RESEARCH ARTICLE

Introducing CC-SeDE: Code Clone Semantic Difficulty Estimator for Effective Type-Wise Classification of Similar Source Code Fragments

Zaira Bano[1] **and Mohd Zeeshan Ansari**[1,*]

[1] Department of Computer Engineering, Jamia Millia Islamia, New Delhi 110025, India

Abstract

Code clone detection aims to identify duplicate code fragments within source code repositories. Existing models give every code pair in a dataset the same learning weight and ignore the fact that pairs differ in semantic complexity. This work introduces a parameter, the Code Clone Semantic Difficulty Estimator (CC-SeDE), to quantify the degree of semantic divergence between the two fragments of a code pair. Rather than treating semantic divergence as a direct proxy for classification difficulty, the approach provides a continuous measure that allows clone pairs to be characterized and stratified across the divergence spectrum. In the validation protocol the parameter is computed from token overlap and relative code-length difference, so that it does not depend on pretrained embedding representations. Experiments on the BigCloneBench benchmark examine the distribution of the parameter, its variation across clone categories, clone density across parameter intervals, and its relationship with lexical and structural characteristics. Higher values of the parameter are associated with greater lexical and structural divergence. These findings suggest that the parameter is useful as an interpretable indicator for identifying and analyzing highly divergent clone pairs, which are hypothesized to be harder to detect. The parameter is introduced and evaluated as a conceptual and descriptive measure of code divergence, not as an empirically validated estimator of semantic complexity or of model-specific classification difficulty. The work provides a basis for difficulty-aware learning in code clone detection.

Keywords: code clone detection, code clone classification, semantic similarity, difficulty-aware learning.

1 Introduction

Software systems often contain repeated or duplicated code fragments, known as code clones, that are usually introduced during development. Detecting these clones is important for software maintenance, refactoring and plagiarism prevention. Traditional methods relied on syntactic similarity, such as token or abstract syntax tree (AST) comparison [1], but they struggle to capture semantic equivalence between

Submitted: 03 July 2026
Revised: 30 October 2026
Accepted: 01 November 2026
Published: 20 November 2026

Vol. 1, **No.** 1, 2026.
10.62762/ICE.2026.187437

***Corresponding author:**
✉ Mohd Zeeshan Ansari
mzansari@jmi.ac.in

Citation
Bano, Z., & Ansari, M. Z. (2026). Introducing CC-SeDE: Code Clone Semantic Difficulty Estimator for Effective Type-Wise Classification of Similar Source Code Fragments. *Intelligent Computing for Engineering*, 1(1), 30–42.

© 2026 by the Authors. Published by Institute of Central Computation and Knowledge. This is an open access article under the CC BY license (https://creativecommons.org/licenses/by/4.0/).

structurally different code fragments. Machine learning, deep learning and transformer-based models [2] have changed code clone detection by learning distributed representations that capture code semantics. Type-1 and Type-2 clones are generally considered easier to detect, whereas Type-3 and Type-4 clones are harder [3].

However, these models assume that all code pairs are equally difficult to learn. In reality some pairs are trivially identical, while others are semantically similar yet syntactically divergent and therefore harder to classify. Semantic clones that involve control-flow restructuring, algorithmic refactoring or other logic-preserving transformations require deeper contextual understanding. This variability creates a continuous spectrum of divergence that discrete clone categories do not fully capture. Pairs with high lexical and structural similarity can often be recognized through direct surface-level correspondence, whereas pairs with substantial lexical and structural differences may require reasoning over the relationships between program elements. Two pairs with the same clone label may thus present very different levels of divergence to a clone-detection system. This motivates a finer-grained characterization of clone pairs: instead of treating clone difficulty as a uniform property of a dataset, or as an inherent property of a discrete clone type, it is useful to examine how individual pairs vary along a continuous spectrum of lexical and structural divergence.

To address this gap, we propose the Code Clone Semantic Difficulty Estimator (CC-SeDE), a quantitative measure of how semantically complex or ambiguous a code pair is. Its score ranges can also be read against the conventional clone types. Because the score is computed per pair, it can later support difficulty-aware learning, in which a model adjusts its focus during optimization; the present study characterizes the score itself and does not train a detector. The main contributions of this work are as follows.

- We introduce CC-SeDE, a difficulty-aware framework for characterizing code pairs in type-based code clone analysis. It allows code pairs that are functionally similar despite substantial divergence to be examined systematically, and it gives a continuous representation of lexical and length-based differences.
- We relate semantic difficulty to clone characteristics through a conceptual analysis of code divergence against the conventional Type-1 to Type-4 taxonomy, showing how different degrees of lexical and length-based difference correspond to code variation from syntactic clones to highly divergent semantic clones.
- An empirical analysis reports the prevalence, statistical properties and distribution of the parameter across the semantic difficulty spectrum.
- Clone density across the semantic difficulty spectrum is analyzed, showing how the proportion of clone pairs changes as semantic difficulty increases and how easy and hard clone instances are distributed.
- We conduct an exploratory analysis that examines the joint distribution of token overlap, length difference ratio and the proposed parameter, to characterize how lexical and structural differences are reflected across the divergence spectrum.

The rest of this paper is organized as follows. Section 2 presents the background and motivation. Section 3 defines the proposed estimator, and Section 4 describes the operational formulation used for validation and its relation to the clone taxonomy. Section 5 reports the experimental setup and results, Section 6 discusses threats to validity, and Section 7 concludes the paper and outlines future work.

