine-tunes CodeBERT with contrastive loss for approximate

nearest-neighbour search across large codebases, reporting strong precision and recall for large-scale clone retrieval. These contributions share a common orientation: making equivalent pairs *rank higher*. Neither addresses the complementary problem of making mutant pairs *rank lower*, a failure mode that is invisible to standard recall-oriented evaluation.

Interpretability studies have begun to examine what CodeBERT attends to during clone detection. Abid et al. [20] applied SHAP-based attribution (CloneSHAP) to CodeBERT’s clone predictions on the SemanticCloneBench dataset and found that the statements CodeBERT weights most heavily correspond only weakly to those humans judge as core to a function’s behaviour (agreement of roughly 53% for Java and 58% for Python). They further observed that CodeBERT’s predictions are sensitive to non-essential code: removing such code flipped the prediction in half of the cases examined. These results indicate that CodeBERT’s clone decisions are driven substantially by features other than high-level functional structure. This observation motivates the failure mode we study: logically inverted programs often share most of their tokens and entire syntactic subtrees, and a model attending primarily to such shared surface structure has no reliable basis for distinguishing a function from its semantic negation. Song et al. [21] provide complementary evidence that AST-structural distance predicts execution match better than token-level overlap metrics; their result is discussed in Section 2.5, where it motivates the structural component of RQ3.

Our dataset serves a specific diagnostic purpose: measuring the false-high-similarity rate of frozen embeddings on mutant pairs. It is not a clone-detection contribution and, drawn from a single controlled source with I/O-verified labels, is not intended to replace BigCloneBench. BigCloneBench remains the right setting for the deployment-oriented question of whether the structural correction (Section 3.5.1) suppresses false positives on confirmed non-clones without degrading Type-1 and Type-2 recall, which is left to future work.

2.4 Robustness of Neural Code Models Under Program Transformation

A substantial body of work has evaluated how neural code models respond to controlled program transformations. Henkel et al. [22] studied adversarial robustness under sequences of semantics-*preserving* transformations and showed that adversarial training substantially improves invariance to such perturbations. Rabin and Alipour [23] evaluated the generalisation of neural program models built on code2vec, code2seq, and gated graph neural networks, tested under six semantics-preserving transformations on a method-name prediction task, finding that even small semantics-preserving changes frequently alter predictions, and arguing that generalisation across equivalent implementations is a necessary prerequisite for reliable code intelligence. Probing analyses of CodeBERT reach a consistent conclusion: its representations are driven largely by surface lexical cues, so that semantics-preserving perturbations such as identifier renaming degrade performance substantially [8, 9]. These findings collectively establish that pre-trained code models are surface-sensitive, but they study a failure of *equivariance*: the model should be invariant to stylistic variation but is not.

We study a different failure: one of *sensitivity* under semantic inversion. A model that orders the pairs correctly must score the mutant pair *lower* than the equivalent pair; this is discriminative power over a specific class of functionally distinct pairs, not robustness to perturbation. We are not aware of prior robustness work that tests whether frozen code embeddings meet this requirement.

The mutation testing literature provides the closest formal grounding for the transformation taxonomy used in this paper. Jia and Harman [4] surveyed the development of mutation testing and catalogued relational and logical operator mutations as core, widely adopted operator classes across mutation frameworks. Papadakis et al. [24] update that survey over the following decade and document equivalent-mutant detection as a still-open problem of the field; our killed-only retention rule (Section 3.1) sidesteps it by construction, keeping only mutants with execution-confirmed divergence on every usable oracle input. Our COD and ROR categories are implemented through these standard conditional and relational operators; our AI category groups higher-level inversions that are operational rather than canonical operators. Equivalent mutant detection, deciding whether a syntactic mutation changes program behaviour, remains an active benchmark task; on Java mutant pairs, LLM-based detectors substantially outperform earlier techniques, with fine-tuned code embeddings performing best [25]. That this judgement over near-identical programs warrants dedicated detectors at all is part of the motivation for the present study.

2.5 Hybrid and Structural Similarity Approaches

Structural signals (AST subtree metrics, tree-edit distance, and program dependency graphs) have complemented token-level methods in code similarity research since before the deep learning era [16]. The tree-edit-distance problem for ordered labelled trees was solved by the Zhang-Shasha algorithm [26] in $O(m^2n^2)$ time. RQ3 of this paper computes tree-edit distance using APTED [27], a more recent memory-efficient algorithm, implemented in the `apted` library. Song et al. [21] demonstrated that AST edit distance computed via APTED (their TSED metric) achieves higher accuracy than BLEU score and Jaccard similarity at predicting execution match across six programming languages, supporting the use of AST-structural distance as a proxy for functional similarity.

Combining learned embeddings with structural signals is less explored for source-code function-level similarity than for binary analysis, where the absence of source text makes structure essential. Contrastive recipes such as ContraBERT (Section 2.2) require weight modification and an augmented corpus; the RQ3 hybrid instead is a plug-in, training-free correction that linearly combines CodeBERT cosine similarity with normalised AST edit distance at inference time. It targets the false-positive signature common in clone screening, where pairs share surface tokens but diverge in overall structure. Mutant pairs carry the opposite signature: a single mutation site leaves the tree almost unchanged, so the normalised edit distance is near zero. We use this inversion deliberately (Section 3.5.1), yielding a clean dissociation between structural similarity and behavioural divergence; the deployment setting the correction targets is left to future work.

2.6 Summary and Research Gap

Table 1 positions the present study against the most closely related prior works, summarising the three streams reviewed above: surface-sensitivity under semantics-preserving transformations, anisotropic embedding geometry, and high clone-detection recall on equivalent pairs. Two further recent strands sharpen where this contribution sits. First, the CodeXGLUE benchmark suite [28] standardised the evaluation of frozen and fine-tuned code encoders across clone detection, code search, and defect prediction, and is the setting in which CodeBERT’s high Type-1/Type-2 clone recall is reported; that recall-oriented framing is exactly what leaves the opposite-pair failure mode unmeasured. Second, execution-aware pre-training such as TRACED [29] responds to the same surface-sensitivity diagnosis we rely on, injecting execution traces, branch coverage, and runtime values into pre-training to expose dynamic behaviour the static objective misses; it is a learned, weight-modifying route to the behavioural signal our training-free probes lack, and so bounds our claims to the no-fine-tuning regime rather than to code encoders in general.

What none of these streams has done is: (a) construct a controlled, I/O-verified dataset of behaviourally divergent mutant pairs, categorised by mutation operator class; (b) measure the rate at which frozen CodeBERT embeddings assign falsely high cosine similarity to such pairs, across multiple embedding configurations and model variants; (c) break the failure down by mutation operator class to identify which structural families drive the confusion; or (d) test, under a no-fine-tuning constraint, whether a training-free structural correction can repair the failure, establishing instead that structural similarity and behavioural divergence dissociate on these pairs. These are the four contributions of the present work.

3 Methodology

The methodology is organised into three components. Section 3.2 describes the dataset construction, which underpins all three research questions. Starting from a single data source, Project_CodeNet_Python800, it applies an eligibility filter, equivalent-pair selection, and a controlled per-problem AST-based mutant-generation procedure with I/O classification across three categories (COD, ROR, and AI); the resulting pair counts and the analysis-set definition are given in Section 3.2.6. Sections 3.3 and 3.4 describe the analysis protocols for RQ1 (false-high-similarity measurement) and RQ2 (category-level confusion) respectively; both treat the pre-trained models as fixed probes and operate on the 2,617-pair analysis set. Section 3.5 describes the hybrid model (RQ3) and its evaluation, which uses a stratified 20/80 validation/test split of the same dataset. The three analysis streams share a common core of statistical methods: paired nonparametric tests with multiple-comparison correction, rank-based effect sizes, and 95% bootstrap confidence intervals ($B = 10,000$ resamples throughout). All experiments are conducted without any model fine-tuning. Implementation details (software versions, hardware, random seeds, and the artefact repository) are collected in the supplementary material (Supplementary Section S11).

Table 1 Comparison of the present study against closely related prior work. Symbols: ✓ = fully addressed; ◦ = partially addressed; – = not addressed. For the **Training-free correction** column, ✓ marks a parameter-free post-hoc transform (e.g. whitening, all-but-the-top), ◦ a correction that requires no backbone fine-tuning but still trains an auxiliary component (e.g. the normalising flow of BERT-flow), and – no such correction.

Study	Code-specific	I/O-verified labels	Mutant-pair focus	Multi-config sweep	Training-free correction
Ethayarajh (2019) [5]	–	–	–	◦	–
Li et al. (2020) [13]	–	–	–	◦	◦
Gao et al. (2021) SimCSE [6]	–	–	–	–	–
Jain et al. (2021) Contra-Code [7]	✓	–	–	–	–
Rabin & Alipour (2021) [23]	✓	–	–	–	–
Naik et al. (2022) [8]	✓	–	–	✓	–
Sonnekalb et al. (2022) [18]	✓	–	–	–	–
Abid et al. (2023) [20]	✓	–	–	–	–
Liu et al. (2023) Contra-BERT [15]	✓	–	–	–	–
Song et al. (2024) [21]	✓	◦	–	–	–
Ahmed et al. (2024) [19]	✓	–	–	–	–
This paper	✓	✓	✓	✓	✓ ^a

^a The training-free-correction column marks that the paper *tests* such a correction, not that it provides a working one: the result is a deliberate dissociation (a negative result), as Section 4.3 establishes.

3.1 Taxonomy of Behavioural-Divergence Mutation Operators

We define *mutant pair* operationally and in mutation testing terminology [4]: each pair consists of an original program and a *first-order mutant* produced by applying one mutation operator at one site. Every candidate mutant is executed against the problem’s oracle inputs and classified by comparison with the original (Section 3.2.4); a pair is retained in the dataset only when the mutant’s output diverges from the original on *every* usable oracle input (the pipeline label **killed**). This retention criterion is deliberately stricter than the standard kill criterion, under which a mutant is killed as soon as one test distinguishes it [4]: partially divergent candidates, although killed in the standard sense, are excluded so that the contrast class exhibits consistent rather than input-dependent divergence. Retained pairs are therefore I/O-verified, single-site, first-order mutants with divergence on every usable oracle input; the label asserts divergence on the tested inputs, not domain-wide logical complementarity across all possible inputs. Three mutant categories produce pairs under this definition.

- **Conditional Operator Deletion (COD):** A condition in an `if`, `while`, or `assert` statement, or a ternary expression condition, is inverted by toggling the presence of a unary `not`. In standard operator terms the category comprises Conditional Operator Insertion (a `not` is added; 766 of 779 pairs) and Conditional Operator Deletion (an existing `not` is removed; 13 pairs) [4]; the code COD is retained as the category label for consistency with the released dataset artefacts. With respect to the mutated condition, the transformed function evaluates the complementary branch behaviour. Example: a function that validates that an input is positive becomes one that accepts only non-positive input.
- **Relational Operator Replacement (ROR):** A relational operator is replaced with its logical complement: $< \leftrightarrow >=$, $<= \leftrightarrow >$, $> \leftrightarrow <=$, $>= \leftrightarrow <$, and $= \leftrightarrow !=$. This is a complement-restricted variant of the standard ROR operator, which in its unrestricted form replaces each relational operator with every alternative; we use only the complement because it guarantees that the mutated predicate negates the original at the mutation site. Each candidate modifies exactly one operator site. Chained comparisons are excluded because complementing one operator inside a chain does not produce the full complement of the chained predicate.
- **Algorithmic Inversion (AI):** The mutant reverses the direction or polarity of the original’s primary operation (a behavioural reversal, not a mathematical inverse in general), covering four templates: `min/max` inversion, arithmetic complement ($+ \leftrightarrow -$), sort direction reversal, and slice reversal. Of these, only the arithmetic complement corresponds to the standard Arithmetic Operator Replacement class; the other three are higher-level inversions in the spirit of method-call and constant replacement operators, not canonical operators in the mutation testing catalogue. We group all four as a single operational category. Example: a sort-ascending function becomes a sort-descending function; a function returning the first element becomes one returning the last.

The label *mutant pair* is used throughout as the study’s shorthand for this class.

A fourth category, Contract Violation (CV), was considered but excluded. CV involves negating a function’s precondition so that the mutated function operates on the complement of the original’s input domain. A structural pre-experiment analysis found that only 47 of 800 base functions (5.9%) contained an identifiable guard condition suitable for CV mutation. Competitive programming solutions are written under the assumption that inputs conform to problem constraints, so explicit precondition guards are rarely present. At 47 candidate pairs, CV would be underpowered for per-category statistical analysis; including it would introduce a category whose results cannot be compared fairly with COD, ROR, and AI. CV is therefore excluded from the taxonomy.

