Cross Lingual Code Embeddings: An Empirical Study of Dialects, Complexity, and Correctness

Anonymous ACL submission

Abstract

We frame code language models as multilingual learners and examine what their embedding spaces encode about language identity, framework choice, algorithmic complexity, and code correctness. Using seven embedding models spanning code-specialised, general-purpose, and commercial architectures, we analyse over 6,600 code snippets across 19 programming languages. We find that (1) language family structure is model-dependent, with only four pairings stable across all seven models; (2) frameworks create statistically significant within-language dialects (Cohen's d up to 1.88); (3) complexity class produces a weak but consistent signal ($d = 0.30$–0.64) across six of seven models; and (4) buggy–fixed code pairs are largely indistinguishable, with dangerous-neighbourhood rates ranging from 30% to 100% at $\tau = 0.10$. Two code-specialised models exhibit opposite behaviours: UniXCoder prioritises structural semantics over surface patterns, while CodeBERT's geometric compression suppresses all discriminative signals. These results indicate that code specialisation alone does not determine embedding geometry, with direct implications for code retrieval and RAG pipelines.

1 Introduction

Yun et al. (2026) showed that programming languages cluster into coherent families under code embedding models, and that fine-tuning on one family transfers to related languages with effect sizes between 0.42 and 0.71. The practical implication is appealing: define language families once, then use those families to guide multilingual adaptation.

A prior question, however, goes unexamined: are those families stable properties of programming languages, or artefacts of the specific embedding model used to discover them? If family boundaries shift with the choice of embedder, any downstream adaptation study built on a single family partition inherits that choice as an unacknowledged assumption. This matters for the same reason it matters in multilingual NLP more broadly: transfer claims depend critically on how language similarity is operationalised (Pires et al., 2019; Conneau et al., 2020).

We extend the analysis across four research questions. **RQ1** asks whether programming language families are stable across embedding models, or whether family assignments shift with the choice of embedder. **RQ2** asks whether popular frameworks create detectable sub-clusters within a language (e.g., does Django Python embed distinctly from Flask Python?). **RQ3** asks whether algorithmic complexity class ($O(n)$ vs. $O(n^2)$ vs. $O(n \log n)$, etc.) manifests as a recoverable geometric signal. **RQ4** asks whether correct and buggy implementations of the same algorithm are geometrically separable.

A hierarchy emerges across the four questions: language identity is the dominant geometric axis, followed by framework choice, then complexity, then correctness. UniXCoder, trained explicitly on code via masked prediction and code-NL alignment, is a systematic outlier in each case, less sensitive to surface patterns and more sensitive to structural ones. A second outlier pattern, geometric compression, appears in CodeBERT and suppresses discriminative signals across all four RQs.

Contributions

1. We show that programming language family assignments are representation-dependent: only four pairings (C–C++, Dart–Java, Kotlin–Scala, Python–Ruby) are invariant across all seven models tested.

2. We show that framework choice creates statistically significant dialects (d up to 1.88), with effect sizes spanning from large ($d > 0.8$) in five models to small ($d = 0.229$) in the most geometrically compressed model. This range suggests that geometry diagnostics should precede deployment decisions.

3. We show that complexity class is weakly but consistently encoded across general-purpose models ($d = 0.43$–0.64), while the two code-specialised models exhibit markedly different behaviours ($d = 0.300$ vs. 0.047).

4. We show that static code embeddings lack a reliable correctness axis. Dangerous-neighbourhood rates at $\tau = 0.10$ range from 30% to 100% across models, and large-scale statistical significance ($p < 0.01$) masks negligible effect sizes ($d < 0.15$) throughout.

All code, data generation scripts, and results are publicly available.[1]

2 Related Work

Programming language families Yun et al. (2026) introduced a feature-based framework for constructing programming language families from code embeddings, using 21 linguistic features across 19 languages. They showed that supervised fine-tuning on Java and Python yields statistically significant improvements on related low-resource languages (Kotlin, Haskell, Swift, AppleScript), with Cohen's d ranging from 0.42 to 0.71. Three models (ada-002, BGE, UniXCoder) produced broadly similar clustering structures in that study. We extend this by adding four further architectures including CodeBERT, explicitly measuring cross-model agreement, and quantifying per-language instability.