2 Background and Motivation

Classical clone detection techniques such as CCFinder [4], SourcererCC [5], NiCad [6] and Deckard [7] rely on token-based lexical analysis, tree-based matching, agile parsing or code normalization. These methods are effective for syntactic clones (Type-1 and Type-2) and, in the case of NiCad and SourcererCC, for many near-miss Type-3 clones, but they are of limited use for semantic (Type-4) clones. Machine learning-based methods draw on a wide range of techniques, from bytecode dependency graphs to graph attention networks, and on algorithms such as DBSCAN and XGBoost [8]. Deep learning-based approaches such as code2vec [9], ASTNN [10], CodeBERT [11] and GraphCodeBERT [12] learn vector embeddings that represent program semantics. ASTNN (AST-based neural network) captures lexical and syntactic features of code fragments and feeds them to a bidirectional recurrent neural network (RNN) to produce the final vector representation of a fragment. Transformer-based models like CodeBERT are pretrained on pairs of code and natural language and

perform well in code search and other downstream code understanding tasks. In all these methods, however, every training pair is treated equally, regardless of its semantic difficulty.

Difficulty-aware learning has been used outside code clone detection to make models more robust in natural language processing and computer vision. The idea is rooted in curriculum learning, where a model is trained on easier examples before harder ones [13]. In automatic distractor generation for gap-fill items, difficulty-aware modeling has been used to account for the varying difficulty of distractors [14]. Difficulty-aware strategies have also been applied to semantic image segmentation, either by processing easy pixels with less computation and passing hard pixels to deeper layers [15], or by prioritizing difficult samples in active learning [16], and to time-continuous emotion prediction by adjusting training to sample difficulty [17]. These studies show that treating samples according to their difficulty is a recognized strategy outside code analysis.

Motivated by work on estimating semantic relatedness between source code terms [18] and on graded semantic textual similarity in natural language processing [19], both of which treat similarity as a continuous quantity rather than a binary label, this study introduces a continuous semantic difficulty metric that stratifies code clone pairs by their semantic complexity. Unlike those works, the operational formulation used here relies only on token overlap and code-length difference (Section 4). The metric allows model performance to be evaluated at different semantic levels and gives a more realistic picture of how well deep learning models capture fine-grained program semantics. To the best of our knowledge, semantic difficulty has not previously been modeled as a continuous, pair-level quantity in code clone detection. The closest work applied curriculum learning to clone detection on CodeXGLUE but ordered samples by code length and cyclomatic complexity rather than by the divergence between the two fragments of a pair [20]. Our approach adds this pair-level perspective and proposes a framework that future researchers could use as an additional evaluation dimension. By introducing an explicit semantic difficulty estimator, we stratify the dataset according to the degree of semantic similarity, so that a model can be trained on, or evaluated across, different difficulty levels.

3 Proposed Difficulty Estimator

Benchmark datasets like BigCloneBench [21] provide a coarse clone classification with no continuous notion of semantic hardness. CC-SeDE addresses this by defining and computing a semantic difficulty score. Figure 1 presents the analytical framework used in this study to characterize code clone pairs by type.

The framework focuses on difficulty-aware dataset analysis and examines the lexical, structural and semantic characteristics of code clone pairs. Three characteristics are extracted for each code pair: token overlap, which represents lexical similarity; length difference ratio, which reflects structural variation; and CC-SeDE, which quantifies semantic divergence between the code fragments. Together they describe clone complexity from lexical, structural and semantic perspectives. Code pairs are then stratified by semantic difficulty, with emphasis on the high-divergence region where CC-SeDE exceeds 0.6. This stratification isolates hard instances that show little surface similarity while possibly preserving functional equivalence. The identified population is examined through frequency distribution analysis, stratification across clone types, clone density analysis across the difficulty spectrum, and a multivariate analysis of semantic difficulty against lexical and structural characteristics. These analyses characterize the hard semantic clone pairs and point to the challenges they pose for current clone detectors.

We define CC-SeDE as a measure of how far the similarity of a code pair falls short of complete similarity. Let f_1 and f_2 denote two code snippets, $\phi(\cdot)$ an embedding function, and e_1 and e_2 the corresponding vector representations, as shown in Eq. (1).

$$e_1 = \phi(f_1), \qquad e_2 = \phi(f_2) \tag{1}$$

The semantic similarity between the two fragments is then computed with cosine similarity, as in Eq. (2). For general embeddings its value lies in $[-1, 1]$, and in $[0, 1]$ when the embedding components are non-negative; higher values correspond to stronger semantic similarity.

$$\mathrm{SS}(f_1, f_2) = \frac{e_1 \cdot e_2}{\|e_1\| \, \|e_2\|} \tag{2}$$

CC-SeDE is derived from semantic similarity as its complement, which ensures a monotonic relationship between semantic dissimilarity and semantic difficulty:

$$\text{CC-SeDE} = g\big(1 - \mathrm{SS}(f_1, f_2)\big) \tag{3}$$

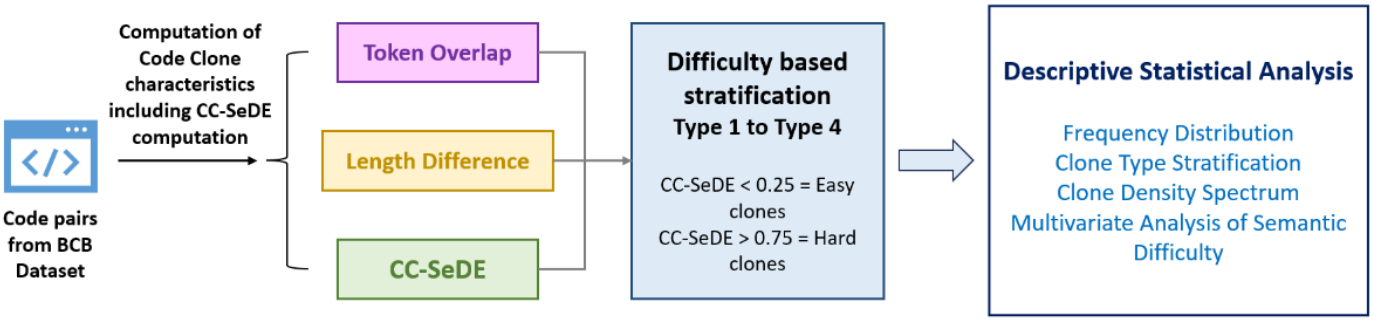


Figure 1. Proposed analytical framework.

where $g(\cdot)$ denotes a monotonic transformation that maps semantic dissimilarity between code fragments to a divergence score in $[0, 1]$. These equations provide the conceptual basis for vector-based assessment of code similarity and motivate the proposed parameter. We choose cosine similarity because, for semantic similarity, the angle between vectors is more meaningful than the difference in their magnitudes: semantic information is encoded mainly in the direction of code embedding vectors rather than in their length. Cosine similarity gives a stable and scale-invariant measure of semantic proximity, but other similarity measures, such as Euclidean distance, Manhattan distance, dot-product similarity, Mahalanobis distance, angular distance, a Siamese similarity score, Jaccard similarity or other learned similarity functions, may also be used.