3.2 Dataset Construction

3.2.1 Source Dataset and Design Constraints

All source functions are drawn from **Project CodeNet Python800**, a curated benchmark subset of IBM Project CodeNet [10]. The subset contains 800 problems,

each with exactly 300 accepted Python submissions, giving 240,000 Python code samples in total. Every submission has been verified as correct by an automated online judge. The judge checks acceptance status against its internal test suite; separately, the transformation pipeline uses the public sample I/O pairs published with each problem statement to verify that accepted transformations produce behaviourally different outputs from their originals. Two accepted submissions to the same problem agree with the judge’s acceptance criterion on its hidden test suite; they are strong candidates for behavioural equivalence but are not equivalent by construction, particularly for problems that admit multiple valid outputs. Each candidate equivalent pair is therefore verified by execution against the assembled oracle inputs before acceptance, and its verification status is recorded (Section 3.2.3). A transformation applied to an accepted submission produces a function whose relationship to the original is controlled by the transformation rule and confirmed by execution on the same oracle inputs.

For each of the 800 problems, accepted Python submissions serve two purposes: constructing one equivalent pair per problem, and generating up to three mutant pairs per problem by applying single-site AST transformations per category (COD, ROR, AI). The two pair-construction pipelines draw from the same eligible pool, but the files selected for the equivalent pair are excluded from mutant pair generation for that problem; the full separation rule is described in Section 3.2.4.

A design constraint worth stating explicitly: Project_CodeNet_Python800 consists of competitive programming solutions, which are algorithmically focused, typically short, and written to pass specific test cases rather than for long-term maintainability. This differs from production code in both structural complexity and the prevalence of defensive programming patterns such as input guards. The dataset is a controlled evaluation environment, not a representative sample of production code. Generalisability implications are addressed in Section 6.

3.2.2 Submission Eligibility Criteria

The eligibility filter is applied once, before any pair construction. Its output, the *eligible pool*, is shared by both the equivalent-pair pipeline and the mutant-pair pipeline. A submission is eligible if and only if it satisfies all four of the following criteria simultaneously:

1. **Length.** Between 5 and 50 non-blank, non-comment lines. The lower bound excludes trivially short functions (single-expression solutions) that carry insufficient syntactic structure. The upper bound keeps functions short; the token-budget criterion below enforces CodeBERT’s 512-token context window directly.
2. **Parseability.** Parses without a `SyntaxError` under the Python `ast` module. Submissions using Python 3.9+ syntax (walrus operator `:=`, `match` statements, union-type hints) are also excluded at the transformation stage as a compatibility constraint of the AST pipeline.
3. **Token budget.** Encodes to at most 512 byte-pair encoding (BPE) tokens, measured with the `RobertaTokenizer` for `microsoft/codebert-base` including special tokens `[CLS]` and `[SEP]`.

4. **Near-duplicate exclusion.** Not flagged as a near-duplicate of any other submission for the same problem. Near-duplicate detection uses BPE unigram cosine similarity: any pair of submissions for the same problem with cosine > 0.99 are flagged; every submission appearing in at least one such pair is removed from the eligible pool for that problem. The threshold 0.99 was chosen to identify submissions that share all but a handful of tokens, which typically indicates copy-paste solutions rather than independently written code. Exclusion is applied at the individual-file level: only the specific near-duplicate submissions are removed, not the entire problem.

Filter outcomes.

After applying this filter, all 800 problems retained at least three eligible submissions, the minimum needed for one equivalent pair plus a disjoint mutant base, so no problem was excluded. The eligible pool contains 190,993 submissions in total (mean 238.7 per problem). Eligible submissions have a mean BPE token count of 174.5 (SD = 113.4, range 27–512) and a mean non-blank, non-comment line count of 14.2 (SD = 7.7, range 5–50); all fall within CodeBERT’s 512-token context window. Near-duplicate detection identified 97,326 near-duplicate pairs across 687 of the 800 problems (85.9%); the corresponding submissions were removed from the eligible pool per problem before any pair selection was attempted.

3.2.3 Equivalent Pairs

Pair search and acceptance.

For each problem, the pipeline enumerates all eligible submission pairs (i, j) where $i < j$, ordered by ascending $i + j$ then ascending i . This ordering prefers pairs drawn from earlier-indexed submissions (which are shorter on average) while still exploring all combinations. The search stops as soon as one pair is accepted; up to 500 pairs are tried per problem before the problem is excluded.

Both functions are executed in isolated subprocesses against every available oracle test, each execution bounded by a 5-second timeout; a pair that times out is retried once with a 15-second limit before the next pair is tried.

Oracle inputs combine two layers in order: the public sample I/O pairs from each problem’s AtCoder or AIZU statement (the primary source, deduplicated first), and Claude-generated extended tests (boundary, corner, and representative valid inputs). Generated inputs are not treated as ground truth: each is validated by executing accepted submissions and retained either as a fixed oracle pair (when a reliable expected output exists) or as an *agreement-only* case (for example, under multiple valid outputs), on which two programs are compared against each other. The merged oracle deduplicates on whitespace-stripped inputs, keeping the sample pair on collision. The HTML samples anchor the oracle to the judge’s published examples and the generated tests extend edge-case coverage; per-problem generation counts are in the supplementary material.

A pair is accepted with status `both_pass_verified_samples` if and only if both functions produce the expected output on every oracle test. For `agree_only` oracle tests, where no stored expected output exists, the pair is accepted if both functions

produce identical outputs on those inputs. A pair is skipped if either function fails a test, produces incorrect output, or times out persistently.

The acceptance statuses encode different verification strengths. Pairs with status `both_pass_verified_samples` are *oracle-verified*: both functions produced the expected output on every available fixed oracle test. Pairs with status `both_outputs_agree` are *agreement-verified*: both functions produced identical outputs on all usable oracle inputs in cases where fixed expected-output verification was not applicable, such as agreement-only generated inputs, problems with multiple valid outputs, or exact-output matching that proved too restrictive. Agreement verification is weaker evidence than oracle verification, since two solutions could in principle share a fault pattern, so all RQ1 headline metrics will additionally be reported on the oracle-verified subset of analysis-set equivalent pairs alone as a sensitivity check (732 of the 800 released pairs are oracle-verified; Section 3.2.6 defines the analysis set).

Coverage and statistics.

Every problem retained at least three eligible submissions after filtering (Section 3.2.2), and all 800 yielded a verified equivalent pair: 732 are oracle-verified (`both_pass_verified_samples`), 67 are agreement-verified (`both_outputs_agree`, used where fixed expected-output verification did not apply), and one is verified under a dedicated interactive simulation (`both_outputs_agree_interactive`, p03439). The `equiv_verification_status` column in the dataset CSV records the outcome for every problem.

Role in the study.

The equivalent pairs are not the primary subject of analysis; they establish the reference distribution that makes the failure on mutant pairs measurable. The 75th percentile of equivalent-pair cosine similarity sets the FHSR threshold, and AUC and JSD are defined relative to the equivalent-versus-mutant separation (Section 3.3.2). A mutant scoring in the upper quartile of the equivalent-pair distribution is operationally indistinguishable from a genuine equivalent: a false-high-similarity event.

3.2.4 Mutant Pairs

The mutant-pair pipeline operates over the same eligible pool as the equivalent-pair pipeline, but candidate base submissions are restricted to files whose filenames do not appear as `program_A_file` or `program_B_file` in the problem’s equivalent-pair record. The two pair types therefore draw on disjoint base programs within each problem. This disjointness is deliberate: AUC is a two-sample rank statistic that assumes the equivalent-pair and mutant-pair score sets are drawn independently, and reusing a program across both strata would couple the samples and bias the estimate. Disjoint base programs remove this coupling, so the primary comparison rests on two independent samples by construction; the cost is that it forecloses a within-anchor comparison, which we recover through the shared-anchor robustness analysis of Section 3.3.2. For each applicable category, the pipeline searches candidate base programs and mutation sites, applies one single-site AST transformation per candidate, executes original and mutant on the same oracle inputs, and classifies each candidate by the resulting

behaviour. Candidates are ranked by behavioural strength, and the strongest observed candidate per problem and category is retained (the full rule is given below). A retained mutant pair is therefore not a syntactic artefact; it is an I/O-verified, behaviourally divergent first-order mutant whose divergence holds on every usable oracle input.

Verification oracle.

Classification uses the same merged per-problem oracle as equivalent-pair construction (Section 3.2.3) but compares the mutant directly against the original program rather than against stored expected outputs, since some inputs are agreement-only and some problems admit multiple valid outputs. Outputs are compared after whitespace normalisation and numeric tolerance (10^{-6}); inputs that cannot be reliably compared (execution failure, timeout, missing output) are excluded from the usable count.

Behavioural classification and retention.

Each candidate mutant receives one of four labels from the program-to-program comparison: **killed** if its output diverges from the original on every usable oracle input; **neither** if it diverges on some usable inputs and agrees on others; **equivalent** if it agrees with the original on every usable input; and **undetermined** if the available execution evidence is insufficient for a reliable classification. During search, candidates are ranked **killed** > **neither** > **equivalent** > **undetermined**; if the pipeline first finds a partially divergent candidate, it continues searching within the same problem and category for a **killed** candidate, so the retained mutant represents the strongest behavioural contrast observed under the available oracle. The dataset retains only pairs labelled **killed**. This killed-only retention produces a stricter and more interpretable contrast class: every retained mutant diverges consistently rather than on an input-dependent subset, and candidates that the oracle cannot distinguish (or can distinguish only partially) are dropped rather than assumed divergent. Here **neither** candidates, killed in the standard mutation-testing sense, are excluded by design (Section 3.1). The full classified mutant file, including non-retained labels, is preserved as an intermediate artefact for provenance.

Transformation categories.

The three categories (COD, ROR, AI) are defined in Section 3.1. Operationally, COD enumerates condition sites in source order; ROR applies true logical operator complements (a boundary-perturbation design, replacing < with <=, was considered and rejected because it does not produce reliable behavioural divergence across the input domain); AI applies four inversion templates in fixed priority order: sort direction, extremum selection, traversal direction, and arithmetic complement. In every category, enumerated candidates are classified and ranked as described above, and each category retains at most one pair per problem.

Search strategy.

The pipeline processes all 800 problems in a single unified pass. For each problem, applicable transformation categories are determined from a pre-analysis of the eligible submission pool; a problem is marked applicable for a category only if at least one

eligible submission contains a suitable transformation site. The pre-analysis found at least one applicable category in every problem and all three in 799 of 800; applicability is an upper bound, since it guarantees neither successful execution nor classifiable behavioural divergence. For each applicable category, the pipeline searches candidate base programs and mutation sites, classifying and ranking candidates as described above. Problem–category combinations can remain without a classified pair for three reasons observed in the run: original-program failures on the oracle inputs, absence of usable mutation sites among the searched bases, and candidate behaviour that could not be reliably determined. The 117 problem–category combinations that yielded no classified pair comprise 46 original-program oracle failures, 38 cases with no usable mutation site among the searched bases, and 33 undetermined candidate sets.

The complete generation run processed all 800 problems and produced 2,282 classified mutant pairs (799 COD, 799 ROR, and 684 AI). Filtering to the `killed` label retained 1,833 mutant pairs (COD: 779, ROR: 775, AI: 279) spanning 784 problems. Of the 16 problems contributing no retained pair, 15 had classified candidates that never met the killed criterion and one (p03046) produced no classified candidate at all, as no attempted mutation yielded a runnable mutant for it. The final dataset therefore contains 800 equivalent pairs and 1,833 I/O-verified mutant pairs, for a total of 2,633 pairs. Table 2 reports retained counts and the transformation-site breakdown per category; the full behavioural-label distribution of the 2,282 classified candidates appears in the supplementary material (Table S10).

Table 2 Killed mutant pairs by category and transformation site. Accepted-column percentages are of all killed pairs; Pairs/% columns are within-category.

Category	Accepted	Transformation	Pairs	%
COD	779 (42.5%)	condition [†]	766	98.3%
		not condition [†]	13	1.7%
ROR	775 (42.3%)	<code>== → !=</code>	549	70.8%
		<code>< → >=</code>	57	7.4%
		<code>> → <=</code>	56	7.2%
		<code><= → ></code>	44	5.7%
		<code>>= → <</code>	38	4.9%
		<code>!= → ==</code>	31	4.0%
AI	279 (15.2%)	<code>+= ↔ -=</code>	167	59.9%
		<code>min ↔ max</code>	105	37.6%
		<code>reverse=False → True</code>	5	1.8%
		<code>[:] ↔ [::-1]</code>	2	0.7%
Total	1,833 (100.0%)			

Note. [†] COD “condition”: a `not` was added; “not condition”: an existing `not` was removed.

3.2.5 Validation Protocol

To assess label reliability, two annotators, both external to the author list and each holding a PhD in computer science with software-engineering research and teaching experience, independently labelled a stratified 10% sample of the analysis set ($n = 263$: 80 equivalent and 183 mutant pairs, COD 77, ROR 78, AI 28), stratified by pair type and by transformation site under a fixed seed; sampling from the analysis set ties the reliability evidence to the pairs on which the RQ1–RQ3 claims rest. Each annotator saw the two programs and the oracle evidence (verification inputs and each program’s outputs) with no gold label, category, mutation site, or program-role information, program order randomised, with those fields held in a separate key. Annotators assigned one of the four pipeline labels: **equivalent** (identical on every usable oracle input), **killed** (divergent on every usable input), **neither** (divergent on some inputs and agreeing on others), or **undetermined**. Because the mutant strata contain only pipeline-killed pairs, the annotation audits the retention label itself; **killed** is used in the pipeline’s strict sense (Section 3.1), not the standard one-distinguishing-test sense.