Code representation learning CodeBERT (Feng et al., 2020) trains on bimodal data pairing code with natural-language docstrings via masked language modelling and replaced token detection, using mean pooling over all token positions. UniXCoder (Guo et al., 2022) trains on code-natural language pairs with masked prediction and cross-modal alignment objectives. Both share a code-specialised training domain yet differ substantially in objective design and pooling strategy, and this architectural divergence is a central finding of this work. Our study uses both as code-specialised embedders alongside BGE-M3 (Chen et al., 2024) and Sentence-BERT variants (Reimers and Gurevych, 2019).

Complexity encoding Folea et al. (2023) showed that custom complexity-derived embeddings can classify algorithms by technique. Whether pre-trained neural code embeddings similarly encode runtime complexity was, to our knowledge, an open question; RQ3 addresses it directly.

Code benchmarks CodeXGLUE (Lu et al., 2021) and MultiPL-E (Cassano et al., 2023) provide standardised multi-language evaluation suites. Our evaluation draws on their polyglot structure while using embedding-space geometry rather than generative accuracy as the primary observable.

3 Methodology

3.1 Embedding Models

We evaluate seven models (Table 1). Following Yun et al. (2026), we retain their three evaluated models (ada-002, BGE-M3, and UniXCoder); the remaining four (all-MiniLM-L6-v2, Octen-Embed.-0.6B, Qwen3-Embed.-0.6B, and CodeBERT) were selected from the MTEB leaderboard (Muennighoff et al., 2022; Enevoldsen et al., 2025) to span code-specialised, multilingual, and sentence-similarity objectives as well as dimensionalities from 384 to 1,536. Crucially, the two code-specialised models (CodeBERT and UniXCoder) use distinct pre-training objectives, allowing us to test whether code specialisation alone determines what is encoded.

3.2 Analysis Pipeline and Metrics

Each RQ follows the same pipeline: embed all snippets through all seven models, compute geometric metrics, visualise per-model projections, then aggregate across models to assess robustness. We report silhouette scores (Rousseeuw, 1987), mean intra- and cross-group cosine distances, Welch's t-tests, and Cohen's

[1] https://anonymous.4open.science/r/inlp/

Key	Model	Dim	Type
ada002	text-embed.-ada-002	1536	API (OpenRouter)
bge_m3	BAAI/bge-m3	1024	Multilingual
codebert	codebert-base	768	Code-specialised
minilm	all-MiniLM-L6-v2	384	Sentence transf.
octen	Octen-Embed.-0.6B	1024	Open-source
qwen3	Qwen3-Embed.-0.6B	1024	Open-source
unixcoder	unixcoder-base	768	Code-specialised

Table 1: The seven embedding models evaluated. The two code-specialised models (CodeBERT, UniXCoder) share a training domain but use distinct pre-training objectives, yielding markedly different embedding geometries.

d (negligible < 0.2, small 0.2–0.5, medium 0.5–0.8, large ≥ 0.8), with 95% bootstrap confidence intervals (5,000 resamples). For RQ1, we use Ward-linkage hierarchical clustering. Dendrogram fidelity is quantified via the Cophenetic Correlation Coefficient (CCC) (Sokal and Rohlf, 1962), while cross-model agreement is measured using Spearman correlation of flattened distance matrices. For RQ4, a *dangerous neighbourhood* is a (bug-type, language) pair whose buggy–fixed cosine distance falls below threshold $\tau \in \{0.05, 0.10, 0.15\}$.

4 Datasets

4.1 RQ1: Programming Language Features

We use the original dataset provided by Yun et al. (2026), which consists of 19 languages spanning imperative, functional, object-oriented, and scripting paradigms. The dataset covers 21 linguistic feature categories (variable definitions, control-flow constructs, etc.).

4.2 RQ2: Framework-Specific Code

We constructed a synthetic dataset of 1,024 snippets at the cross-product of 8 languages $\times$ 4 frameworks (3 popular + vanilla baseline) $\times$ 8 software patterns $\times$ 4 stylistic variations. Languages were selected to cover different levels of framework opinionatedness. These range from thin-wrapper frameworks such as Go (Gin, Echo, Fiber) to more opinionated ones in Python (Django, Flask, FastAPI) and Ruby (Rails, Sinatra, Hanami). The set also includes Java, JavaScript, Kotlin, PHP, and Rust. The eight patterns cover common web-development contexts where frameworks impose distinctive idioms: authentication, REST endpoints, database operations, data validation, caching, background jobs, middleware, and file operations. All code was generated by Gemini 2.5 Pro (temperature 0.2).