Mathematical definitions for each of these measures are given in Table 1; e_1 and e_2 are computed as in Eq. (1). Supporting several similarity functions makes the parameter applicable to different embedding representations and adaptable to application-specific requirements.

Code clones may preserve the same functionality while differing substantially in their surface representation. Measuring similarity between two code fragments therefore requires considering how far their observable characteristics diverge. In general, the similarity between two code fragments can be represented in a vector space, where semantic similarity may be quantified with measures such as cosine similarity. This gives a simple basis for relating similarity to divergence: higher similarity corresponds to lower divergence, and lower similarity to greater divergence.

4 Validation Protocol

The embedding-based formulation in Eqs. (1)-(3) is a conceptual reference for quantifying similarity between code representations with vector-based measures. These equations describe a general approach to code similarity, not the operational formulation adopted in this study. For the experiments, CC-SeDE is computed only from token overlap and relative code-length difference, as defined in Eq. (7), without a pretrained embedding model or embedding-based cosine similarity. This separates the discussion of alternative similarity measures from the actual computation and evaluation of the proposed divergence score. CC-SeDE thus uses two directly measurable characteristics of a code pair, token overlap and length difference ratio, to build a transparent and reproducible continuous divergence score, and semantic divergence can be examined without model-dependent embedding choices.

Let T_1 and T_2 denote the sets of tokens extracted from code fragments f_1 and f_2, respectively. Token overlap (TO) is calculated with the Jaccard similarity:

$$\mathrm{TO}(f_1, f_2) = \frac{|T_1 \cap T_2|}{|T_1 \cup T_2|} \tag{4}$$

A higher TO indicates greater lexical correspondence between the two fragments. Lexical divergence (LD) is its complement:

$$\mathrm{LD}(f_1, f_2) = 1 - \mathrm{TO}(f_1, f_2) \tag{5}$$

In addition to lexical divergence, differences in code length are considered. Let L_1 and L_2 denote the numbers of tokens, counted with repetition, extracted from f_1 and f_2. The length difference ratio (LDR) is defined as:

$$\mathrm{LDR}(f_1, f_2) = \frac{|L_1 - L_2|}{\max(L_1, L_2)} \tag{6}$$

Finally, CC-SeDE is obtained by combining the two normalized divergence components with equal weights:

$$\text{CC-SeDE} = 0.5\,\mathrm{LD} + 0.5\,\mathrm{LDR} = 0.5\,(1 - \mathrm{TO}) + 0.5\,\mathrm{LDR} \tag{7}$$

Table 1. Calculation of CC-SeDE based on various similarity functions.

Method	CC-SeDE formulation
Cosine similarity	$\frac{1}{2}\left(1-\frac{e_1\cdot e_2}{\|e_1\|\,\|e_2\|}\right)$
Angular distance	$\frac{\arccos\big(\mathrm{SS}(f_1,f_2)\big)}{\pi}$
Euclidean distance	$\frac{\|e_1-e_2\|_2}{\|e_1\|_2+\|e_2\|_2}$
Manhattan distance	$\frac{\|e_1-e_2\|_1}{\|e_1\|_1+\|e_2\|_1}$
Mahalanobis distance	$\frac{d_M}{d_M+1},\quad d_M=\sqrt{(e_1-e_2)^\top\Sigma^{-1}(e_1-e_2)}$
Dot product	$1-\frac{\sigma(e_1\cdot e_2)}{\sigma(\|e_1\|_2\,\|e_2\|_2)}$
Siamese similarity score	$1-\mathrm{Model}(f_1,f_2)$
Jaccard similarity	$1-\frac{\|A\cap B\|}{\|A\cup B\|}$

Note: SS is the cosine similarity of Eq. (2); Σ is the covariance matrix of the code embeddings; $\sigma(\cdot)$ is the logistic sigmoid; $\mathrm{Model}(f_1,f_2)\in[0,1]$ is the output of a trained Siamese network; A and B are the token sets of f_1 and f_2.



Figure 2. Conceptual semantic difficulty spectrum of CC-SeDE (0.0 to 1.0).

The resulting score lies on a continuous scale from 0 to 1, where larger values indicate greater lexical and length-based divergence between the two code fragments. The measure is therefore used as an analytical parameter for characterizing semantic divergence in labeled clone pairs, not as a learned embedding distance or a universally calibrated boundary of classification difficulty.

CC-SeDE can be read as a lexical-to-semantic continuum rather than only as a difficulty score, as shown in Figure 2. A higher value indicates a harder pair with low lexical overlap, and a lower value an easier pair with high lexical overlap.

Conventional code clone taxonomies classify code pairs according to the nature and extent of their differences. CC-SeDE characterizes observable differences between code fragments through token overlap and relative length difference. Conceptually, code pairs with high token overlap and small length differences show characteristics commonly associated with Type-1 or Type-2 clones, whereas pairs with lower token overlap or larger length differences may show variations also found in Type-3 or Type-4 clones. These relationships are illustrative rather than deterministic. Clone types are defined by properties that token overlap and length difference do not fully capture; in particular, Type-4 classification depends on functional or semantic equivalence, which the proposed lexical and length-based measure cannot establish. CC-SeDE is not intended to replace the clone taxonomy or to provide a validated mapping from score intervals to clone types. The taxonomy only provides a conceptual context for interpreting the forms of code variation represented by different scores. Establishing an empirical correspondence between CC-SeDE values and clone types would require independently annotated clone-type data and systematic validation, and the present study does not treat the scores as definitive indicators of clone type or semantic equivalence. Although CC-SeDE is discussed in the context of the clone taxonomy, it does not analyze ASTs, control-flow or data-flow structures, or program behavior, so its relationship with clone types is conceptual rather than a validated type-prediction mechanism.