Agreement is reported as raw percentage agreement, Cohen’s κ , and prevalence-adjusted κ (PABAK) against a pre-registered target of $\kappa \geq 0.80$; all three are reported because κ is sensitive to the label skew of the **killed**-dominated mutant strata, where per-stratum κ collapses despite near-perfect raw agreement. The two annotators agree on 254 of 263 pairs (raw 96.6%), giving pooled $\kappa = 0.925$ (bootstrap 95% CI [0.877, 0.968]; chance agreement 0.544) and PABAK = 0.932, above target; per-stratum raw agreement is high throughout (Table 3). All nine disagreements are **killed**-versus-**neither** (seven) or **neither**-versus-**equivalent** (two); each is a **neither** judgement resting on agreement observed only on inputs outside the oracle set, so it concerns oracle coverage rather than the labelling of the supplied evidence (Section 6), and joint adjudication against the oracle-scoped protocol altered no label. The stratification quotas, blinding procedure, confusion matrix, and per-rater accuracy against the pipeline labels appear in the supplementary material.

Table 3 Inter-rater reliability sample and per-stratum agreement. Per-stratum κ is omitted because it is uninformative when one label dominates a stratum (see text).

Pair type	Dataset pairs	IRR sample	Raw agreement	Disagreements
Equivalent pairs	784	80	97.5% (78/80)	2
COD	779	77	97.4% (75/77)	2
ROR	775	78	97.4% (76/78)	2
AI	279	28	89.3% (25/28)	3
Total	2617	263	96.6% (254/263)	9

Note. The IRR sample is stratified by pair type over the analysis set, giving 263 pairs in total (10.0% of the 2,617-pair analysis set; the equivalent stratum is drawn slightly above its proportional share to represent all three verification-provenance groups). Pooled over all 263 pairs, Cohen’s $\kappa = 0.925$ (chance agreement 0.544) and prevalence-adjusted κ (PABAK) = 0.932. All nine disagreements are **killed**-versus-**neither** (seven) or **neither**-versus-**equivalent** (two).

3.2.6 Dataset Composition and Pre-Experiment Calibration

Figure 1 shows the composition of the analysis set, on which all RQ1–RQ3 analyses operate: the 784 problems that contribute both an equivalent pair and at least one retained mutant pair, giving 784 equivalent pairs and 1,833 mutant pairs (2,617 pairs, a mean of 3.34 per problem). Within a problem, the equivalent pair and the mutant pairs draw on disjoint base submissions; the mutant categories retained for a problem may share a base submission or use different eligible bases, depending on which candidate yields the strongest classification in each category, so a problem contributes one equivalent pair and up to three mutant pairs. The full released dataset adds the equivalent pairs of the 16 problems with no retained mutant, for 2,633 pairs across all 800 problems; those 16 pairs are flagged and excluded from analysis (detailed at the end of this section).

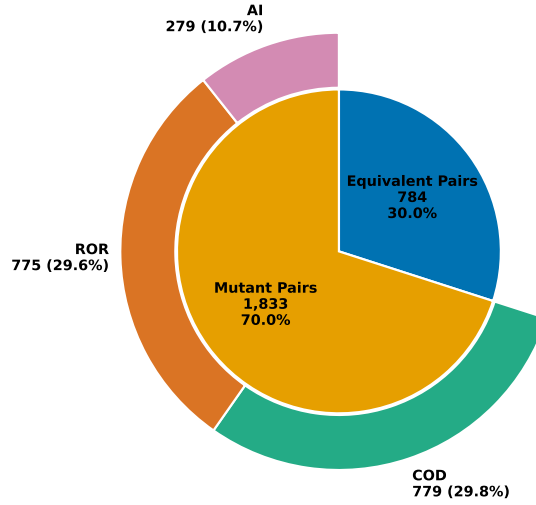


Fig. 1 Composition of the 2,617-pair analysis set by pair type (the 16 equivalent pairs without a retained mutant are excluded; see text).

The category distribution is not a deliberate sampling design, nor is it driven by site availability: at least one COD and one ROR site exists in all 800 problems and at least one AI site in 799 (99.9%), with 799 problems applicable under all three categories (upper-bound yield 2,399 pairs). Classification succeeded for 2,282 combinations (95.1% of the upper bound), so the dataset shape is set almost entirely by the killed-only retention rule, which keeps 1,833 of 2,282 classified pairs (80.3%). Retention differentiates the categories sharply: 779 of 799 classified COD and 775 of 799 classified ROR pairs are retained (97.4% and 96.9% of problems), whereas AI retains only 279 of 684 (34.9% of problems). AI’s shortfall reflects partial rather than absent divergence: of the 449 classified-but-not-retained candidates, 405 are AI (378 **neither**, 27 **equivalent**, the latter no-ops such as reversing an already-sorted sequence) against 20 COD and 24 ROR, all **neither**; the full label distribution is in the supplementary material (Table S10). Across the 800 problems, 274 (34.2%) contribute retained pairs

in all three categories, 501 (62.6%) in two, 9 (1.1%) in one, and 16 (2.0%) in none, giving a mean of 2.34 retained mutant pairs among the 784 analysis-set problems (median 2, maximum 3). This per-category distribution is the basis for the RQ2 analysis.

Why the analysis set excludes 16 problems. The 16 problems with no retained mutant are held out so both strata span the same 784 problems, keeping the equivalent and mutant score distributions matched on problem composition; their lack of any killed mutant also marks them as where the oracle separates behaviour least sharply. The 16 equivalent pairs remain in the released dataset, flagged by an `in_analysis_set` column, so the full 800-problem artefact stays available for reuse.

Table 4 reports two properties of the dataset that are central to the study’s motivation. First, every pair fits within CodeBERT’s 512-token context window; this is guaranteed by the token-budget eligibility criterion (Section 3.2.2). Second, and more critically, the unigram Jaccard overlap between the two functions in a pair differs sharply by pair type: equivalent pairs are independently written solutions with genuine stylistic diversity (mean Jaccard 0.36), whereas each mutant is a single-site transformation of its base in which one operator, condition, or template changes while every other token remains identical (mean Jaccard 0.79). This asymmetry is the surface-similarity signature that makes the hypothesised embedding failure in RQ1 non-trivial: a model that relies primarily on token overlap would assign *higher* similarity to mutant pairs than to equivalent ones, inverting the correct semantic ordering. The mutant strata are near-uniform in overlap (COD 0.80, ROR 0.79, AI 0.78), so the asymmetry is a property of single-site mutation in general rather than of any one operator family. The 800-pair cross-problem control used to validate the FHSR threshold (Section 3.3.2) is not part of the dataset and is not shown in Figure 1.

Table 4 Function-level characteristics per stratum on the 2,617-pair analysis set. Mean lines and BPE token counts are per pair, averaged across both functions; BPE tokens use the `microsoft/codebert-base` tokenizer with special tokens included. Jaccard overlap is the unigram BPE-token-set Jaccard between the two functions.

Pair type	Mean lines (SD) [†]	Mean BPE tokens (SD)	Jaccard overlap
Equivalent	13.6 (6.9)	166.1 (101.9)	0.36 (0.12)
Mutant pairs	15.8 (8.3)	202.2 (125.3)	0.79 (0.15)
<i>COD</i>	15.7 (8.3)	200.2 (124.2)	0.80 (0.15)
<i>ROR</i>	15.9 (8.3)	203.6 (125.3)	0.79 (0.14)
<i>AI</i>	16.0 (8.3)	204.0 (128.1)	0.78 (0.15)
Total	15.2 (8.0)	191.4 (119.9)	0.66 (0.24)

[†] Non-blank, non-comment lines. The mutant differs from its base by at most one line, so base and mutant counts are effectively identical.

Pre-experiment similarity calibration.

Before the RQ1 embeddings, a fast BPE-frequency proxy confirmed the threshold design is not arbitrary: the per-problem mean pairwise BPE cosine between eligible submissions (0.730, SD 0.093) sits well above the cross-problem null (0.664, SD 0.193;

19,900 inter-problem pairs), with the per-problem 75th percentile averaging 0.8498. These proxy figures inform only the threshold design; in RQ1 the 75th-percentile threshold is computed separately for each configuration from its own score distribution over the 784 analysis-set equivalent pairs, so FHSR stays comparable across configurations despite differing raw scales, whereas a single shared threshold would conflate miscalibration with discrimination failure.

3.3 RQ1: Embedding Extraction and Discrimination Analysis

3.3.1 Embedding Models and Extraction Configurations

We extract embeddings from three models, all loaded from the Hugging Face Transformers library with frozen weights and no task-specific fine-tuning.

CodeBERT (`microsoft/codebert-base`): the primary subject of this study. We extract representations at layers 6, 9, and 12 of the 12-layer encoder. Layer 12 is the final output; layers 6 and 9 are included because prior work on BERT-family probing [5] has shown that representations become increasingly anisotropic in upper layers: cosine similarities cluster near 1 regardless of semantic content, which predicts that discrimination failure will be more severe at layer 12 than at layers 6 or 9. We test whether this theoretical prediction holds for the mutant-pair discrimination task. For each layer we apply two pooling strategies:

- **CLS pooling:** The representation of the [CLS] token at the chosen layer, which CodeBERT’s pre-training treats as a sequence-level summary.
- **Mean pooling:** The mean of all non-padding token representations at the chosen layer, which averages out positional biases and is commonly used in embedding-based retrieval.

This yields six embedding configurations for CodeBERT (3 layers $\times$ 2 pooling strategies).

GraphCodeBERT (`microsoft/graphcodebert-base`) [2] and **UniXcoder** (`microsoft/unixcoder-base`) [3]: used as comparative baselines using their respective [CLS] representations at the final layer only. We do not sweep layers or pooling strategies for these models because our research question concerns CodeBERT specifically; the baselines serve to establish whether the discrimination failure is unique to CodeBERT’s pre-training scheme or also present in two related encoder-only models at their standard setting (a two-model probe, not a characterisation of the family as a whole). Using the final-layer CLS representation is the standard evaluation setting for both models as reported in their original papers.

In total this yields eight configurations across three models. The CodeBERT configuration achieving the highest AUC on the 2,617-pair analysis set is designated the best configuration and used as the fixed probe for RQ2 and RQ3; ties are broken by lowest FHSR (computed against each configuration’s own threshold, Section 3.3.2). If a baseline model outperforms every CodeBERT configuration, that result is reported in RQ1, but RQ2 and RQ3 remain on CodeBERT, the model under diagnosis. RQ1 uses the analysis set of 2,617 pairs (784 equivalent and 1,833 I/O-verified mutant pairs; Section 3.2.6 and Figure 1) for all eight configurations. The dataset is not split for

RQ1/RQ2 analysis; the stratified 20/80 validation/test split described in Section 3.5 applies only to the RQ3 hyperparameter selection.

Additional frozen probes and training-free corrections.

Two analyses added during revision probe whether the failure is specific to the model family or removable without fine-tuning; both are reported with the RQ1 results (Section 4.1). First, we evaluate CodeSage [30], a frozen encoder whose second pre-training stage is contrastive, as an additional probe under the same protocol and metrics, using the model’s documented sentence-transformers embedding with the end-of-sequence token appended, as it requires; this separates the effect of the pre-training objective from that of the encoder family. Second, we post-process each of the eight configurations’ embeddings with two standard, training-free isotropy transforms, whitening [31] and all-but-the-top [32], and recompute the discrimination metrics. Whitening has been applied to code embeddings before, for semantic code search [14]; we apply it here to the opposite-pair discrimination task. Each transform is fit on a disjoint background of 4,000 eligible functions that appear in no analysis pair, so it never sees the evaluation pairs; neither transform uses labels or modifies model weights. These probes carry no fine-tuning; they test the objective-change and geometry-correction explanations of the failure directly, without altering the no-fine-tuning constraint of RQ1.

3.3.2 Discrimination Metrics

For each configuration we compute pairwise cosine similarity over all pairs and evaluate discrimination with three complementary metrics, each reported with 95% bootstrap confidence intervals ($B = 10,000$).