4.3 RQ3: Algorithmic Solutions

We used solutions for 150 LeetCode problems from the NeetCode 150 collection, covering 9 languages (Python, C++, Java, JavaScript, C#, Go, Kotlin, Rust, Swift) for a total of 4,662 solutions (516–519 per language, balanced across Easy/Medium/Hard difficulty). They comprise 9 complexity classes, dominated by $O(n)$ (36.3%) and $O(n^2)$ (22.0%).

4.4 RQ4: Buggy–Fixed Code Pairs

The dataset contains 1,000 snippets: 100 bug types $\times$ 5 languages (C, Go, Java, Python, Swift) $\times$ 2 code types (buggy and fixed). Bug categories span syntax errors, logic and control-flow errors, state and algorithmic bugs, and numeric/memory/concurrency bugs, covering 100 distinct bug types. Pairs were generated with a mixture of Gemini 3, Claude Sonnet, and GPT 5 from correct implementations.

5 Results

5.1 RQ1: Model-Dependent Language Family Structure

Clustering fidelity The Cophenetic Correlation Coefficient varies substantially across models (Table 2). BGE-M3 achieves the highest CCC (0.799), indicating the most faithful dendrogram, while CodeBERT achieves the lowest (0.584). UniXCoder (0.661) ranks third.

Rank	Model	CCC
1	bge_m3	0.7990
2	qwen3	0.7040
3	unixcoder	0.6614
4	ada002	0.6328
5	minilm	0.6289
6	octen	0.6162
7	codebert	0.5844

Table 2: Cophenetic Correlation Coefficients for hierarchical clustering of 19 programming languages per model.

Stable and unstable pairings The seven models do not converge on a single family partition. Only four pairings are invariant across all seven models: C–C++, Dart–Java, Kotlin–Scala, and Python–Ruby. Raku joins

the Python–Ruby cluster in six of seven models; JavaScript and PHP co-cluster in six; Pascal and Visual Basic co-cluster in five. Beyond these local cores, assignments become unstable. Only six languages achieve a mean cross-model family overlap above 0.50 (Visual Basic 0.544; Python and Ruby 0.540; C and C++ 0.520; Raku 0.502). Rust is the least stable language (0.306), followed by Swift (0.329), JavaScript (0.347), AppleScript (0.350), PHP (0.353), and Haskell (0.367).

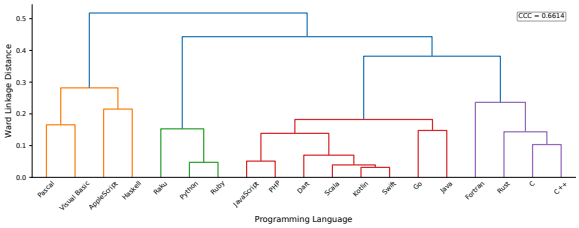

Figure 1: Hierarchical clustering dendrogram under UniXCoder (CCC = 0.661). The partition differs substantially from those produced by general-purpose models, with Rust joining the C/C++/Fortran systems cluster rather than the typed-language family.

At the other extreme, the lowest-CCC model (0.584) produces a distinct family partition: AppleScript and Haskell form a compact pair, most scripting and object-oriented languages merge into one large family, and Java and Go are isolated singletons. The low CCC traces to highly compressed cosine distances (maximum pairwise distance ≈ 0.06 versus > 0.50 for UniXCoder), which makes the dendrogram a poor representation of the underlying geometry (Figure 2).

Cross-model agreement Spearman correlations of flattened distance matrices (Table 3) reveal that Octen and Qwen3 are near-identical ($\rho = 0.984$), reflecting their shared base architecture. UniXCoder diverges from both ($\rho = 0.280$ and 0.252 respectively), the lowest values among all non-compressed models. CodeBERT shows moderate correlations with most models ($\rho = 0.19$–0.53) despite its compressed geometry; the rank order of pairwise distances is partially preserved even when absolute magnitudes collapse. MiniLM shows the weakest overall agreement ($\rho = 0.19$–0.36), consistent with training that targets natural-language sentence similarity rather than code structure.

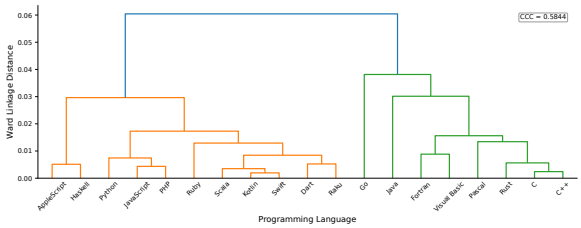

Figure 2: Hierarchical clustering dendrogram for the lowest-CCC model (CCC = 0.584). The compressed distance scale (y-axis maximum ≈ 0.06 versus > 0.50 for UniXCoder) indicates that this model conflates most languages into undifferentiated proximity.