For interpretation only, the CC-SeDE scale can be divided into four equal-width intervals and read against the conventional Type-1 to Type-4 taxonomy [22], as shown in Table 2.

The association between CC-SeDE ranges and clone

Table 2. Conceptual correspondence between CC-SeDE ranges and the conventional code clone taxonomy.

CC-SeDE range	Clone type	Description	Interpretation
0.00-0.25	Type-1	Identical fragments except for variations in whitespace, layout and comments	Very low divergence; high lexical similarity
0.25-0.50	Type-2	Syntactically identical fragments except for variations in identifiers, literals, types, layout and comments	Low divergence; minor lexical changes
0.50-0.75	Type-3	Copied fragments with further modifications: statements changed, added or removed	Moderate divergence; reduced lexical overlap
0.75-1.00	Type-4	Fragments that perform the same computation but are implemented with different syntax	High divergence; minimal lexical similarity

types is a conceptual interpretation of observable code divergence, not an empirically validated classification rule. The ranges characterize lexical and length-based differences and should not be read as definitive boundaries between Type-1, Type-2, Type-3 and Type-4 clones. Under this reading, the discrete notion of clone type can be complemented by a continuous spectrum of divergence along which lexical and length-based differences increase with CC-SeDE. Clone type and divergence are treated here as two descriptive perspectives, not as an established single continuum.

The parameter could therefore serve two purposes in future work: as a weighting or ordering signal for difficulty-aware learning, and as a coarse, interpretable indicator of the kind of variation a pair exhibits. Neither use is evaluated here, and the score does not replace clone-type labels (Section 6.8). Because it is computed independently of any detector, the parameter supports difficulty-aware analysis without changes to the underlying clone detection architecture and can be applied across different clone models and benchmark datasets.

5 Experiments and Results

5.1 Experimental Setup

We conduct experiments on the CodeXGLUE version [23] of BigCloneBench, whose training split contains 901,028 labeled code pairs. The dataset includes both clone and non-clone pairs, with binary labels indicating whether the two fragments form a clone pair. To examine CC-SeDE as an indicator of semantic hardness, the analysis focuses on pairs with a parameter value greater than 0.6. This threshold is an operational cutoff that selects pairs from the upper part of the observed divergence distribution, so that pairs with comparatively large lexical divergence and length differences can be examined descriptively. It is an exploratory selection criterion, not a statistically validated boundary between easy and difficult pairs. Of the 901,028 code pairs, 305,856 pairs with CC-SeDE above the threshold were identified as the high-divergence population. For computationally intensive experiments, a reproducible balanced subset of 10,000 pairs, comprising 5,000 clone and 5,000 non-clone pairs, was drawn from this population. The full population of 305,856 pairs was used for distributional analysis, and the smaller subset for controlled computational experiments. The analysis therefore describes the distribution of observable lexical and length-based divergence within the selected population, and the balanced subset provides a controlled sample for comparing these characteristics between clone and non-clone pairs. Other CC-SeDE thresholds can be applied in the same way to investigate other regions of the divergence distribution.

5.2 Statistical Characterization of the Proposed Parameter

Table 3 reports descriptive statistics for token overlap, length difference ratio and CC-SeDE computed over the high-divergence population (CC-SeDE $>$ 0.6) described in Section 5.1. Token overlap, as used in token-based clone detectors such as SourcererCC [5], measures the overlap between the tokens of two code blocks to compute their pairwise similarity. In this study it is computed as the Jaccard similarity of the token sets, as in Eq. (4). Length difference

ratio is a structural divergence metric based on the difference in length between two code fragments. These metrics describe surface-level divergence and allow CC-SeDE to be related to observable differences between the fragments. The statistics cover central tendency, dispersion and spread.

These pairs show low token-level overlap but substantial differences in code length, which results in consistently high CC-SeDE values. Token overlap has a mean of 0.1264 and a median of 0.1167, so the high-difficulty pairs share relatively few lexical tokens. Its maximum is only 0.3673, which shows that even the most lexically similar pairs in this population have limited token-level correspondence.

The length difference ratio has a much higher mean of 0.6343 and a median of 0.6364. The interquartile range from 0.5055 to 0.7638 shows that large differences in code length are typical of this population. The maximum of 0.9933 means that some pairs have almost completely different code lengths. These statistics cover both clone and non-clone pairs.

CC-SeDE has a mean of 0.7540, with values from 0.6000 to 0.9827. The median of 0.7478 and the relatively narrow interquartile range of 0.6744-0.8264 show that the selected pairs mostly lie in a clearly divergent region of the scale. The minimum of 0.6000 confirms that the statistics refer to the high-difficulty population and not to the complete dataset.

The high-CC-SeDE population is thus characterized by limited lexical correspondence combined with large length divergence. This supports using CC-SeDE as a continuous measure for examining semantic divergence within the analyzed code pairs, instead of treating the conventional clone categories as discrete categories.

Within this population, token overlap is concentrated in a narrow band (standard deviation 0.0590), which limits its ability to separate pairs. The length difference ratio varies much more (standard deviation 0.1652), but it reflects differences in size rather than in content. CC-SeDE lies between the two (standard deviation 0.0940) because it averages a lexical and a length-based component. This intermediate spread should be read with care: CC-SeDE is a linear combination of the other two measures, and the population was selected on CC-SeDE itself, which truncates its range at 0.6. Its median of 0.7478 therefore follows from the selection and does not on its own show greater sensitivity to semantic transformations. The table does allow a consistency check: by Eq. (7), the mean CC-SeDE should equal $0.5 \times (1 - 0.1264) + 0.5 \times 0.6343 = 0.7540$, which matches the reported value.