1. **AUC-ROC.** The probability that a randomly chosen equivalent-pair scores above a randomly chosen mutant pair: 0.5 is chance, 1.0 perfect separation, and < 0.5 systematic inversion (mutant pairs ranked above equivalent pairs). We use $\text{AUC} > 0.8$ as a practical reference point only; all values are reported in full.
2. **Jensen-Shannon Divergence (JSD).** The symmetric divergence between the equivalent- and mutant-pair score distributions (base-2 logarithm, so $\text{JSD} \in [0, 1]$). Because anisotropy compresses scores near 1 (Section 4.1), we bin on the quantiles of the pooled scores (50 equal-mass bins per configuration) rather than a fixed grid. JSD measures the *magnitude* of separation; read with AUC, which gives its *direction*, a high JSD with $\text{AUC} < 0.5$ means two well-separated distributions ordered the wrong way.
3. **False-High-Similarity Rate (FHSR).** The proportion of mutant pairs whose score exceeds the 75th percentile of the same configuration’s equivalent-pair distribution. The per-configuration threshold makes FHSR a quantile-referenced rate comparable across configurations despite differing raw scales (Section 3.2.6); the 75th percentile marks the boundary above which a downstream filter is most likely to treat a pair as equivalent. FHSR at the 50th and 90th percentiles is reported as a sensitivity analysis.

Random-pair control. To check that each threshold separates semantically unrelated code, we compute a control FHSR over 800 cross-problem pairs (each combining eligible submissions from two different problems, fixed seed, files disjoint from the dataset), which have no semantic relationship by construction. A control FHSR far below the mutant-pair FHSR confirms the threshold identifies the failure signal rather than flagging arbitrary pairs; it is reported alongside the main results, not entered into the statistical tests.

Anchored robustness comparison. The primary comparison contrasts independent samples: equivalent-pair scores $\cos(\text{soln}_i, \text{soln}_j)$ against mutant-pair scores $\cos(\text{soln}_k, \text{mutant of soln}_k)$, which the disjoint base-program design (Section 3.2.4) keeps independent as AUC’s rank statistic assumes. As a robustness check we add a within-anchor comparison that holds the anchor soln_k fixed, scoring it against a behaviourally equivalent partner (positive) and against its own killed mutant (hard negative); a failure that persists here is not an artefact of the disjoint sampling. Each positive is verified directly by execution on the merged oracle (agreement on every usable input), yielding 1,098 verified anchors of 1,118 candidates; the 20 unverified bases (which skew longer) are excluded, and the resulting selection bias, confined to this check, is characterised in the supplementary material. This anchored comparison does *not* control for surface overlap, since the mutant is more lexically similar to the anchor than the independent equivalent partner is; that confound is addressed directly by the RQ2 token- and AST-overlap analyses (Sections 3.4.2 and 3.4.3).

Token-overlap-matched comparison. As a further surface control we bin pairs by unigram Jaccard and subsample the larger class to the smaller within each bin (coarsened exact matching), equalising the overlap distribution across classes; AUC on the matched subset isolates the label’s contribution beyond lexical near-duplication. The common-support region (the Jaccard bands where both classes are populated) and the resulting matched-set size are reported with the result; the matched AUC is reported with its 95% bootstrap confidence interval so that “below chance” is backed by an interval, not a point estimate. All are reported with the RQ1 results (Section 4.1).

3.3.3 Statistical Testing Protocol

AUC summarises the overall discrimination ability of a configuration, and comparisons between configurations are therefore conducted on AUC directly. For each pair of configurations, we compute the AUC difference under paired bootstrap resampling ($B = 10,000$; each resample draws dataset pairs with replacement and evaluates both configurations on the same resample, preserving the pairing) and report the confidence interval of the difference; a difference whose interval excludes zero is treated as significant. We apply Bonferroni correction for the $\binom{8}{2} = 28$ pairwise comparisons by widening the intervals to the $1 - 0.05/28$ confidence level (corrected threshold $\alpha' = 0.05/28 \approx 0.0018$). As a complementary check on score-level behaviour, we also apply the two-tailed paired Wilcoxon signed-rank test (α' as above) to the per-pair cosine scores, pairing observations by dataset pair index. The Wilcoxon test detects a systematic score shift between configurations, which is related to but distinct from a discrimination difference: a uniform shift moves every score and leaves AUC unchanged, so the two analyses answer different questions and are reported separately.

Effect sizes for the paired score comparisons are reported as the matched-pairs rank-biserial correlation. Cross-model comparisons (e.g., a CodeBERT configuration versus GraphCodeBERT) are interpreted with caution, as the two model families differ in pre-training data and architecture beyond the pooling strategy.

Minimum detectable effect. A minimum detectable effect analysis at the corrected significance level $\alpha' = 0.0018$ and target power $1 - \beta = 0.80$ will be reported with the results, expressed in the same units as the reported statistics (AUC difference for the bootstrap comparisons; matched-pairs rank-biserial correlation for the Wilcoxon checks). Effects below the detectable threshold will be reported as not significant without inference of equivalence.

3.4 RQ2: Category-Level Confusion Analysis

3.4.1 Per-Category False-High-Similarity Analysis

We compute FHSR separately for each of the three transformation categories (COD, ROR, AI), using the single embedding configuration that achieves the highest overall AUC in RQ1. Fixing to one configuration ensures the per-category analysis reflects the model’s best-case behaviour; categories that are still confused under the best configuration are the ones that present the most fundamental challenge. The FHSR threshold is the best configuration’s 75th-percentile threshold over the 784 analysis-set equivalent pairs, identical to that configuration’s threshold in RQ1, so that per-category rates are directly comparable. We report per-category FHSR values with 95% bootstrap confidence intervals ($B = 10,000$ resamples). Differences between categories are tested with pairwise two-proportion tests across the three category pairs, Holm-corrected; the unequal category sizes (COD: 779, ROR: 775, AI: 279) give the AI estimate markedly less precision than the other two, which the confidence intervals make explicit.

3.4.2 Token Overlap Analysis

We define a pair as *high-confusion* if its cosine similarity (under the best RQ1 configuration) exceeds the 75th percentile FHSR threshold established in RQ1. For all mutant pairs, we compute token-level Jaccard overlap between the two functions after sub-word tokenisation via CodeBERT’s BPE tokeniser, then use the high-confusion label as the binary outcome in the regression:

$$J_{\text{unigram}} = \frac{|T_1 \cap T_2|}{|T_1 \cup T_2|}, \quad J_{\text{bigram}} = \frac{|B_1 \cap B_2|}{|B_1 \cup B_2|} \quad (1)$$

where T_i and B_i are the sets of unigram and bigram tokens of function f_i respectively. We fit a logistic regression model predicting the binary confusion label (high vs. low confusion) from J_{unigram} and J_{bigram} as predictors. The model is fit on 80% of the mutant pairs and assessed using the area under the ROC curve on the held-out 20% of mutant pairs (sampled independently of the RQ3 validation/test split), and the pseudo- R^2 (McFadden) is reported; fitting and evaluation use the same population because the high-confusion outcome is defined relative to the equivalent-pair distribution and is meaningful only for mutant pairs. The analysis is explanatory rather than

predictive: the outcome is a thresholded function of cosine similarity, and the predictors correlate with cosine by construction, so the regression quantifies how much of the confusion signal token overlap accounts for. A high AUC and large coefficient on token overlap would support the hypothesis that surface similarity drives embedding confusion.

3.4.3 AST Overlap Analysis

We parse each function with the `tree-sitter` Python library to obtain its abstract syntax tree (AST). For every mutant pair, we compute the proportion of shared subtree patterns up to depth 3:

$$\text{ASTOverlap}(f_1, f_2) = \frac{|\mathcal{S}_1^{\leq 3} \cap \mathcal{S}_2^{\leq 3}|}{|\mathcal{S}_1^{\leq 3} \cup \mathcal{S}_2^{\leq 3}|} \quad (2)$$

where $\mathcal{S}_i^{\leq 3}$ is the multiset of depth-bounded subtree patterns of f_i : one pattern rooted at each node of the tree and truncated to three levels. The patterns are serialised over node types and retain the operator and punctuation tokens that the COD and ROR transformations alter, so that operator-level changes are reflected in the overlap. Because the metric is defined over node types, a mutation that changes only an identifier (notably the `min/max` inversions in AI) leaves the overlap at 1.0; this blindness is quantified in Section 4.2 and is the reason the token-level analysis is reported alongside the structural one. We limit to depth 3 because shallow subtrees (statements, expressions, operators) capture the structural patterns most likely to be shared between behaviourally divergent mutant pairs; deeper subtrees diverge rapidly even for small mutations. ASTOverlap distributions are compared between high-confusion and low-confusion mutant pairs; without the low-confusion comparison group, no contrast could be drawn. This analysis is complementary to the token-overlap analysis: high token overlap with low AST overlap would suggest the confusion is purely lexical; high token overlap with high AST overlap would indicate that the structural similarity is also preserved, which is the more serious failure mode for any similarity model that lacks a structural component.

3.5 RQ3: Hybrid Similarity Model

3.5.1 HybridSim Model Definition

Let $\mathbf{e}_1, \mathbf{e}_2$ be the CodeBERT embeddings of two functions under the best-performing CodeBERT configuration identified in RQ1, and let $d_{\text{AST}}(f_1, f_2) \in [0, 1]$ be the normalised AST edit distance computed using the APTED tree-edit-distance algorithm [27] as implemented in the `apted` library. The distance is normalised by the sum of the sizes (number of nodes) of the two trees, so that $d_{\text{AST}} = 0$ for identical trees and $d_{\text{AST}} \rightarrow 1$ for trees that share no nodes. We define:

$$\text{HybridSim}(f_1, f_2) = \alpha \cdot \cos(\mathbf{e}_1, \mathbf{e}_2) + (1 - \alpha) \cdot (1 - d_{\text{AST}}(f_1, f_2)) \quad (3)$$

where $\alpha \in \{0.3, 0.5, 0.7\}$ controls the relative weight of the embedding signal versus the structural signal. When $\alpha = 1$ the model reduces to standard cosine similarity;

when $\alpha = 0$ it reduces to AST-based structural similarity alone. We sweep three values to cover low, balanced, and high embedding weight scenarios. No parameters are learned; the model is fully deterministic given α .

Directional expectation.

The direction in which the structural term operates on this dataset is worth stating before the results, because it determines what the hybrid can and cannot do here. A mutant pair differs from its base at a single site, so its normalised AST edit distance is near zero and the structural similarity term $1 - d_{\text{AST}}$ is near one; equivalent pairs are independently written and structurally diverse, so the same term is lower for them. On this dataset, Eq. (3) therefore raises mutant-pair scores relative to equivalent-pair scores for any $\alpha < 1$: the structural term follows the same surface-proximity gradient as the token-overlap asymmetry documented in Section 3.2.6, and no convex combination of cosine and structural similarity is expected to repair the polarity failure measured in RQ1. The evaluation is designed around exactly this point: it tests whether structural similarity and behavioural equivalence dissociate on mutant pairs, and it quantifies what the structural term costs when its signature is inverted relative to the clone-screening setting the correction was designed for. That deployment setting, where false positives typically combine token overlap with divergent overall structure, is the converse case and is left to future work.

3.5.2 Discrimination Evaluation on the Controlled Dataset

The analysis set of 2,617 pairs is split into a validation set (20%, $n = 523$ pairs) and a test set (80%, $n = 2,094$ pairs). Stratification is performed over the joint (category, label) strata using a fixed random seed (seed = 42), so that each category and label type is proportionally represented in both sets. The α value is selected on the validation set by maximising AUC and fixed before any evaluation on the test set, preventing information leakage. All results reported for RQ3 are computed on the held-out test set only.

We evaluate four models on the test set to isolate the contribution of each component:

1. **Cosine** ($\alpha = 1$): CodeBERT cosine similarity alone. This is the baseline the paper diagnoses in RQ1.
2. **JaccardPenalty**: Eq. (3) with the unigram Jaccard distance $1 - J_{\text{unigram}}(f_1, f_2)$ substituted for d_{AST} , so the structural similarity term equals J_{unigram} , evaluated at the same α values. This tests whether a simpler token-overlap signal achieves comparable results without requiring AST parsing.
3. **ASTOnly** ($\alpha = 0$): Normalised AST structural similarity alone, with no embedding component. This establishes whether the structural signal is sufficient on its own.
4. **HybridSim** ($\alpha \in \{0.3, 0.5, 0.7\}$, best selected on validation set): The full model combining CodeBERT cosine similarity with normalised AST edit distance.

The JaccardPenalty and ASTOnly baselines are necessary to demonstrate that the AST-edit-distance component of HybridSim contributes beyond what a simpler structural proxy would achieve. If JaccardPenalty matches HybridSim, the AST computation overhead is not justified. If ASTOnly is competitive, the embedding component adds no value in this setting.

We evaluate all four models using the following criteria:

- **Primary criterion:** AUC improvement on the equivalent-vs-mutant discrimination task. An improvement of at least 0.05 AUC points over the baseline is considered practically meaningful. This threshold corresponds to moving from, for example, $\text{AUC} = 0.70$ to $\text{AUC} = 0.75$, a shift that downstream systems operating at a fixed decision threshold would experience as a non-trivial reduction in false-positive rate. This threshold is treated as an operational definition for this study; all AUC values are reported in full regardless of whether they cross it.
- **Secondary criterion:** FHSR on mutant pairs, computed for each model against that model’s own 75th-percentile threshold over the equivalent-pair scores (quantile-matched thresholds). Because each threshold is a quantile of the model’s own equivalent-pair distribution, the equivalent-pair reference rate is 25% for every model by construction, so an FHSR reduction cannot be obtained by suppressing equivalent-pair scores wholesale; comparing one model’s scores against another model’s threshold would conflate scale differences with discrimination differences.