	Oct.	BGE	UniX.	CB	Qw3	Mini.	Ada
Oct.	1.00	.652	.280	.429	**.984**	.364	.649
BGE		1.00	.547	.467	.597	.331	.619
UniX.			1.00	.442	.252	.337	.504
CB				1.00	.436	.191	.529
Qw3					1.00	.362	.631
Mini.						1.00	.237
Ada							1.00

Table 3: Spearman rank correlation between model distance matrices. The Octen–Qwen3 near-identity ($\rho = 0.984$) reflects shared architecture; MiniLM and UniXCoder show the weakest cross-model agreement overall.

Sources of instability Instability typically arises when a language occupies multiple taxonomic roles, forcing the model to prioritise one similarity signal over others. Rust, for instance, clusters with C/C++/Fortran (systems paradigm) under UniXCoder, with Go under Qwen3, and with Kotlin/Scala/Swift (typed-language family) under BGE-M3 and Ada-002. AppleScript, Haskell, and Fortran show similar cross-model variation. Three factors drive these shifts: paradigmatic ambiguity, uneven corpus scale for lower-resource languages, and surface token overlap that dominates when a model lacks explicit code-structure pre-training.

Implication for family-based adaptation Family-based adaptation studies presuppose that language families are stable enough to serve as an experimental unit. Our results suggest this assumption is fragile: whether Rust belongs in a systems family or a typed-language family is a model-specific claim, not a neutral

preprocessing decision, and any interference matrix constructed under one family definition conflates transfer effects with the upstream representational choice.

> **RQ1.** Language family assignments are model-dependent: only 4 pairings are stable across all 7 models. BGE-M3 produces the most faithful clustering (CCC = 0.799); CodeBERT the least (CCC = 0.584). Octen and Qwen3 are near-identical ($\rho = 0.984$); UniXCoder and MiniLM are the most structurally divergent. Neither scale nor code specialisation alone determines language similarity.

5.2 RQ2: Framework-Driven Dialects

Overall results Five of seven models detect a large effect ($d > 0.8$) for framework dialect separation; one shows a medium effect (UniXCoder, $d = 0.679$); and CodeBERT achieves only a small effect ($d = 0.229$), the lowest of all models. BGE-M3 achieves the strongest separation ($d = 1.878$). Table 4 gives the full results.

Model	FW Sil.	Cross	Intra	d	Effect
bge_m3	0.108	0.304	0.214	1.878	Large
minilm	0.126	0.672	0.431	1.587	Large
octen	0.021	0.370	0.257	1.486	Large
qwen3	0.022	0.383	0.264	1.448	Large
ada002	0.060	0.189	0.129	1.424	Large
unixcoder	−0.006	0.316	0.245	0.679	Medium
codebert	−0.114	0.022	0.019	0.229	Small

Table 4: RQ2 cross-model summary. FW Sil. = framework silhouette; Cross/Intra = mean cross-/intra-framework cosine distances. CodeBERT's compressed distances (Cross = 0.022) explain its anomalously low effect size.

Per-language variation Averaging framework silhouette across all seven models, Rust (+0.138), JavaScript (+0.130), and Python (+0.128) show the strongest dialect structure, while Go (+0.087) is consistently the weakest. Individual model rankings vary substantially. BGE-M3 favours Ruby (0.216), Kotlin (0.192), and Rust (0.191); MiniLM favours Rust (0.253) and Kotlin (0.243); and Octen and Qwen3 favour JavaScript. The top three languages by cross-model average, however, are stable. The t-SNE projection in Figure 4 makes this substructure visible; the per-language silhouette scores in Figure 3 quantify it under BGE-M3, where all eight languages exceed zero.

Go is the weakest case across all models: although its framework silhouette remains positive under six of seven models, the values are consistently the lowest. UniXCoder is the only model that assigns Go a negative framework silhouette (−0.016), consistent with Go's design philosophy of minimal abstraction where frameworks are thin wrappers rather than opinionated ecosystems.