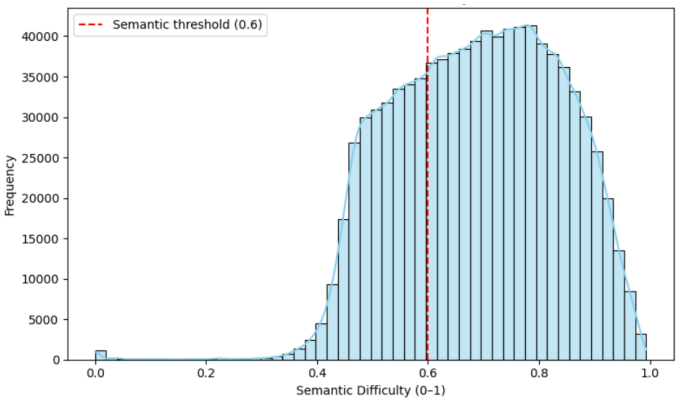


Figure 3. Frequency distribution of CC-SeDE scores (0-1 range).

5.3 Distributional Analysis

Figure 3 shows the empirical distribution of CC-SeDE computed over all code pairs in the training split. The horizontal axis gives CC-SeDE values in $[0, 1]$ and the vertical axis the frequency. Nearly all pairs fall between 0.4 and 1.0, with the largest concentration between 0.6 and 0.8, and very few pairs lie between 0.05 and 0.35. The range below 0.35 is therefore sparsely populated, and the 0.25-0.50 interval associated with Type-2 in Table 2 contains pairs mainly in its upper part, above 0.4.

Apart from a small spike at 0, which corresponds to near-identical pairs, the distribution is unimodal: it rises steeply above 0.4, peaks near 0.78 and falls off towards 1.0. Semantic difficulty therefore appears as a graded quantity rather than a binary easy/hard property, which supports designing CC-SeDE as a continuous metric rather than a categorical classifier. The concentration of pairs above 0.6 is also consistent with choosing this value as the operational threshold for high-difficulty pairs. Under Eq. (7), a score above 0.6 means that the average of lexical divergence and length difference ratio exceeds 0.6.

Values close to 1.0 indicate pairs with very low similarity, while the small mass near 0 reflects near-identical clones. The dataset thus contains a large proportion of challenging instances, which underlines the need for semantics-aware clone detection models. The histogram shows no artificial clustering at the interval boundaries, which is consistent with a well-behaved continuous score.

Table 3. Summary statistics for high-difficulty pairs (CC-SeDE > 0.6).

	Mean	Std	Min	25%	50%	75%	Max
Token overlap	0.1264	0.0590	0.0050	0.0833	0.1167	0.1600	0.3673
Length difference ratio	0.6343	0.1652	0.2558	0.5055	0.6364	0.7638	0.9933
Proposed CC-SeDE	0.7540	0.0940	0.6000	0.6744	0.7478	0.8264	0.9827

Table 4. Clone type distribution (%) among clone pairs with CC-SeDE > 0.6.

Clone type	Percentage (%)
Type-3	51.1553
Type-4	37.6990
Type-2	10.9204
Type-1	0.2251

Table 5. Clone ratio across the difficulty spectrum.

CC-SeDE range	Clone ratio (%)
0.00-0.25	99.2607
0.25-0.50	59.3848
0.50-0.75	49.5444
0.75-1.00	47.7070

5.4 Semantic Difficulty Stratification Across Clone Types

Table 4 and Figure 4(a) give the percentage distribution of clone types among the clone pairs with CC-SeDE greater than 0.6; non-clone pairs have no clone type and are excluded from this distribution. This distribution is consistent with the intended behavior of the estimator.

Type-1 clones, which are exact or near-exact duplicates, are almost entirely excluded by the threshold. This agrees with the expectation that trivial lexical duplicates should have low semantic difficulty. The small proportion of Type-2 clones likewise indicates that simple identifier renaming or formatting changes rarely meet the high-difficulty condition. In contrast, 88.85% of the retained clone pairs belong to the Type-3 and Type-4 categories, which involve substantial structural modification or semantic re-implementation. A threshold of 0.6 therefore filters out most syntactic clones while retaining semantically complex pairs, in line with established clone taxonomy definitions. At the same time, the 10.92% of Type-2 pairs above 0.6, outside the 0.25-0.50 interval associated with Type-2 in Table 2, confirms that those intervals should not be used as type boundaries. The dominance of Type-3 and Type-4 clones suggests that the high-value population is a demanding evaluation setting for clone detection models, since these types require semantic reasoning rather than lexical matching. Because the clone-type labels are assigned independently of CC-SeDE, this agreement gives some external support to the view that the parameter captures complexity beyond surface-level similarity.

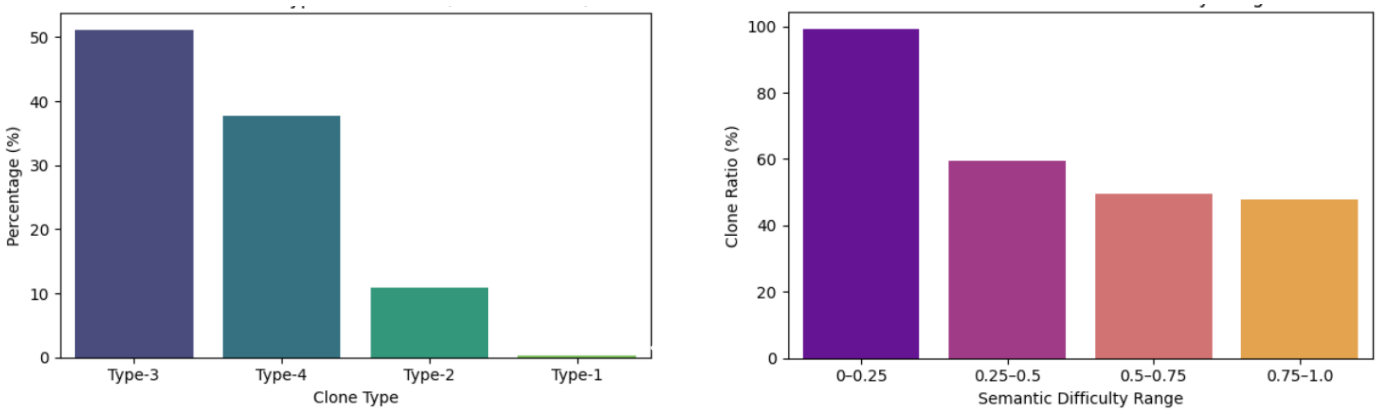
5.5 Clone Density Across the Semantic Difficulty Spectrum