Differences in per-pair similarity score distributions between HybridSim and the cosine baseline are assessed using the paired Wilcoxon signed-rank test (two-tailed, $\alpha = 0.05$) with effect size reported as the matched-pairs rank-biserial correlation. AUC differences between models are assessed using paired bootstrap confidence intervals ($B = 10,000$ resamples); a difference whose 95% CI excludes zero is considered statistically significant.

4 Results

4.1 RQ1: Embedding Discrimination Results

Table 5 reports AUC, JSD, and FHSR for all eight embedding configurations. Bootstrap 95% confidence intervals are based on $B = 10,000$ resamples. The cross-problem random-pair control FHSR is reported per configuration in the final column to validate threshold calibration; it is given per configuration rather than as a single figure because it ranges from near zero (CodeBERT CLS, where the 75th-percentile threshold cleanly separates semantically unrelated code) to one (the baseline models, where it does not). Figure 2 shows the per-configuration score distributions underlying these metrics.

Every configuration fails, and the failure is an inversion rather than mere noise. The AUC for separating equivalent from mutant pairs is far below the 0.5 chance level in all eight cases, from 0.017 (UniXcoder) to 0.090 (CodeBERT CLS, layer 12), so the mutant pair is scored above the equivalent pair in the large majority of comparisons. CodeBERT CLS at layer 12 is the strongest CodeBERT configuration on AUC and is also the lowest on FHSR (0.870); it is designated the best configuration and used as the

Table 5 RQ1 discrimination results across the eight embedding configurations (metrics defined in Section 3.3.2; arrows give the favourable direction). **Ctrl-FHSR** is the cross-problem random-pair FHSR against each configuration’s own 75th-percentile threshold. Bracketed values are 95% bootstrap confidence intervals ($B = 10,000$).

Model	Config	AUC $\uparrow$	JSD $\uparrow$	FHSR $\downarrow$	Ctrl-FHSR $\downarrow$
CodeBERT	CLS, L6	0.054 [0.046, 0.063]	0.616	0.927 [0.914, 0.940]	0.000
	CLS, L9	0.037 [0.031, 0.044]	0.676	0.957 [0.941, 0.969]	0.000
	CLS, L12	0.090 [0.079, 0.101]	0.474	0.870 [0.849, 0.892]	0.026
	Mean, L6	0.058 [0.050, 0.066]	0.619	0.915 [0.900, 0.931]	0.410
	Mean, L9	0.051 [0.044, 0.060]	0.640	0.931 [0.917, 0.944]	0.426
	Mean, L12	0.057 [0.048, 0.065]	0.616	0.919 [0.904, 0.934]	0.287
GraphCodeBERT	CLS, L12	0.037 [0.030, 0.043]	0.691	0.948 [0.937, 0.958]	1.000
UniXcoder	CLS, L12	0.017 [0.013, 0.021]	0.796	0.978 [0.971, 0.985]	1.000

fixed probe for RQ2 and RQ3, with no tie to break. The layer-anisotropy prediction of Section 3.3.1 (that the final layer would be worst) is not borne out for the CLS configurations: among them layer 12 discriminates best, not worst. A plausible reading is that the final-layer [CLS] vector is the representation CodeBERT’s pre-training optimises as a sequence-level summary, so it carries marginally more class-separating signal than the layer-6 and layer-9 [CLS] vectors even though raw cosine similarities are more compressed at the final layer; we report this as an observation rather than a mechanism, since all six CodeBERT configurations remain far below chance and the spread among them (AUC 0.037 to 0.090) is small in absolute terms. The JSD values are high (0.47 to 0.80) while the AUC values are near zero, which is the signature of two well-separated score distributions ordered in the wrong direction: the model assigns the two classes distinct similarity ranges, with the mutant range above the equivalent range. The control FHSR confirms the thresholds are calibrated for the CodeBERT CLS configurations (near zero, so semantically unrelated code is correctly placed below the threshold) while the baseline thresholds do not separate unrelated code (control FHSR 1.000), so the baseline rows are read on AUC rather than FHSR.

The percentile and provenance sensitivity analyses leave the picture unchanged. FHSR falls as the threshold rises but stays high throughout: across the eight configurations it spans 0.956–0.992 at the 50th percentile, 0.870–0.978 at the 75th (Table 5), and 0.769–0.957 at the strict 90th, so the failure is not an artefact of the 75th-percentile choice. Restricting the equivalent reference distribution to the 728 oracle-verified analysis-set pairs (Section 3.2.3) moves every configuration’s AUC by at most 0.001 and its FHSR by at most 0.005, so the 55 agreement-verified equivalent pairs do not drive the result.

The shared-anchor robustness comparison (Section 3.3.2) confirms the failure within a single anchor. For each of the 1,098 verified anchors we score the anchor against its behaviourally equivalent partner (the positive) and against its own killed mutant (the negative); Table 6 reports the result. The anchored AUC stays far below 0.5 for every configuration (0.017–0.079), and the mutant outscores the equivalent partner on average in every case, so the inversion is not an artefact of the disjoint-sampling structure of the primary comparison. The anchored range (0.017–0.079)

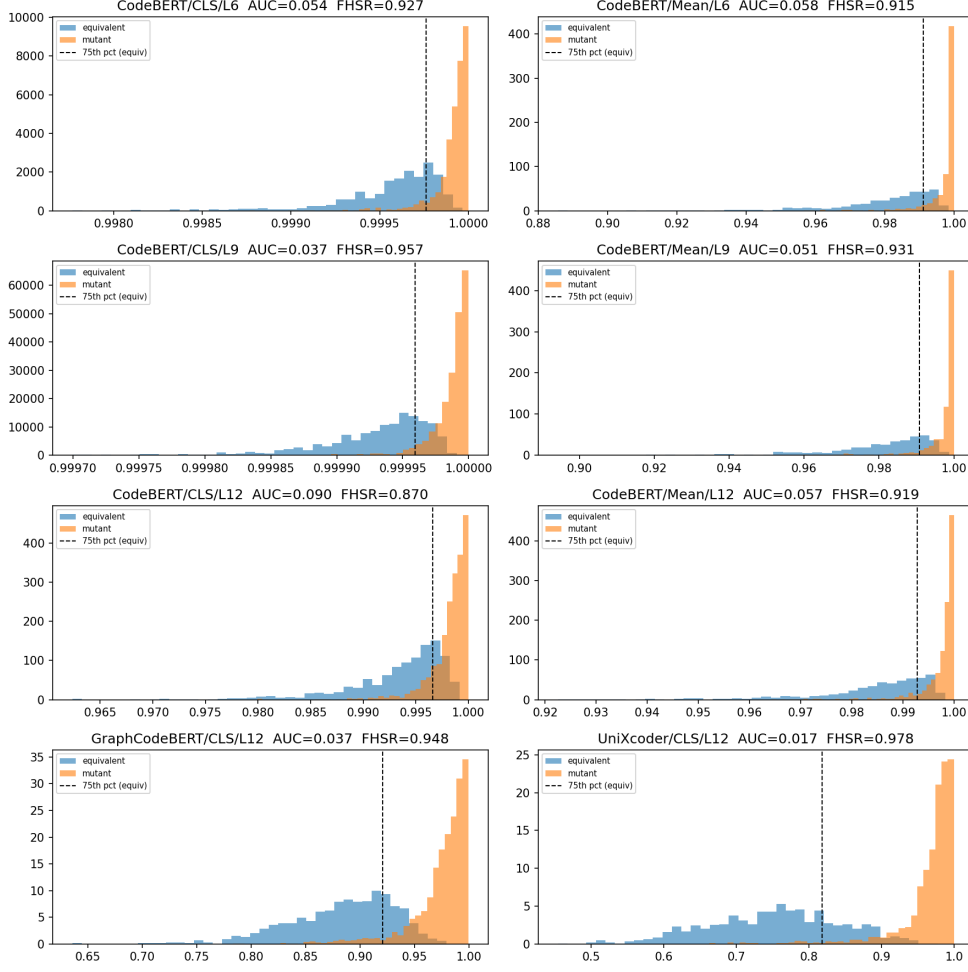


Fig. 2 Cosine-similarity distributions of equivalent (blue) and mutant (orange) pairs for each configuration; the dashed line marks the 75th percentile of the equivalent-pair scores (the FHSR threshold). In every panel the mutant distribution lies to the right of the equivalent distribution, and the per-panel axis ranges show the anisotropic compression of the CodeBERT CLS configurations.

differs slightly from the primary range (0.017–0.090, Table 5) because this is a different comparison (within-anchor positive-versus-negative), not because the two analyses disagree; both place every configuration far below chance. As set out above, this is a robustness check on the independence assumption, not a control for surface overlap; the surface-versus-behaviour question is taken up directly in RQ2 (Sections 3.4.2 and 3.4.3).

Pairwise comparison of the eight configurations confirms that they differ but all fail. Of the $\binom{8}{2} = 28$ paired-bootstrap AUC comparisons, 23 have Bonferroni-adjusted intervals excluding zero, and 27 of 28 paired Wilcoxon tests on the per-pair scores are significant at the corrected level $\alpha' = 0.05/28 \approx 0.0018$; the full comparison is given in

Table 6 Anchored robustness comparison over the 1,098 verified anchors. Positive = $\cos(\text{anchor}, \text{equivalent partner})$, negative = $\cos(\text{anchor}, \text{killed mutant})$; AUC below 0.5 means the mutant is scored above the equivalent partner.

Model	Config	AUC $\uparrow$	FHSR $\downarrow$	Mean pos.	Mean neg.
CodeBERT	CLS, L6	0.045	0.942	0.999	1.000
	CLS, L9	0.036	0.962	1.000	1.000
	CLS, L12	0.079	0.892	0.993	0.998
	Mean, L6	0.053	0.925	0.977	0.997
	Mean, L9	0.048	0.938	0.977	0.997
	Mean, L12	0.050	0.931	0.981	0.997
GraphCodeBERT	CLS, L12	0.032	0.956	0.881	0.974
UniXcoder	CLS, L12	0.017	0.978	0.752	0.965

Supplementary Section S1 (Table S1). CodeBERT CLS/Layer 12 differs significantly from every other configuration and from both baselines. The minimum detectable effect at this corrected level and power $1 - \beta = 0.80$, estimated from the paired-bootstrap standard error of the highest-AUC pair, is 0.017 AUC points, so differences smaller than this are reported as not significant without inferring equivalence.

4.1.1 Training-free isotropy corrections deepen the failure

Section 5 argues that restoring isotropy cannot, on its own, repair a polarity failure rooted in surface near-identity; we test this directly. Each configuration’s embeddings are post-processed with two standard, training-free isotropy transforms fit on a disjoint background of 4,000 eligible functions that appear in no analysis pair: whitening [31] and all-but-the-top [32]. The untransformed (*raw*) column of Table 7 reproduces the AUC values of Table 5 from the freshly extracted embeddings, confirming the extraction. Neither correction helps; both worsen discrimination in every configuration. Whitening drives AUC to 0.004–0.008 and FHSR to 0.994–0.998 across all eight; all-but-the-top moves them to 0.014–0.024 and 0.972–0.985. For the best configuration (CodeBERT CLS, layer 12), AUC falls from 0.090 to 0.007 under whitening and 0.024 under all-but-the-top. The mechanism is the one anticipated there: removing the shared anisotropic component that compresses scores toward 1 exposes, rather than offsets, the surface-driven proximity that already ranks each mutant above an independent equivalent. Isotropy is not the missing ingredient.

4.1.2 A contrastively trained encoder inverts too

To test whether the failure is specific to masked-language-model pre-training, we evaluate CodeSage [30], a frozen encoder whose second pre-training stage is contrastive, as an additional probe on the same analysis set and metrics. It does not escape the inversion: AUC = 0.016 with FHSR = 0.981, comparable to the weakest of the eight configurations. Its equivalent-pair similarity threshold (0.679) sits well below the near-1 thresholds of the anisotropic CodeBERT CLS configurations, so CodeSage occupies a far less compressed region of the representation space yet still ranks behaviourally divergent mutants above equivalents. The failure tracks the surface form of the inputs

Table 7 Training-free isotropy corrections applied to each configuration’s embeddings, fit on a disjoint 4,000-function background. **raw** reproduces Table 5; **wht** = whitening [31]; **abt** = all-but-the-top [32]. FHSR is against each transform’s own 75th-percentile equivalent threshold. Both corrections lower AUC and raise FHSR in every configuration.