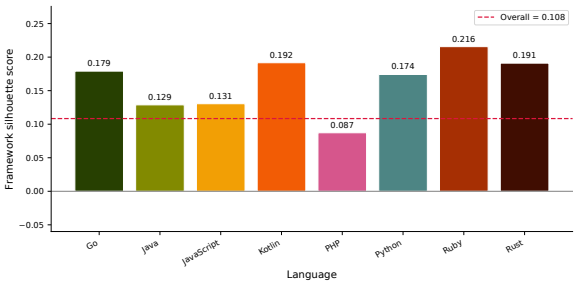

Figure 3: Per-language framework silhouette scores under BGE-M3. Ruby (0.216), Kotlin (0.192), and Rust (0.191) show the strongest framework-driven dialect structure; PHP (0.087) is the weakest. All languages exceed zero, confirming dialect signal across all eight languages.

Pattern-level analysis Under the best-performing model (BGE-M3), middleware/interceptors shows the largest cross-framework distance gap (0.100), where implementation is necessarily framework-specific. Authentication shows the smallest cross-framework distance (0.289), reflecting standardised patterns across frameworks.

Code-specialised outliers Both code-specialised models underperform general-purpose embedders, but for distinct reasons. UniXCoder's framework silhouette is marginally negative (−0.006) and its $d = 0.679$ is weaker than all five general-purpose and sentence-similarity models: code-specialised training partially normalises surface signals (import paths, decorator syntax, API names) in favour of semantic intent, yielding a medium rather than large effect. CodeBERT is the more extreme outlier: its framework silhouette (−0.114) is nearly twenty times more negative than UniXCoder's, and its distance gap (0.003) is ≈21× smaller, yielding only a small effect ($d = 0.229$). This traces to geometric compression rather than selective suppression. CodeBERT's maximum pairwise cosine distance is ≈0.06 versus >0.30 for UniXCoder, collapsing all 1,024 snippets into a narrow region where neither framework nor