The proportion of true clone pairs in four CC-SeDE intervals spanning $[0, 1]$ is shown in Table 5 and Figure 4(b). The clone ratio declines monotonically as semantic difficulty increases, steeply between the first two intervals and gradually afterwards. In the lowest interval (0.00-0.25) the clone ratio is 99.26%, so nearly all low-value pairs are genuine clones. This agrees with the interpretation that low semantic difficulty reflects strong semantic and lexical overlap.

Higher CC-SeDE values thus correspond to increasingly ambiguous or semantically distant pairs. The upper intervals still contain a large proportion of true clones, but their near-balanced ratio of clones to non-clones makes them harder discrimination settings. Statistically, the parameter stratifies the dataset along a difficulty axis: low-value regions are dominated by trivial or highly similar clones, whereas high-value regions approach a near-balanced classification setting. This behavior indicates that CC-SeDE stratifies the dataset in a way that tracks increasing ambiguity between clones and non-clones. The gradual decline across intervals in Figure 4(b) supports reading the parameter as a continuous difficulty gradient rather than a categorical split, and suggests that it is suitable for difficulty-aware evaluation protocols.

5.6 Exploratory Multivariate Visualization of Semantic Difficulty

Figure 5 gives an exploratory joint view of token overlap, length difference ratio and CC-SeDE for pairs spanning the full CC-SeDE range, not only the high-divergence population. Higher values



(a) (b)

Figure 4. (a) Clone type distribution of clone pairs with CC-SeDE > 0.6; (b) clone ratio across CC-SeDE intervals.

generally occur where token overlap is low and length differences are large. Because CC-SeDE is constructed from these two components, these patterns are expected; the plot is used to interpret the distribution visually, not to establish statistically significant associations.

Pairs with high token overlap and a low length difference ratio mostly have low CC-SeDE values and are easy instances; they lie in the lower-right region of the plot, where overlap is high and structural change is minimal. Pairs with high CC-SeDE are concentrated where lexical overlap is low and structural divergence moderate to high. This separation follows from the definition of the score, and the plot mainly shows where the pairs lie in this space. The color changes gradually rather than abruptly across the feature space.

The plot highlights three descriptive patterns: lower token overlap generally goes with higher CC-SeDE values; larger length differences generally go with higher values; and pairs with both low lexical overlap and large length differences tend to occupy the upper end of the spectrum. These patterns show how the parameter jointly reflects lexical and length-based divergence and give an interpretable representation of divergence across clone pairs. The visualization does not, however, independently establish semantic sensitivity, discriminative capability or the actual difficulty of detecting individual pairs. These properties would require evaluation against independent semantic annotations, expert assessments or the predictions of clone detection models, which are beyond the scope of this study.

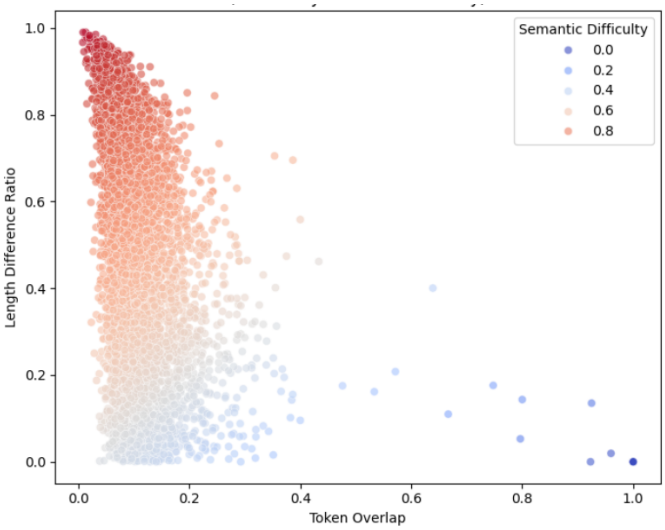


Figure 5. Token overlap versus length difference ratio, colored by CC-SeDE.

5.7 Summary of Empirical Findings

The preceding results indicate that CC-SeDE gives a continuous and interpretable representation of semantic divergence across code pairs, while the predefined intervals provide a convenient stratification for analysis. A substantial subset of pairs with high semantic difficulty is identified, which shows that the parameter can separate pairs with greater divergence from the broader collection. CC-SeDE is a continuous score in $[0, 1]$ rather than a discrete variable, and its frequency distribution spans the difficulty spectrum, so the four intervals act as divergence strata rather than arbitrary categorical labels.

The composition of the high-difficulty population shows how different clone categories are represented as divergence increases. Across the complete spectrum, the clone ratio falls from 99.26% in the 0.00-0.25 interval to 47.71% in the 0.75-1.00 interval. This change shows that the scale captures meaningful variation in the relationship between semantic divergence and clone occurrence, and that CC-SeDE separates highly

similar pairs from increasingly divergent regions of the clone-pair space.

The clone type distribution of the high-difficulty pairs shows that these pairs are mainly Type-3 (51.16%) and Type-4 (37.70%) clones, with smaller shares of Type-2 and Type-1, and that the difficulty spectrum can be used to characterize different regions of the clone population. The joint plot of token overlap and length difference ratio, colored by the parameter value, links the score to observable lexical and structural characteristics of the code pairs, so that CC-SeDE values correspond to recognizable patterns of code divergence rather than to numbers in isolation.