Configuration	AUC $\uparrow$			FHSR $\downarrow$		
	raw	wht	abt	raw	wht	abt
CodeBERT/CLS/L6	0.054	0.007	0.021	0.927	0.997	0.979
CodeBERT/CLS/L9	0.037	0.006	0.015	0.957	0.995	0.985
CodeBERT/CLS/L12	0.090	0.007	0.024	0.870	0.995	0.976
CodeBERT/Mean/L6	0.058	0.006	0.019	0.915	0.995	0.980
CodeBERT/Mean/L9	0.051	0.005	0.021	0.931	0.994	0.972
CodeBERT/Mean/L12	0.057	0.004	0.018	0.919	0.998	0.979
GraphCodeBERT/CLS/L12	0.037	0.008	0.019	0.948	0.995	0.979
UniXcoder/CLS/L12	0.017	0.007	0.014	0.978	0.995	0.985

rather than the pre-training objective or the degree of anisotropy. CodeSage is a diagnostic probe, not a proposed correction to CodeBERT; Table 8 summarises its configuration and result.

Table 8 CodeSage frozen-probe configuration and RQ1 discrimination result, with the best CodeBERT configuration for reference. Metric definitions match Table 5; the threshold is the 75th percentile of the probe’s own equivalent-pair score distribution.

Property	Value
Checkpoint	<code>codesage/codesage-base</code>
Pre-training	two-stage (second stage contrastive)
Pooling	end-of-sequence (EOS) token embedding
Equivalent-pair threshold (75th pct)	0.679
AUC	0.016
FHSR	0.981
Reference: CodeBERT (CLS, layer 12)	AUC 0.090, FHSR 0.870

4.1.3 The inversion survives token-overlap matching.

The primary comparison contrasts single-site mutants (near-identical to their base) with independently written equivalents, so surface overlap is correlated with the label (Section 3.3.2). To separate the two, we recompute discrimination with unigram token overlap held fixed by coarsened exact matching: pairs are binned by unigram Jaccard and the larger class is subsampled to the smaller within each bin, equalising the overlap distribution across classes. On the matched set (best configuration; $n = 390$ over six common-support bands), AUC rises from 0.090 to 0.352 (95% bootstrap CI [0.300, 0.409], $B = 10,000$; the upper bound excludes 0.5, and no resample reached it)

but stays far below 0.5: with overlap fixed, the behaviourally divergent mutant still scores above the equivalent in roughly two comparisons of three. Lexical overlap is the larger driver of the gap to chance matching closes 0.262 of the 0.410 gap between the raw AUC (0.090) and chance (0.5), but it is not the whole story: the remaining 0.148 survives with overlap held fixed, leaving the matched AUC far below chance rather than merely reduced. That overlap-independent component is the operationally important one, since it is the part a practitioner cannot remove by deduplicating inputs, and it keeps the mutant above the equivalent in a clear majority of overlap-matched comparisons, consistent with the partial token-overlap effect quantified in RQ2.

4.2 RQ2: Category-Level Confusion Results

Figure 3 reports per-category FHSR under the best RQ1 configuration (CodeBERT CLS, layer 12). The three categories fail at almost exactly the same rate: COD 0.872 (95% CI [0.847, 0.895]), ROR 0.868 ([0.844, 0.892]), and AI 0.867 ([0.828, 0.907]). None of the three Holm-corrected pairwise two-proportion tests approaches significance ($p_{\text{Holm}} = 1.000$ in every case, with $|z| < 0.2$), and the confidence intervals overlap almost entirely. The discrimination failure is therefore pervasive rather than category-specific: no transformation family escapes it, and none is disproportionately affected. The wider interval for AI reflects its smaller sample (279 pairs against 779 and 775), as anticipated in Section 3.4.1, not a genuine difference in rate.

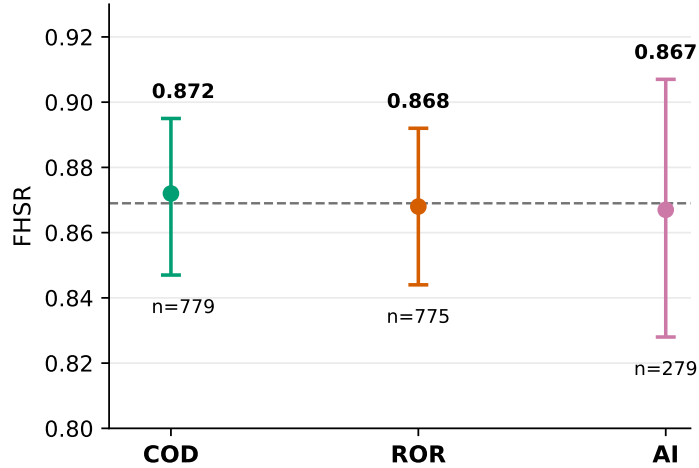


Fig. 3 Per-category FHSR under the best RQ1 configuration (CodeBERT CLS, layer 12), with 95% bootstrap confidence intervals ($B = 10,000$). Point estimates are COD 0.872 ($n = 779$), ROR 0.868 ($n = 775$), and AI 0.867 ($n = 279$); all three Holm-corrected pairwise two-proportion tests are non-significant ($p_{\text{Holm}} = 1.000$), so the categories fail at statistically indistinguishable rates.

Token overlap accounts for part of the confusion but not all of it. High-confusion mutant pairs are more lexically similar to their originals than low-confusion pairs

(mean unigram Jaccard 0.809 against 0.653; bigram 0.709 against 0.529), and a logistic regression of the high-confusion label on J_{unigram} and J_{bigram} (Supplementary Table S7) assigns a large positive coefficient to unigram overlap and achieves a held-out AUC of 0.768 with a McFadden pseudo- R^2 of 0.149. The association is clear but far from deterministic: an AUC of 0.77 and a modest pseudo- R^2 leave a substantial share of high-confusion pairs with only moderate overlap. The two predictors are strongly collinear (Pearson $r = 0.94$), so the unigram term absorbs the shared signal and the bigram coefficient (-0.477) is not separately interpretable; the predictors should be read jointly. Surface similarity is thus a significant contributor to the confusion rather than its sole cause, consistent with an embedding geometry that already places most mutant pairs near the equivalent-pair distribution irrespective of overlap.

Structural overlap is uniformly high and tracks confusion only weakly. Mean ASTOverlap is near-identical across categories (COD 0.939, ROR 0.940, AI 0.954), confirming that single-site mutants are structural near-twins of their originals. High-confusion pairs have slightly higher overlap than low-confusion pairs (median 0.950 against 0.928; Mann–Whitney $p = 5.3 \times 10^{-17}$), but the effect is small to medium (Cliff’s $\delta = 0.336$). The depth-bounded metric is defined over node *types*, so it does not register a mutation that changes only an identifier: 19.7% of AI pairs have an ASTOverlap of exactly 1.0 (the `min`/`max` and similar identifier-level inversions), against 0% for COD and ROR, whose operator changes always alter a node. This is why AI shows the highest mean structural overlap yet the same FHSR as the other categories, and it is precisely the case the token-overlap analysis captures, since the BPE tokeniser does distinguish `min` from `max`. The two views are complementary: the most-confused pairs are those that remain close to their originals on *both* the lexical and the structural axis, the more serious failure mode.

Taken together, the three analyses give a consistent picture. The failure is uniform across mutation categories; surface and structural similarity are both significantly elevated among the most-confused pairs yet account for only part of the confusion; and the residual is consistent with the anisotropic geometry documented in RQ1, in which nearly all mutant pairs are driven close to the equivalent-pair distribution regardless of their overlap with the original. Lexical and structural proximity modulate the confusion within that regime rather than generating it.

4.3 RQ3: Hybrid Model Results

The structural signal separates the two pair types almost perfectly, but in the wrong direction. Across the analysis set the normalised AST edit distance is near zero for mutant pairs (mean $d_{\text{AST}} = 0.010$) and an order of magnitude larger for equivalent pairs (0.360), so the structural similarity term $1 - d_{\text{AST}}$ averages 0.990 for mutants against 0.640 for equivalents. The asymmetry is uniform across categories (mutant $1 - d_{\text{AST}}$ of 0.993 for AI, 0.991 for ROR, 0.987 for COD), mirroring the token-overlap and AST-overlap asymmetries. Every parse succeeded, so all 2,617 pairs enter the analysis. This is the dissociation the directional expectation in Section 3.5.1 anticipated: structural similarity is highest exactly where behaviour diverges (Figure 4a).

Table 9 reports the four models on the held-out test set (2,094 pairs: 628 equivalent, 1,466 mutant). On the validation set every swept α gives HybridSim AUC at most

2×10^{-4} , statistically indistinguishable and far below both the cosine baseline (0.079) and the 0.5 chance level; we therefore fix $\alpha = 0.7$, the most cosine-weighted value, since the structural term carries no usable signal. It does not help. On the test set the Cosine baseline reproduces its RQ1 AUC (0.090; test-set FHSR 0.866), and adding any structural weight collapses discrimination: HybridSim falls to AUC 0.002 with FHSR 1.000 at all three α values, and ASTOnly is identical (AUC 0.002). An AUC of 0.002 means that in 99.8% of equivalent-versus-mutant comparisons the behaviourally divergent mutant receives the higher score, and an FHSR of 1.000 means every mutant pair now exceeds the 75th-percentile equivalent threshold. The HybridSim($\alpha = 0.7$) deficit relative to Cosine is -0.089 (95% bootstrap CI $[-0.101, -0.077]$, excluding zero; paired Wilcoxon $p < 10^{-300}$, matched-pairs rank-biserial -0.98). JaccardPenalty, which substitutes token for structural similarity, is less extreme but still significantly worse than Cosine (AUC 0.023, -0.067 , CI $[-0.079, -0.056]$).

That the structural term points the wrong way on this dataset is expected by construction: a single-site mutant differs from its base at one node, so its normalised AST distance is near zero. What the evaluation establishes is not the sign of the effect but its magnitude and reach. That α barely moves the result is itself informative. The structural term’s between-class separation (0.99 against 0.64) dwarfs the cosine term’s (0.998 against 0.993, and already inverted), so for any $\alpha < 1$ the ranking is governed by structure and the hybrid tracks ASTOnly. The robustness of the conclusion does not depend on this scale comparison: both ingredients are individually inverted, with Cosine at AUC 0.090 and ASTOnly at 0.002, and across the convex combinations and normalisations evaluated here, the two wrong-direction signals reinforce the same ordering rather than correcting it. The Cosine and ASTOnly rows of Table 9 are the two endpoints of the sweep and bound every intermediate model. No training-free combination of cosine and AST similarity repairs the failure measured in RQ1; each one deepens it (Figure 4b). Read together with the isotropy corrections, this closes the training-free design space from both directions a practitioner would try: correcting the embedding geometry and adding an input-side structural signal both fail, because each leaves untouched the surface near-identity of the mutant inputs that produces the inversion. On this controlled dataset RQ3 therefore establishes a dissociation, not a deployment verdict: the correction is built for the opposite signature, clone-screening false positives that pair surface overlap with divergent structure, and whether the structural term helps there is left to future work.

5 Discussion

The central RQ1 finding is not that cosine similarity over frozen CodeBERT embeddings is imprecise but that it is inverted. Across all eight configurations the AUC for separating equivalent from behaviourally divergent pairs lies far below the 0.5 chance level (0.017 to 0.090), so a randomly drawn mutant pair receives a higher similarity than a randomly drawn equivalent pair more than nine times in ten. The best CodeBERT configuration (CLS, layer 12) reaches only 0.090, and its false-high-similarity rate is 0.870: most behaviourally divergent pairs score above the 75th percentile of the equivalent-pair distribution. A pipeline that ranks or filters candidates by frozen

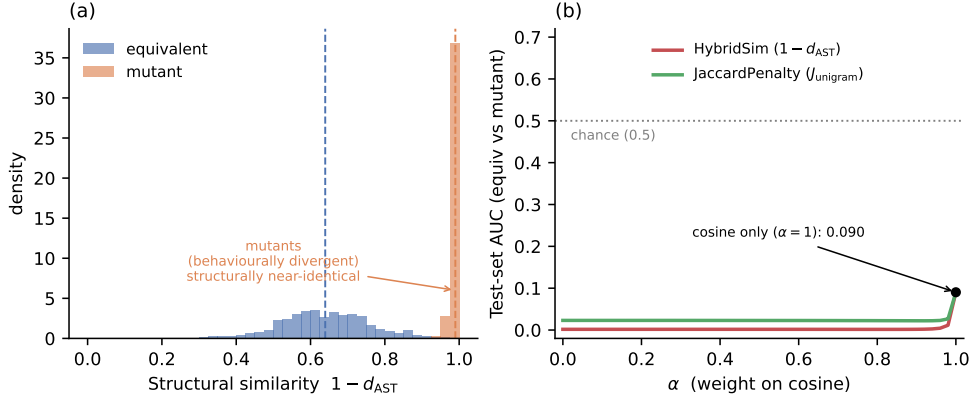


Fig. 4 Structure-behaviour dissociation. (a) Structural similarity $1 - d_{AST}$ by pair type: mutants are structurally near-identical to their originals while equivalents are diverse, so structure separates the classes opposite to behaviour (dashed lines: class means). (b) Held-out AUC of HybridSim and JaccardPenalty against the cosine weight α : discrimination stays near zero until $\alpha = 1$ and never approaches the 0.5 chance line, so no combination beats cosine alone.

