ZC³: Zero-Shot Cross-Language Code Clone Detection

Jia Li
Key Lab of High Confidence Software
Technology, MoE (Peking University)
Beijing, China
lijiaa@pku.edu.cn

Chongyang Tao
School of Computer Science
Peking University
Beijing, China
chongyangtao@gmail.com

Zhi Jin*
Key Lab of High Confidence Software
Technology, MoE (Peking University)
Beijing, China
zhijin@pku.edu.cn

Fang Liu
Beihang University
School of Computer Science
Beijing, China
fangliu@buaa.edu.cn

Jia Li ♂
Key Lab of High Confidence Software
Technology, MoE (Peking University)
Beijing, China
lijia@stu.pku.edu.cn

Ge Li*
Key Lab of High Confidence Software
Technology, MoE (Peking University)
Beijing, China
lige@pku.edu.cn

Abstract—Developers introduce code clones to improve programming productivity. Many existing studies have achieved impressive performance in monolingual code clone detection. However, during software development, more and more developers write semantically equivalent programs with different languages to support different platforms and help developers translate projects from one language to another. Considering that collecting cross-language parallel data, especially for low-resource languages, is expensive and time-consuming, how designing an effective cross-language model that does not rely on any parallel data is a significant problem. In this paper, we propose a novel method named ZC³ for <u>Z</u>ero-shot <u>C</u>ross-language <u>C</u>ode <u>C</u>lone detection. ZC³ designs the contrastive snippet prediction to form an isomorphic representation space among different programming languages. Based on this, ZC³ exploits domain-aware learning and cycle consistency learning to further constrain the model to generate representations that are aligned among different languages meanwhile are diacritical for different types of clones. To evaluate our approach, we conduct extensive experiments on four representative cross-language clone detection datasets. Experimental results show that ZC³ outperforms the state-of-the-art baselines by 67.12%, 51.39%, 14.85%, and 53.01% on the MAP score, respectively. We further investigate the representational distribution of different languages and discuss the effectiveness of our method.

Index Terms—code clone detection, zero-shot learning, cross-language, deep neural networks

I. INTRODUCTION

Code clones are similar or selfsame source codes that have the same functionality. Given a source code as the query, code clone detection aims to retrieve codes with the same semantics from a collection of candidates. With the increasing number of programming language types and the rapid expansion of open-source codes, developers are facing a growing need for detecting code clones across different languages. Programmers need to develop semantically equivalent but linguistically distinct programs to support diverse platforms, such as Apps in C/C# for a Windows system, Java for an Android system, and Objective-C for a Mac system [1, 2, 3]. Meanwhile, when requested to implement a specific function with an unfamiliar programming language, cross-language clone detection can help developers retrieve corresponding clones by using their proficient language programs and accomplish program migration. Besides, duplicated source codes increase the complexity of software maintenance and might introduce fault propagation if code clones change inconsistently. Therefore, the ability to automatically detect cross-language clones is significant for efficiently maintaining software systems and helps developers translate codes from one language to another by retrieving source codes with the same semantics.

Cross-language code clone detection [1, 3, 4] has been attracting increasing attention over the years. Early works detect cross-language clones by analyzing syntactic and lexical features of the source codes [3, 4, 5]. For example, LICCA [4] parses multilingual programs into unified syntax trees through the SSQSA platform [6], and measures similarities of their syntax trees; CLCD-SA [3] analyzes 9 types of syntactic features across different languages to detect clones; CLCMiner [5] mines clones from revision histories of different languages. Although training-free, most of these approaches have application limitations due to manually customized rules and service conditions. Besides, their performances are usually mediocre since they heavily depend on lexical overlap or exterior syntactic features. With the rise of deep neural networks, pre-trained language models (PLMs) trained on the multilingual corpus have achieved impressive performances on various downstream tasks [1, 7, 8] due to their strong capability in programming representation and understanding. Researchers turn to neural networks to detect cross-language clones. Some works [9] directly apply PLMs to detect cross-language clones without fine-tuning. Despite

```php
<?php
$array = array('a'=>"8", 'b'=>"3", 'c'=>"7");
asort($array);
foreach($array as $k=>$v){
    echo $k. '...' .$v. "<br>";
}

?>
```

```java
public class Main {
    public static void main(String[] args) {
        int[] a = {8, 3, 7};
        Arrays.sort(a);
        for(int i = 0; i < a.length; i ++) {
            System.out.print(a[i] + " ");
        }
    }
}
```

```python
def sortlist(a):
    a.sort(key=lambda x:x[0],reverse=False)
    print(a)

    return a
```

(a) PHP (b) Java (c) Python

Fig. 1. Exhibition of domain divergence in PHP, Java, and Python languages.

mapping multilingual representations into the same space, these works ignore aligning the semantically similar representations of different languages, thus achieving suboptimal performances. To further align multilingual representations, some approaches [1] utilize a large amount of cross-language parallel clone data to fine-tune pre-trained models. However, in the software system, there are insufficiently labeled cross-language clones, especially for low-resource languages, and the costs of data collection, cleaning, and modeling parallel corpora are all non-neglectable. Therefore, how to design an effectively unsupervised cross-language code clone model without any parallel multilingual clone pairs is a meaningful problem.