The statistical, distributional and multivariate results together indicate that CC-SeDE forms an interpretable divergence scale, and that the four intervals are useful analytical strata for examining progressively different levels of divergence. The evidence supports the use of CC-SeDE for semantic-difficulty stratification of clone pairs, while the Type-1 to Type-4 terminology is used as a descriptive point of comparison and the numerical thresholds are not assumed to be equivalent to the clone taxonomy.

6 Threats to Validity

6.1 Construct Validity

In this study, semantic divergence is operationalized through token overlap and length difference ratio. These measures are transparent and reproducible indicators of observable code divergence, but they do not directly model program semantics, execution behavior, control flow or data flow. CC-SeDE should therefore be interpreted as an operational measure of divergence rather than a complete representation of semantic equivalence. Future work may use richer program representations, such as ASTs, control-flow graphs (CFGs), data-flow graphs (DFGs) or learned code embeddings, to study semantic divergence further.

6.2 Internal Validity

The study uses a fixed weighting scheme that gives equal importance to lexical divergence and length disparity when computing CC-SeDE. Other weightings could produce different score distributions. The analytical cutoff of 0.6 was used to examine the high-divergence region and should not be read as an optimized classification threshold. Although the same computation was applied to all code pairs, other parameterizations may yield different stratification patterns.

6.3 External Validity

The experiments used BigCloneBench through the CodeXGLUE benchmark, which consists of Java code pairs. The findings may therefore not generalize directly to other programming languages, application domains or industrial code repositories. Studies on datasets from multiple languages and software ecosystems are needed to determine how widely the observed divergence patterns apply.

6.4 Conclusion Validity

The study mainly performs descriptive and exploratory analyses of divergence distributions. Its conclusions are empirical observations about the analyzed dataset, not universal statistical laws. The reported distributions, stratifications and visualizations identify patterns within the studied population but do not establish causal relationships between semantic divergence and clone-detection performance.

6.5 Label Validity

A further threat arises from the use of BigCloneBench labels as ground truth. The dataset provides binary clone/non-clone annotations, while this study investigates semantic divergence within those labeled pairs. Recent research has pointed out limitations of BigCloneBench for evaluating semantic clone detection, particularly for weak Type-3 and Type-4 clones, where semantic equivalence may be hard to define objectively [24]. Inaccurate or inconsistent benchmark labels may therefore affect the observed CC-SeDE distributions, and the findings should be interpreted in the context of the available annotations rather than as definitive measurements of semantic equivalence.

6.6 Dataset Bias and Sampling Validity

The study uses an analysis subset derived from the original BigCloneBench data. Although the subset was created to keep the experiments computationally tractable, sampling may introduce biases relative to the full dataset. In addition, the proportion of clone and non-clone pairs in BigCloneBench may not reflect the prevalence of clones in real-world software systems. These factors may affect the observed distributions and category proportions.

6.7 Model Dependence

The CC-SeDE formulation is intentionally model-independent and does not rely on a

specific clone-detection architecture. However, the relationship between semantic divergence and classification difficulty may vary across clone-detection models and feature representations. The present study therefore characterizes semantic divergence and makes no claim of universal behavior across clone-detection systems.

6.8 Limitations of the Taxonomy Correspondence

The conceptual relationship between CC-SeDE and the Type-1 to Type-4 taxonomy has not been empirically validated in this study. The benchmark labels used in the analysis do not by themselves establish a correspondence between individual parameter values and specific clone types, and lexical overlap and code-length differences cannot reliably distinguish all forms of syntactic variation from semantic equivalence. The proposed relationship should therefore be regarded as a conceptual framework for discussing code divergence, not as evidence that CC-SeDE can classify clone types automatically. Validation on independently annotated clone-type data would be needed to establish the strength and reliability of this correspondence.

7 Conclusions and Future Work

This study introduced the Code Clone Semantic Difficulty Estimator (CC-SeDE) as a continuous and quantitative measure of semantic difficulty in code clone detection. Unlike traditional clone taxonomies that place clones in discrete types, it gives a spectrum-based representation, computed in this study from token overlap and relative length difference. The experimental results showed that the parameter varies consistently with lexical divergence, length variation and the benchmark clone categories, and that it behaves as a coherent and interpretable indicator of clone divergence.

The study examined the distribution of CC-SeDE across clone pairs and identified a substantial population with high divergence. The analysis of this subset, together with the distributional statistics and visualizations, shows that clone pairs can differ considerably in lexical and structural terms while keeping their benchmark clone labels. The results also show systematic changes in clone prevalence across the parameter strata and illustrate the relationship between token overlap, length difference and the resulting divergence scores. These findings provide an empirical basis for treating semantic divergence as a continuous property instead of assuming that clone categories alone capture the full range of variation among clone pairs.

Several limitations point to directions for future research. First, the current operationalization relies on token overlap and code-length disparity and does not directly capture deeper semantic properties such as control-flow equivalence, data-flow relationships or functional behavior; richer representations based on ASTs, CFGs, DFGs, program dependence information or learned code representations can be investigated. Second, the analysis is based on BigCloneBench, which contains only Java code, and should be repeated on other datasets, programming languages and real-world software repositories. Third, future studies can test whether the continuous divergence information of CC-SeDE can be used in difficulty-aware training, curriculum learning, adaptive sampling or model evaluation, and whether such strategies benefit different clone-detection architectures. Finally, validation against manually annotated or execution-based semantic equivalence information would give a stronger assessment of the relationship between the divergence parameter and actual functional similarity.

CC-SeDE thus offers a transparent, continuous framework for analyzing divergence within code-clone populations and a starting point for more semantically informed, difficulty-aware clone detection.

Data Availability Statement

The dataset used in this study is publicly available. The CodeXGLUE version of BigCloneBench can be obtained from the Hugging Face repository (https://huggingface.co/datasets/google/code_x_glue_cc_clone_detection_big_clone_bench), and the formulas used to compute token overlap, length difference ratio and CC-SeDE are given in Section 4.