Table 9 RQ3 four-model comparison on the held-out test set (2,094 pairs). AUC is equivalent-versus-mutant discrimination; FHSR is computed against each model’s own 75th-percentile equivalent threshold (quantile-matched). ΔAUC is relative to the Cosine baseline. HybridSim $\alpha = 0.7$ is the validation-selected value. Higher AUC and lower FHSR are better; every hybrid is worse than Cosine on both.

Model	α	AUC $\uparrow$	FHSR $\downarrow$	ΔAUC
Cosine	1.0	0.090	0.866	—
HybridSim	0.3	0.002	1.000	−0.089
HybridSim	0.5	0.002	1.000	−0.089
HybridSim	0.7	0.002	1.000	−0.089
ASTOnly	0.0	0.002	1.000	−0.089
JaccardPenalty	0.3	0.023	0.979	−0.067

cosine similarity will therefore treat a function and its single-operator inversion as more alike than two independently written correct solutions to the same problem. For code search, clone screening, and defect prediction that consume such scores, the score is not a weak signal of behavioural equivalence but a misleading one on exactly the pairs where polarity decides correctness.

To make the operational cost concrete, consider a clone-screening or semantic-search filter that admits pairs whose frozen cosine similarity exceeds the 75th percentile of a reference equivalent distribution, the standard “similar enough to surface” cutoff. At the best configuration this filter admits 87% of behaviourally divergent mutant pairs, and in a ranking setting a mutant outranks a genuine independent equivalent in about 91% of head-to-head comparisons (AUC 0.090), so a top- k retrieval over a

pool holding both tends to surface the mutant first. The score orders the two cases backwards, the more damaging failure for any system that consumes the ranking directly.

Two factors compound to produce the inversion, and they call for different remedies. The first is the anisotropy of BERT-family embedding spaces [5, 13]: cosine similarities concentrate near one regardless of content, visible here in equivalent-pair medians at or near 1.0 and in the high JSD values (0.47 to 0.80) that show the two score distributions are well separated yet ordered backwards. The second is the surface near-identity of the inputs: a single-site mutant shares almost every token and subtree with its original, whereas two independent equivalents do not. Isotropy-restoring methods such as contrastive fine-tuning [6, 15] address the first factor by spreading representations across the hypersphere, but they cannot on their own correct the second. Even in a perfectly isotropic space, a representation driven mainly by lexical and structural form, as probing studies report for these models [8, 9], will place a mutant nearer its original than an independently written equivalent. The failure is a property of what the representation encodes, not only of how the space is shaped. The training-free isotropy results bear this out and sharpen it: whitening [31] and all-but-the-top [32] not only fail to help but deepen the inversion in all eight configurations, because removing the anisotropic component exposes the surface-driven proximity rather than masking it. A frozen contrastively trained encoder (CodeSage [30]) inverts as well, in a far less compressed region of the space, so neither isotropy nor a generic contrastive objective is by itself sufficient here: the contrast pairs that would teach polarity, syntactically near-identical programs with divergent behaviour, are exactly the negatives such pre-training never sees.

RQ2 sharpens the reading for practitioners. The confusion does not concentrate in one operator family: the per-category false-high-similarity rates are statistically indistinguishable (all Holm-adjusted pairwise $p = 1.000$). Reliability cannot be recovered by restricting cosine similarity to a “safe” category, because none of the three is safer than the others. Token overlap accounts for part of the confusion but not all of it, and structural overlap is uniformly high and only weakly higher for the most-confused pairs. The two analyses agree: surface and structural proximity both raise mutant scores, but the dominant driver is the embedding geometry, which holds nearly all mutant pairs above the threshold irrespective of their precise overlap with the original.

RQ3 tests the natural remedy and shows why it fails. Because a single-site mutant is a structural near-twin of its original (normalised AST edit distance 0.010 against 0.360 for equivalent pairs), adding a structural similarity term moves mutant scores up rather than down: every hybrid we evaluated reduced discrimination relative to cosine alone, collapsing AUC from 0.090 to 0.002 and raising the false-high-similarity rate to 1.000. Structural similarity and the behavioural label are dissociated, with structure most similar exactly where behaviour diverges. Both ingredients are individually inverted, so across the reweightings, rescalings, and convex mixtures evaluated here the two signals reinforce the same wrong ordering; we do not claim a learned or non-convex combination cannot help (Section 6). The question a practitioner would reasonably ask, whether a cheap structural penalty can repair the embedding, is answered for this class of pairs, and the answer is no. Together with the isotropy corrections, which fail

for the same reason, this exhausts the training-free options from both the geometry and the input-structure side.

These results align with, and help explain, the equivalent-mutant-detection literature. Recent work reports that dedicated detectors, and fine-tuned code embeddings in particular, are needed to decide whether a syntactic mutation changes behaviour [25]. RQ1 supplies the reason the cheaper alternative is insufficient: frozen embedding geometry ranks the changed program above a genuine equivalent. The contribution here is to quantify that inversion on a controlled, I/O-verified benchmark and to show that it survives the obvious training-free corrections, which leaves fine-tuning or execution-based verification, rather than embedding or structural heuristics, as the route to behavioural discrimination in this setting.

The practical recommendation is correspondingly specific. When a pipeline must judge whether two Python functions are behaviourally equivalent and the candidate set contains near-duplicates that differ in polarity (inverted conditions, complemented relational operators, reversed algorithmic direction), cosine similarity over frozen CodeBERT-family embeddings should not serve as the discriminating signal, and neither should a training-free structural augmentation of it, nor a training-free isotropy correction such as whitening or all-but-the-top, both of which deepen the failure here: a high score in this setting reflects surface proximity, not behavioural agreement. Where labelled contrast pairs exist, contrastive fine-tuning is the established route to more functionality-aware representations [6, 7, 15], though the CodeSage result indicates the negatives must include behaviourally divergent, near-identical pairs rather than generic ones; where such labels do not exist, execution-based checks remain the dependable fallback. These statements are scoped to Python and to the frozen encoder models tested; whether the same inversion holds for CodeBERT’s other languages or for decoder-derived and instruction-tuned embedders is left open (Section 6).

6 Threats to Validity

6.1 Construct Validity

The equivalence labels rest on execution evidence, not proofs. Online-judge acceptance establishes agreement with the judge’s hidden test suite; the verification pipeline adds agreement on public sample inputs and on LLM-generated edge-case inputs. None of this establishes semantic equivalence over the full input domain. For the agreement-verified pairs (67 in the released dataset, 55 within the analysis set) the evidence is weaker still: the two functions agree with each other on the usable inputs, without an authoritative expected output, and could in principle share a fault pattern. We treat equivalent pairs as an operational proxy for behavioural equivalence and report all RQ1 headline metrics on the oracle-verified subset of analysis-set equivalent pairs alone as a sensitivity check (Section 3.2.3). The mutant-pair labels face the converse limit: divergence on every usable oracle input demonstrates consistent divergence on the tested inputs, not total divergence across the input domain, and the usable-input sets differ in size across problems. The `killed` label is therefore oracle-relative: a mutant can diverge from its original on every available oracle input yet still agree with it on inputs the oracle does not cover. Expanding the oracle would reclassify some such

pairs, so the reported mutant counts are specific to the oracle constructed here. The reliability study bears this out: all nine inter-annotator disagreements were **neither** judgements resting on agreement observed only on inputs introduced from outside the oracle set (Section 3.2.5), so they reflect oracle coverage rather than errors in the retained labels. Part of the extended oracle is generated by an LLM. Validation by executing an accepted submission confirms that a generated input is well formed and processable; it does not guarantee branch coverage, and a malformed input that the reference solution silently tolerates would pass undetected. The structural measures are also constructs rather than ground truth: the RQ2 AST overlap and the RQ3 normalised edit distance compare node *types*, so a mutation that changes only an identifier (the `min/max` inversions in AI) registers as structurally identical. This is faithful to the structural question being asked, but it means the structural signal is blind to identifier-level polarity changes, which is why the token-level analysis is reported alongside it.

6.2 Internal Validity

All pairs come from one corpus. This removes the domain-shift confound between equivalent and mutant pairs, at the price of inheriting any corpus-specific artefact in both subsets. Equivalent-pair selection retains the first verified pair in a deterministic enumeration order, which prefers earlier-indexed submissions; accepted pairs are therefore not a uniform sample of the eligible pool. Mutant selection retains the strongest classified candidate per problem and category, and the killed-only retention rule keeps only mutations whose effect reaches the output on every usable input; mutations off the main execution path are systematically excluded, so the mutant dataset represents consistently divergent mutants rather than mutants in general. The analysis-set restriction (Section 3.2.6) compounds this: the 16 problems excluded for lacking any killed mutant are precisely those whose oracle evidence separates behaviour least sharply, so analysis coverage is tied to oracle strength and problems with weaker oracles are under-represented. The anchored robustness check (Section 3.3.2) carries a related skew: its 20 excluded bases are longer than those retained, so the anchored set under-represents long functions, though those bases' mutant pairs remain in the primary analysis and the anchored and primary comparisons agree. Near-duplicate exclusion uses a BPE unigram cosine threshold of 0.99; submissions below that threshold can still overlap heavily, which inflates within-problem lexical similarity. The 75th-percentile FHSR threshold is one of several defensible choices, which is why FHSR at the 50th and 90th percentiles is reported as a sensitivity analysis.

6.3 External Validity

The dataset consists of competitive programming solutions in Python: short, algorithm-focused functions with few defensive guards, as the Contract Violation analysis in Section 3.1 quantifies (47 of 800 base functions contain an identifiable precondition guard). Production code differs in structure and in the prevalence of input validation, and the reported failure rates should not be extrapolated to it without further evidence. All polarity results are scoped to Python and to the frozen encoder

models tested (CodeBERT, GraphCodeBERT, UniXcoder, and the contrastively trained CodeSage); whether the inversion holds for decoder-derived or instruction-tuned code embedders is left open. The cross-model evidence supports “CodeBERT inverts, and three further frozen encoders also invert at their standard setting”, not a characterisation of the encoder-only family as a whole: each non-CodeBERT model is probed at a single configuration, so the family-level reading is a suggestive extrapolation rather than a measured claim. The hybrid-correction result (RQ3) is likewise scoped to the controlled Python dataset: it establishes that structural similarity dissociates from behavioural divergence on mutant pairs, and does not speak to the deployment-style clone-screening setting the correction targets, which is left to future work.

6.4 Conclusion Validity

Each family of comparisons carries its own multiple-comparison correction (Bonferroni for the RQ1 configuration comparisons, Holm for the RQ2 category contrasts), but the study spans several metric families, and borderline individual results should be read with that in mind. We correct within rather than across families because the families test logically distinct hypotheses (configuration discrimination, category equality, hybrid effect) and a study-wide correction would over-penalise comparisons that are not jointly interpreted; we note this choice so the reader can apply a stricter threshold if preferred, observing that the headline effects clear any reasonable correction by a wide margin (the AUC inversions sit far outside their bootstrap intervals). Cohen’s κ is sensitive to label prevalence, so the reliability study reports raw agreement and PABAK alongside it (Section 3.2.5); reporting all three guards against the prevalence concern in this skewed sample. Both annotators are external to the author list, so the reliability estimate reflects two independent judgements rather than author self-assessment; a larger annotated sample would tighten the estimate further. In the event, the three measures agree that reliability is high (pooled $\kappa = 0.925$, raw agreement 96.6%), so the prevalence concern does not undermine the result. Finally, the training-free conclusions are bounded to their claim: the corrections we tested, convex combinations of cosine and AST similarity (RQ3) and the whitening and all-but-the-top isotropy transforms (Section 4.1), all fail on this contrast class, a conclusion that rests on the component signals being individually inverted; it does not assert that learned, fine-tuned, or polarity-aware-negative methods are unable to help.

7 Conclusion

We constructed a controlled, I/O-verified benchmark of behaviourally divergent mutant pairs drawn entirely from Project_CodeNet_Python800 (800 equivalent and 1,833 killed mutant pairs across the COD, ROR, and AI operators; a 2,617-pair analysis set over 784 problems; inter-rater $\kappa = 0.925$) and used it to ask whether cosine similarity over frozen CodeBERT embeddings separates behaviourally divergent pairs from equivalent ones. It does not. Across eight embedding configurations spanning three frozen encoders the discrimination AUC stays far below chance (0.017 to 0.090) and the false-high-similarity rate remains at or above 0.87; the inversion

persists under a shared-anchor comparison and a token-overlap-matched comparison, a frozen contrastively trained encoder (CodeSage) does not escape it, and two standard training-free isotropy corrections (whitening and all-but-the-top) deepen rather than repair it. The confusion is uniform across the three operator families (RQ2) and is explained only in part by token and structural overlap; a training-free hybrid that adds a normalised AST signal does not repair it but deepens it (RQ3: AUC falls to 0.002), because single-site mutants are structurally near-identical to their originals.