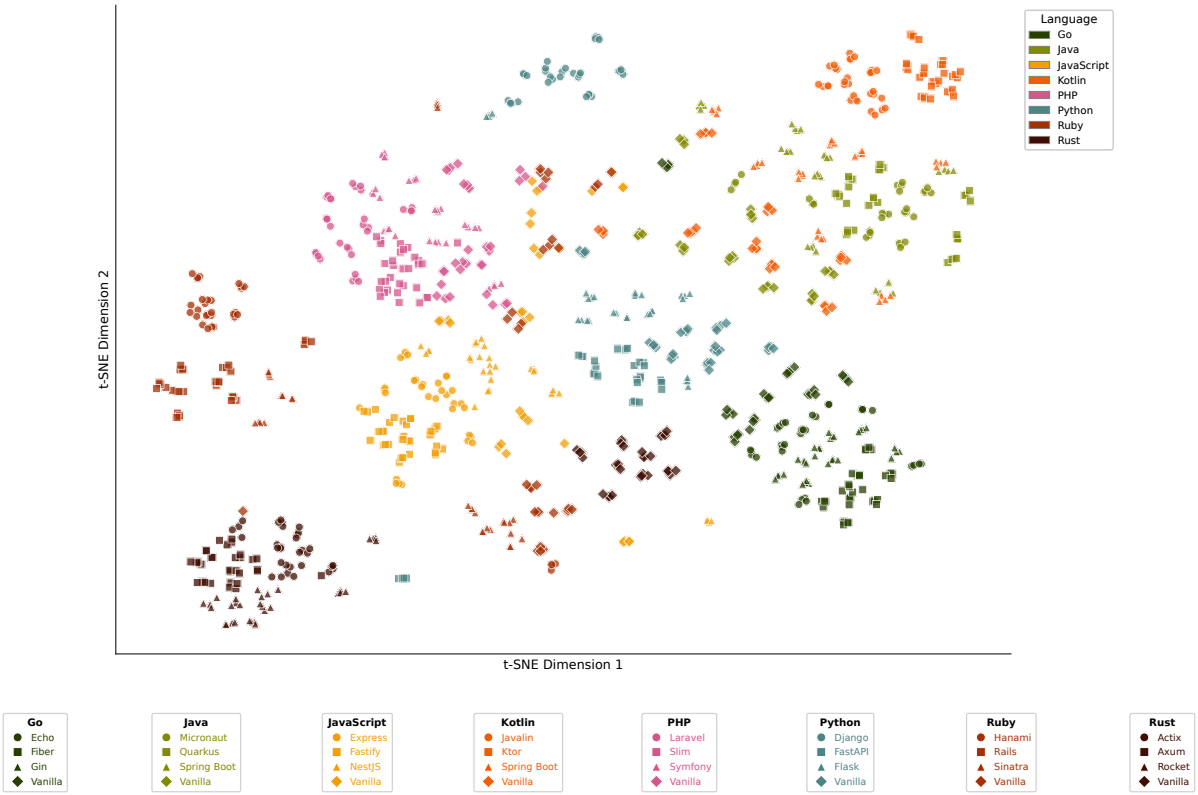

Figure 4: t-SNE projection of 1,024 embeddings under BGE-M3. Language clusters are tightly separated (visible as distinct point clouds); within-language framework sub-structure is most apparent for Ruby, Kotlin, and Rust, where silhouette scores of 0.216, 0.192, and 0.191 confirm quantitatively what the projection suggests visually.

language structure is recoverable at scale.

RQ2. Framework dialects are real but secondary to language identity. Five models detect a large effect ($d > 0.8$); UniXCoder a medium effect ($d = 0.679$); CodeBERT a small one ($d = 0.229$). Averaged across all seven models, Rust (+0.138), JavaScript (+0.130), and Python (+0.128) show the strongest dialect structure; Go (+0.087) is the weakest, consistent with its thin-wrapper framework design.

5.3 RQ3: Algorithmic Complexity Signal

Overall results Six of seven models produce statistically significant cross-/intra-complexity distance differences ($p < 0.001$). No model achieves a positive complexity silhouette, and d ranges from 0.047 (codebert, negligible) to 0.638 (Octen, medium), with the remaining five models falling in the small-to-medium range (Table 5).

The code-specialised model paradox CodeBERT achieves the lowest d overall ($d = 0.047$, negligible) due to geometric compression, while among the remaining six models

Model	Sil.	Cross	Intra	d	Effect
octen	−0.019	0.709	0.621	0.638	Medium
qwen3	−0.023	0.599	0.522	0.593	Medium
minilm	−0.014	0.607	0.538	0.540	Medium
ada002	−0.036	0.217	0.194	0.478	Small
bge_m3	−0.014	0.386	0.352	0.428	Small
unixcoder	−0.037	0.496	0.452	0.300	Small
codebert	−0.190	0.021	0.020	0.047	Neg.

Table 5: RQ3 cross-model summary. CodeBERT achieves a negligible effect ($d = 0.047$) and the worst silhouette (−0.190), far below the already weak baseline set by UniXCoder ($d = 0.300$).

UniXCoder achieves the smallest d (0.300), the same ranking pattern seen in RQ2. The direct explanation for UniXCoder's low score is that it has been trained to treat an $O(n)$ and an $O(n^2)$ solution to the same problem as functionally related. General-purpose models treat code more lexically: an $O(n^2)$ solution with two nested loops produces measurably different token statistics from an $O(n)$ solution, and consequently these models produce a stronger complexity signal. This parallels the RQ2 finding where UniXCoder showed weaker

framework separation for the same reason.

Compressed geometry versus semantic suppression The outlier pattern from RQ2 recurs: the geometrically compressed model's complexity silhouette (-0.190) is nearly five times more negative than any other model, and its $d = 0.047$ is negligible. Unlike UniXCoder's selective suppression of surface signals in favour of structural semantics, the compressed model renders even the weak complexity signal present in general-purpose models undetectable. Its cross-complexity cosine distance (0.021) differs from the intra-complexity distance (0.020) by only 0.001. The two code-specialised models thus show opposite failure modes: one encodes structure while ignoring surface; the other collapses all distances to near-zero. This indicates that *code specialisation alone does not uniformly improve embedding geometry.*

Language and complexity class effects Python achieves the least-negative silhouette across four of the six non-compressed models: its stylistic diversity across complexity classes (list comprehensions for $O(n)$, nested loops for $O(n^2)$, `bisect` for $O(\log n)$) provides a stronger surface signal. Rust consistently performs worst; its ownership annotations and iterator combinators introduce high intra-complexity variance. Across models, $O(n \log n)$ is the most geometrically isolated class, driven by the distinctive structure of sorting and divide-and-conquer algorithms; polynomial-degree distinctions ($O(n^2)$ vs. $O(n^3)$) are the least separable. Problem identity dominates complexity class: an $O(n)$ and an $O(n^2)$ solution to the *same* problem are more similar than two $O(n)$ solutions to *different* problems.