Funding

This work was supported without any funding.

Conflicts of Interest

The authors declare no conflicts of interest.

AI Use Statement

The authors declare that ChatGPT (OpenAI) was used for language editing and rephrasing of selected portions of the manuscript. The authors have carefully

reviewed, revised, and verified the AI-assisted output and take full responsibility for the content of the manuscript.

Preprint Statement

A preprint version of this article is available at SSRN: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=6570859.

Ethical Approval and Consent to Participate

Not applicable. This study analyzed only a publicly available source-code benchmark and involved no human participants or animals.

References

[1] Ain, Q. U., Butt, W. H., Anwar, M. W., Azam, F., & Maqbool, B. (2019). A systematic review on code clone detection. *IEEE Access, 7*, 86121-86144. [CrossRef]

[2] Lei, M., Li, H., Li, J., Aundhkar, N., & Kim, D.-K. (2022). Deep learning application on code clone detection: A review of current knowledge. *Journal of Systems and Software, 184*, 111141. [CrossRef]

[3] Bano, Z., & Ansari, M. Z. (2024). Code clone detection using machine learning: Brief overview and latest developments. In *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)* (pp. 1-6). IEEE. [CrossRef]

[4] Kamiya, T., Kusumoto, S., & Inoue, K. (2002). CCFinder: A multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering, 28*(7), 654-670. [CrossRef]

[5] Sajnani, H., Saini, V., Svajlenko, J., Roy, C. K., & Lopes, C. V. (2016, May). SourcererCC: Scaling code clone detection to big-code. In *Proceedings of the 38th International Conference on Software Engineering* (pp. 1157-1168). [CrossRef]

[6] Roy, C. K., & Cordy, J. R. (2008, June). NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. In *2008 16th IEEE International Conference on Program Comprehension* (pp. 172-181). IEEE. [CrossRef]

[7] Jiang, L., Misherghi, G., Su, Z., & Glondu, S. (2007, May). Deckard: Scalable and accurate tree-based detection of code clones. In *29th International Conference on Software Engineering (ICSE'07)* (pp. 96-105). IEEE. [CrossRef]

[8] Zhang, Z., & Saber, T. (2025). Machine learning approaches to code similarity measurement: A systematic review. *IEEE Access, 13*, 51729-51764. [CrossRef]

[9] Alon, U., Zilberstein, M., Levy, O., & Yahav, E. (2019). code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages, 3*(POPL), 1-29. [CrossRef]

[10] Zhang, J., Wang, X., Zhang, H., Sun, H., Wang, K., & Liu, X. (2019, May). A novel neural source code representation based on abstract syntax tree. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)* (pp. 783-794). IEEE. [CrossRef]

[11] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., & Zhou, M. (2020, November). CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020* (pp. 1536-1547). [CrossRef]

[12] Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., ... & Zhou, M. (2020). Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366*. [CrossRef]

[13] Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning* (pp. 41-48). [CrossRef]

[14] Yeung, C. Y., Lee, J. S., & Tsou, B. K. (2019). Difficulty-aware distractor generation for gap-fill items. In *Proceedings of the 17th Annual Workshop of the Australasian Language Technology Association* (pp. 159-164). https://aclanthology.org/U19-1021.pdf

[15] Li, X., Liu, Z., Luo, P., Loy, C. C., & Tang, X. (2017). Not all pixels are equal: Difficulty-aware semantic segmentation via deep layer cascade. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 3193-3202).

[16] Xie, S., Feng, Z., Chen, Y., Sun, S., Ma, C., & Song, M. (2020, November). DEAL: Difficulty-aware active learning for semantic segmentation. In *Asian Conference on Computer Vision* (pp. 672-688). Cham: Springer International Publishing. [CrossRef]

[17] Zhang, Z., Han, J., Coutinho, E., & Schuller, B. (2019). Dynamic difficulty awareness training for continuous emotion prediction. *IEEE Transactions on Multimedia, 21*(5), 1289-1301. [CrossRef]

[18] Mahmoud, A., & Bradshaw, G. (2015). Estimating semantic relatedness in source code. *ACM Transactions on Software Engineering and Methodology (TOSEM), 25*(1), 1-35. [CrossRef]

[19] Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I., & Specia, L. (2017, August). SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)* (pp. 1-14). [CrossRef]

[20] Khant, K. S., Lin, H. Y., & Thongtanunam, P. (2025, April). Should Code Models Learn Pedagogically? A Preliminary Evaluation of Curriculum Learning for Real-World Software Engineering Tasks. In *2025 IEEE/ACM 22nd International Conference on Mining*

Software Repositories (*MSR*) (pp. 249-254). IEEE. [CrossRef]

[21] Svajlenko, J., Islam, J. F., Keivanloo, I., Roy, C. K., & Mia, M. M. (2014, September). Towards a big data curated benchmark of inter-project code clones. In *2014 IEEE International Conference on Software Maintenance and Evolution* (pp. 476-480). IEEE. [CrossRef]

[22] Rattan, D., Bhatia, R., & Singh, M. (2013). Software clone detection: A systematic review. *Information and Software Technology, 55*(7), 1165-1199. [CrossRef]

[23] Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., ... & Liu, S. (2021). Codexglue: A machine learning benchmark dataset for code understanding and generation. arXiv preprint arXiv:2102.04664. [CrossRef]

[24] Krinke, J., & Ragkhitwetsagul, C. (2022, October). BigCloneBench considered harmful for machine learning. In *2022 IEEE 16th International Workshop on Software Clones* (*IWSC*) (pp. 1-7). IEEE. [CrossRef]

Zaira Bano is with the Department of Computer Engineering, Jamia Millia Islamia, New Delhi, India. (Email: zaira.kabir90@gmail.com)

Mohd. Zeeshan Ansari is with the Department of Computer Engineering, Jamia Millia Islamia, New Delhi, India. (Email: mzansari@jmi.ac.in)