The failure is not inherent to CodeBERT, which performs well on the tasks it was trained for. It arises from applying cosine similarity over frozen embeddings to a polarity-sensitive equivalence decision, where a high score reflects surface proximity rather than behavioural agreement. Practitioners who must separate a function from its behavioural inverse should not rely on frozen embedding geometry, nor on a training-free structural augmentation of it; contrastive fine-tuning or execution-based verification is the route that remains. These conclusions are scoped to Python and to the frozen encoder models tested.

Three directions follow from this study. The first is the deployment-oriented counterpart of RQ3. The structural correction examined here is designed for the clone-screening false positive, where surface tokens overlap but overall structure diverges, which is the opposite of the mutant-pair signature. Evaluating it on an inter-project clone benchmark such as BigCloneBench [17], where the AST term should help rather than hurt, and comparing it against fine-tuned baselines to establish whether a training-free correction is competitive, is the subject of ongoing work. The second is cross-language generalisation: whether the inversion documented here for Python holds for CodeBERT’s other supported languages (Java, JavaScript, PHP, Ruby, and Go) is an open empirical question. The third concerns the taxonomy: the Contract Violation category was excluded for want of precondition guards in competitive-programming code (47 of 800 base functions contain an identifiable guard), and releasing those 47 candidates as a descriptive stratum, together with extending the benchmark to production code with richer defensive structure, would broaden the range of behavioural divergence the benchmark covers.

Declarations

Acknowledgements

The authors thank the two independent annotators, both external to the author list, for performing the inter-rater reliability labelling.

Funding

The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

Competing interests

The authors have no relevant financial or non-financial interests to disclose.

Ethics approval and consent to participate

Not applicable. The study involves no human or animal subjects; the inter-rater reliability annotation was a technical code-labelling task performed by two external expert annotators and did not constitute research involving human participants.

Consent for publication

Not applicable.

Data availability

The dataset (2,633 pairs: 800 equivalent and 1,833 I/O-verified behaviourally divergent mutant pairs, with an `in_analysis_set` flag marking the 2,617-pair analysis set), together with the full classified mutant file retained as an intermediate artefact, verification statuses, oracle inputs, and IRR annotation materials, are available at submission from <https://github.com/letsholok/codebert-mutant-similarity>.

Code availability

All construction, verification, and analysis code is available at submission in the same repository.

Supplementary material

This article is accompanied by a supplementary document (fifteen tables and worked code examples), referenced throughout as the supplementary material. It reports the full pairwise comparison of the eight embedding configurations (Table S1); FHSR at the 50th, 75th, and 90th percentiles (Table S2); the oracle-verified-subset re-run of RQ1 (Table S3); the token-overlap-matched (coarsened exact matching) control (Table S4); the RQ3 validation-set α sweep (Table S5); per-category token- and AST-overlap distributions (Table S6) and the token-overlap logistic regression (Table S7); the inter-rater reliability confusion matrix and per-rater accuracy (Tables S8 and S9); the behavioural classification of all 2,282 mutant candidates (Table S10); the anchor-selection bias analysis and excluded-item lists (Table S11); the oracle-generation procedure; implementation and reproducibility details, including software versions, hardware, and the artefact repository; the released-data dictionary (Tables S12–S14); and worked example pairs (Table S15).

Author contribution

All authors contributed to the study conception and design. Keletso J. Letsholo and Nourchène Elleuch Ben Ayed contributed equally, leading the methodology, implementation, dataset construction, analysis, and manuscript drafting. Muath Alrammal and Achraf Ghorbel contributed to the experimental design, interpretation of results, and critical revision of the manuscript. All authors read and approved the final manuscript.

References

- [1] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., Zhou, M.: CodeBERT: A pre-trained model for programming and natural languages. In: Findings of the Association for Computational Linguistics: EMNLP 2020, pp. 1536–1547. Association for Computational Linguistics, Online (2020). <https://doi.org/10.18653/v1/2020.findings-emnlp.139>
- [2] Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., Zhou, L., Duan, N., Svyatkovskiy, A., Fu, S., Tufano, M., Deng, S.K., Clement, C.B., Drain, D., Sundaresan, N., Yin, J., Jiang, D., Zhou, M.: GraphCodeBERT: Pre-training code representations with data flow. In: 9th International Conference on Learning Representations (ICLR 2021), Online (2021). <https://doi.org/10.48550/arXiv.2009.08366>
- [3] Guo, D., Lu, S., Duan, N., Wang, Y., Zhou, M., Yin, J.: UniXcoder: Unified cross-modal pre-training for code representation. In: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 7212–7225. Association for Computational Linguistics, Dublin, Ireland (2022). <https://doi.org/10.18653/v1/2022.acl-long.499>
- [4] Jia, Y., Harman, M.: An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering* **37**(5), 649–678 (2011) <https://doi.org/10.1109/TSE.2010.62>
- [5] Ethayarajh, K.: How contextual are contextualized word representations? Comparing the geometry of BERT, ELMo, and GPT-2 embeddings. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pp. 55–65. Association for Computational Linguistics, Hong Kong, China (2019). <https://doi.org/10.18653/v1/D19-1006>
- [6] Gao, T., Yao, X., Chen, D.: SimCSE: Simple contrastive learning of sentence embeddings. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pp. 6894–6910. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic (2021). <https://doi.org/10.18653/v1/2021.emnlp-main.552>
- [7] Jain, P., Jain, A., Zhang, T., Abbeel, P., Gonzalez, J.E., Stoica, I.: Contrastive code representation learning. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pp. 5954–5971. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic (2021). <https://doi.org/10.18653/v1/2021.emnlp-main.482>
- [8] Naik, S., Patil, R., Agarwal, S., Baths, V.: Probing semantic grounding in language models of code with representational similarity analysis. In: Chen, W., Yao,

- L., Cai, T., Pan, S., Shen, T., Li, X. (eds.) Advanced Data Mining and Applications (ADMA 2022), Part II. Lecture Notes in Computer Science, vol. 13726, pp. 395–406. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-22137-8_29
- [9] Zhang, L., Cao, C., Wang, Z., Liu, P.: Which Features are Learned by CodeBert: An Empirical Study of the BERT-based Source Code Representation Learning (2023). <https://arxiv.org/abs/2301.08427>
 - [10] Puri, R., Kung, D.S., Janssen, G., Zhang, W., Domeniconi, G., Zolotov, V., Dolby, J., Chen, J., Choudhury, M., Decker, L., Thost, V., Buratti, L., Pujar, S., Ramji, S., Finkler, U., Malaika, S., Reiss, F.: CodeNet: A large-scale AI for code dataset for learning a diversity of coding tasks. In: Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks, vol. 1 (2021). <https://doi.org/10.48550/arXiv.2105.12655>
 - [11] Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT), pp. 4171–4186. Association for Computational Linguistics, Minneapolis, Minnesota (2019). <https://doi.org/10.18653/v1/N19-1423>
 - [12] Kanade, A., Maniatis, P., Balakrishnan, G., Shi, K.: Learning and evaluating contextual embedding of source code. In: Proceedings of the 37th International Conference on Machine Learning (ICML). Proceedings of Machine Learning Research, vol. 119, pp. 5110–5121. PMLR, Online (2020). <https://doi.org/10.48550/arXiv.2001.00059>
 - [13] Li, B., Zhou, H., He, J., Wang, M., Yang, Y., Li, L.: On the sentence embeddings from pre-trained language models. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 9119–9130. Association for Computational Linguistics, Online (2020). <https://doi.org/10.18653/v1/2020.emnlp-main.733>
 - [14] Diera, A., Galke, L., Scherp, A.: Isotropy matters: Soft-ZCA whitening of embeddings for semantic code search. In: Proceedings of the 33rd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2025), Bruges, Belgium, pp. 627–632 (2025). <https://doi.org/10.14428/esann/2025.ES2025-58>
 - [15] Liu, S., Li, Y., Xie, X., Meng, G., Liu, Y.: ContraBERT: Enhancing code pre-trained models via contrastive learning. In: Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE), pp. 2476–2487. IEEE, Melbourne, Australia (2023). <https://doi.org/10.1109/ICSE48619.2023.00207>
 - [16] Roy, C.K., Cordy, J.R., Koschke, R.: Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. Science of Computer

Programming **74**(7), 470–495 (2009) <https://doi.org/10.1016/j.scico.2009.02.007>

- [17] Svajlenko, J., Islam, J.F., Keivanloo, I., Roy, C.K., Mia, M.M.: Towards a big data curated benchmark of inter-project code clones. In: Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 476–480. IEEE Computer Society, Victoria, Canada (2014). <https://doi.org/10.1109/ICSME.2014.77>
- [18] Sonnekalb, T., Gruner, B., Brust, C.-A., Mäder, P.: Generalizability of code clone detection on codebert. In: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. ASE '22. Association for Computing Machinery, New York, NY, USA (2023). <https://doi.org/10.1145/3551349.3561165>
- [19] Ahmed, G.A., Patten, J.V., Han, Y., Lu, G., Hou, W., Gregg, D., Buckley, J., Chochlov, M.: Nearest-neighbor, BERT-based, scalable clone detection: A practical approach for large-scale industrial code bases. *Software: Practice and Experience* **54**(12), 2349–2374 (2024) <https://doi.org/10.1002/spe.3355>
- [20] Abid, S., Cai, X., Jiang, L.: Interpreting CodeBERT for semantic code clone detection. In: 2023 30th Asia-Pacific Software Engineering Conference (APSEC), pp. 229–238. IEEE, Seoul, South Korea (2023). <https://doi.org/10.1109/APSEC60848.2023.00033>
- [21] Song, Y., Lothritz, C., Tang, D., Bissyandé, T.F., Klein, J.: Revisiting code similarity evaluation with abstract syntax tree edit distance. In: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pp. 38–46. Association for Computational Linguistics, Bangkok, Thailand (2024). <https://doi.org/10.18653/v1/2024.acl-short.3>
- [22] Henkel, J., Ramakrishnan, G., Wang, Z., Albarghouthi, A., Jha, S., Reps, T.: Semantic robustness of models of source code. In: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 526–537. IEEE, Honolulu, USA (2022). <https://doi.org/10.1109/SANER53432.2022.00070>
- [23] Rabin, M.R.I., Alipour, M.A.: On the generalizability of neural program models with respect to semantic-preserving program transformations. *Information and Software Technology* **135**, 106552 (2021) <https://doi.org/10.1016/j.infsof.2021.106552>
- [24] Papadakis, M., Kintis, M., Zhang, J., Jia, Y., Traon, Y.L., Harman, M.: Mutation testing advances: An analysis and survey. In: *Advances in Computers* vol. 112, pp. 275–378. Elsevier, ??? (2019). <https://doi.org/10.1016/bs.adcom.2018.03.015>
- [25] Tian, Z., Shu, H., Wang, D., Cao, X., Kamei, Y., Chen, J.: Large language models for equivalent mutant detection: How far are we? In: Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)

- 2024), pp. 1733–1745. Association for Computing Machinery, New York, NY, USA (2024). <https://doi.org/10.1145/3650212.3680395>
- [26] Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing* **18**(6), 1245–1262 (1989) <https://doi.org/10.1137/0218082>
 - [27] Pawlik, M., Augsten, N.: Tree edit distance: Robust and memory-efficient. *Information Systems* **56**, 157–173 (2016) <https://doi.org/10.1016/j.is.2015.08.004>
 - [28] Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., Clement, C.B., Drain, D., Jiang, D., Tang, D., Li, G., Zhou, L., Shou, L., Zhou, L., Tufano, M., Gong, M., Zhou, M., Duan, N., Sundaresan, N., Deng, S.K., Fu, S., Liu, S.: CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS Datasets and Benchmarks 2021)*, vol. 1 (2021). <https://doi.org/10.48550/arXiv.2102.04664>
 - [29] Ding, Y., Steenhoeck, B., Pei, K., Kaiser, G., Le, W., Ray, B.: TRACED: Execution-aware pre-training for source code. In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE 2024)*, pp. 36–13612. Association for Computing Machinery, Lisbon, Portugal (2024). <https://doi.org/10.1145/3597503.3608140>
 - [30] Zhang, D., Ahmad, W.U., Tan, M., Ding, H., Nallapati, R., Roth, D., Ma, X., Xiang, B.: CODE representation learning at scale. In: *The Twelfth International Conference on Learning Representations*, Vienna, Austria (2024). <https://doi.org/10.48550/arXiv.2402.01935>
 - [31] Su, J., Cao, J., Liu, W., Ou, Y.: Whitening sentence representations for better semantics and faster retrieval. *arXiv preprint arXiv:2103.15316* (2021) <https://doi.org/10.48550/arXiv.2103.15316>
 - [32] Mu, J., Viswanath, P.: All-but-the-top: Simple and effective postprocessing for word representations. In: *6th International Conference on Learning Representations (ICLR 2018)*, Vancouver, Canada (2018). <https://doi.org/10.48550/arXiv.1702.01417>

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- [AUSESsupplementaryDocument.pdf](#)