RQ3. Complexity class is weakly but consistently encoded across all models ($d = 0.047$–0.638); no model achieves a positive silhouette. Among non-compressed models, d ranges from 0.300 (UniXCoder) to 0.638 (Octen). Problem identity is the stronger axis. $O(n \log n)$ is the most isolated class; polynomial-degree distinctions are the least separable. CodeBERT achieves the lowest d overall ($d = 0.047$, negligible) due to geometric compression.

5.4 RQ4: Correctness Regions

Overall correctness clustering Across all seven models, the correctness silhouette is near zero (range 0.007–0.019). For four of seven models, language silhouette is higher than correctness silhouette, confirming that embeddings organise code by language rather than correctness. Ada-002 records the lowest correctness silhouette (0.007) while simultaneously showing the strongest language clustering (0.083).

Dangerous neighbourhoods At a cosine-distance threshold of 0.10, dangerous-neighbourhood rates range from 29.6% (UniXCoder) to 100% (CodeBERT). Table 6 gives the full breakdown across thresholds.

Thr.	CB	Ada	Qw3	Oct.	BGE	Mini.	UniX.
0.05	98.0	72.6	32.8	30.8	24.8	24.2	**17.6**
0.10	**100**	96.4	54.0	51.8	45.8	41.8	29.6
0.15	**100**	99.2	69.2	68.0	62.6	60.8	38.2

Table 6: Dangerous-neighbourhood rates (%) at three cosine-distance thresholds, sorted by rate at $\tau = 0.10$. The compressed-geometry model (CB) saturates at 100% from $\tau = 0.10$ onward; UniXCoder consistently achieves the lowest rates across all thresholds.

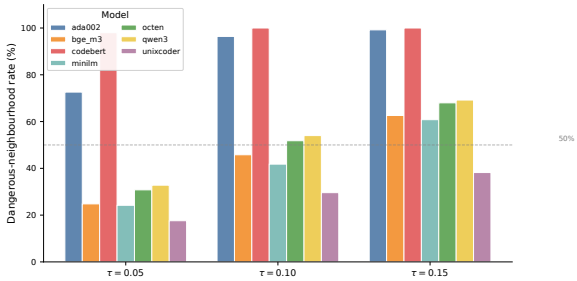

Figure 5: Dangerous-neighbourhood rates across all three thresholds. The compressed-geometry model saturates at 100% from $\tau = 0.10$ onward; UniXCoder achieves the lowest rates at all thresholds; Ada-002 is the second-worst performer despite its 1,536-dimensional embeddings.

Key findings CodeBERT saturates at 100% dangerous neighbourhoods from $\tau = 0.10$ onward; its mean buggy–fixed distance (0.011) is 20× smaller than UniXCoder's (0.223), making buggy and fixed code point-coincident. UniXCoder outperforms Ada-002 (0.039 mean distance) by nearly 6× despite having fewer than half as many dimensions, confirming that scale is not a proxy for correctness awareness. Six of seven models yield $p < 0.01$ but $d < 0.15$ throughout: the significance is a sample-size

artefact. CodeBERT is the exception: geometric compression eliminates even the tiny cross-cluster signal, yielding $p = 0.87$ and $d = -0.004$ (not significant). CodeBERT's correctness silhouette (0.018) falls within the range of all models (0.007–0.019) despite geometric collapse, illustrating that silhouette scores alone are insufficient; pairwise distance distributions must be examined directly. At $\tau = 0.10$, roughly half of all bug-fix pairs are nearest neighbours under most models; embedding-based code search and RAG pipelines may therefore return or inject buggy code indistinguishable from correct references.

RQ4. Static embeddings provide no reliable correctness axis. Six models yield $p < 0.01$ but $d < 0.15$ (negligible); CodeBERT fails significance entirely ($p = 0.87$, $d = -0.004$). Dangerous-neighbourhood rates at $\tau = 0.10$ range from 30% (UniXCoder, lowest/best) to 100% (CodeBERT, highest/worst). Ada-002, despite 1,536 dimensions, is the second-worst performer. Silhouette scores are positive but misleading when pairwise distances are uniformly tiny.

6 Cross-Cutting Analysis

Two code-specialised models, opposite geometries CodeBERT and UniXCoder share a code-specialised training domain yet behave as near-opposites across all four RQs. UniXCoder's cross-modal alignment objective produces representations invariant to surface idioms yet sensitive to structural semantics: it ranks third in clustering fidelity (CCC = 0.661), produces a medium framework-dialect effect ($d = 0.679$), and achieves the lowest dangerous-neighbourhood rate at every threshold. CodeBERT's replaced-token-detection objective combined with mean pooling collapses the embedding space into a narrow cone where cosine distances are one to two orders of magnitude smaller than those of any other model; it ranks last on all four measures (CCC = 0.584, $d = 0.229$ for frameworks, $d = 0.047$ for complexity, 100% dangerous neighbourhoods at $\tau = 0.10$). UniXCoder's behaviour is a coherent representational trade-off; CodeBERT's geometric collapse renders cosine similarity uninformative for any task. This contrast is the clearest evidence that code specialisation alone does not determine embedding geometry.

Dimensionality does not determine quality Ada-002 (1,536d) is the worst performer on correctness; UniXCoder (768d) is the best. The five open models spanning 384–1,024d show no consistent dimensional advantage. Most models in both RQ3 and RQ4 yield $p < 10^{-100}$ with negligible Cohen's d, confirming that significance driven by sample size is not evidence of practically meaningful separation.

Representational hierarchy Across all four RQs, a consistent ordering emerges: language identity is the strongest organising axis, followed by framework choice, then complexity class, then correctness. This hierarchy maps onto a syntactic-to-semantic axis: embedding sensitivity falls as the target property becomes more semantic. The compressed-geometry model is outside this ordering; its distance collapse renders all four levels undetectable simultaneously.

7 Conclusion

We characterised what seven code embedding models encode about programming languages, frameworks, algorithmic complexity, and code correctness. Language identity is robustly encoded across all models; framework choice creates a secondary signal whose strength varies across models; complexity class leaves a weak but consistent trace; and correctness is largely absent, with dangerous-neighbourhood rates from 30% to 100% at $\tau = 0.10$.

Two outlier patterns emerge across all four analyses. UniXCoder is a structured outlier: less sensitive to surface patterns, more sensitive to structural ones, and the best performer on correctness separation. CodeBERT exhibits geometric compression severe enough to suppress all four signals simultaneously. That both are code-specialised yet behave so differently confirms that code specialisation alone does not determine embedding geometry. For practitioners, model selection should include geometry diagnostics before deployment; correctness-sensitive retrieval requires augmentation beyond static embedding proximity.

Limitations

Synthetic data The RQ2 and RQ4 datasets were generated by Gemini, Claude Sonnet, and

GPT. Real-world code exhibits greater stylistic diversity; systematic patterns LLMs introduce may inflate or deflate separation signals relative to production corpora.

Embedding-only analysis We measure geometry in embedding space and do not evaluate downstream task performance. A correctness axis absent from static embeddings may be recoverable through fine-tuning or execution-augmented representations.

Model and language coverage The CodeBERT geometric-compression finding should be replicated with other bimodal pre-training models before generalising. The RQ4 dataset covers five languages; extending to all 19 would strengthen generalisability, especially for functional languages.

References

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, and 1 others. 2023. MultiPL-E: A scalable and polyglot approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering*, 49(7):3675–3691.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. BGE M3-Embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216*.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 8440–8451.

Kenneth Enevoldsen, Isaac Chung, Imene Kerboua, Márton Kardos, Ashwin Mathur, David Stap, Jay Gala, Wissam Siblini, Dominik Krzemiński, Genta Indra Winata, and 1 others. 2025. MMTEB: Massive multilingual text embedding benchmark. *arXiv preprint arXiv:2502.13595*.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547. Association for Computational Linguistics.

Rares Folea, Radu Iacob, Emil Slusanschi, and Traian Rebedea. 2023. Complexity-based code embeddings. In *Computational Collective Intelligence (ICCCI)*, volume 14162 of *Lecture Notes in Computer Science*. Springer Nature Switzerland.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified cross-modal pre-training for code representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 7212–7225.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, and 1 others. 2021. 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)*.

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2022. MTEB: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*.

Telmo Pires, Eva Schlinger, and Dan Garrette. 2019. How multilingual is multilingual BERT? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 4996–5001.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3982–3992.

Peter J. Rousseeuw. 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65.

Robert R. Sokal and F. James Rohlf. 1962. The comparison of dendrograms by objective methods. *TAXON*, 11(2):33–40.

Shangbo Yun, Xiaodong Gu, Jianghong Huang, and Beijun Shen. 2026. Beyond language boundaries: Uncovering programming language families for code language models. In *Proceedings of the ACM on Software Engineering (FSE)*.