Compared with supervised monolingual or multilingual clone detection, zero-shot cross-language clone detection is more challenging. The main difficulties are as follows. First, *the characteristics of various programming languages are discrepant.* Each language exists numerous language-specific lexical contexts, syntax, and API terms. Figure 1 reports the domain divergence between PHP, Java, and Python languages. As shown in Figure 1, PHP language uses "echo" and Python language utilizes "print" for outputting strings, respectively. "Array.sort()", "asort()", and ".sort()" are used for sorting arrays and have distinguishing usage patterns in different languages. If we directly utilize models to detect multilingual clones, they will achieve suboptimal performances due to recognizing these discrepancies is difficult. Second, *parallel corpus is not available for fine-tuning.* Though existing pre-trained models [1] have achieved impressive performance on clone detection, most of them rely heavily on a large amount of parallel data to detect clones. Collecting high-quality labeled cross-language datasets is expensive and time-consuming, especially for low-resource languages. In the zero-shot cross-language clone detection scenario, there is no labeled multilingual parallel data for models to learn the semantic similarity among different languages. Third, *how to detect code clones.* Due to the diversity of the code implementations, functionally equivalent programs usually have dissimilar implementations. In addition, some non-equivalent programs look similar in lexicon and syntax. It is difficult for models to detect functionally equivalent programs with dissimilar implementations or distinguish the non-equivalent programs that have similar implementations.

To address the aforementioned challenges, we propose a novel approach dubbed ZC^3 to investigate zero-shot cross-language code clone detection. ZC^3 is based on the large language model CodeBERT [8] which is pre-trained on six

programming languages, thus various languages can be represented in the same vector space. Next, we aim to align the representations among different languages in high-dimensional space without any labeled cross-language parallel corpus. We propose contrastive snippet prediction to construct an isomorphic representation space among diverse languages. Based on the isomorphic structure, we exploit domain-aware learning and cycle consistency learning to empower the model to generate representations that are aligned among different languages, meanwhile, are diacritical among diverse functions as much as possible.

To assess the effectiveness of our ZC^3, we conduct extensive experiments. (1) We evaluate ZC^3 on four representative cross-language clone detection datasets, including the CSN_{CC} [9], CodeJam [3], AtCoder [3], and XLCOST [10] datasets. (2) In terms of MAP, ZC^3 outperforms the state-of-the-art (SOTA) baselines by up to 46.59% absolute improvements. Concretely, compared to the SOTA methods, ZC^3 improve the performances from 13.37% to 66.38%, 24.83% to 91.96%, 23.85% to 75.24%, and 82.09% to 96.94% on the four datasets, respectively. (3) We also evaluate ZC^3 to detect clones on the monolingual languages and on the unseen programming languages. ZC^3 achieves impressive performance on unseen programming languages and also performs well in detecting monolingual clones. (4) We provide the ablation results by gradually removing each part. The results demonstrate the contributions of each module. (5) We visualize the representations of source codes. The representational distributions are well-aligned among different languages, meanwhile, are diacritical for different types of clones. The code and data for this paper can be found at https://github.com/lairikeqiA/ZC3.

We summarize our contributions in this paper as follows:

- Our paper investigates zero-shot cross-language clone detection where no multilingual parallel corpus is required.
- We propose a novel zero-shot cross-language clone detection method dubbed ZC^3. It designs the contrastive snippet prediction to form an isomorphic representation structure. Based on the isomorphic space, we exploit domain-aware learning and cycle consistency learning to constrain the model to generate aligned and diacritical representations among various languages.
- We conduct extensive experiments on four cross-language datasets. Experimental results reveal that ZC^3 significantly outperforms the state-of-the-art baselines.

II. BACKGROUND

A. Code Clone Detection

Given a program as the query, we target to retrieve codes with the same semantics from a collection of candidates. In this paper, we do not consider the classifying scenario that classifies whether two codes are semantically equivalent or not, since the retrieval task is more practical in software systems.

According to Roy and Cordy [11], similar codes can be divided into two kinds. They are similar in the program text, or semantically equivalent without being textually similar. Based on the two kinds, code clones are summarized in four types, including Type I, Type II, Type III, and Type IV. Type I: Identical source codes except for variations in white space (or layout) and comments. Type II: Code fragments have identical syntax but can be different in identifiers, literals, types, layouts, and comments. Type III: Fragments have more discrepancy except for variations in identifiers, literals, types, layout, and comments, where statements can be removed, added, and changed. Type IV: Programs have the same semantics but can be implemented through different methods, named semantic clones. In terms of their definitions, Type I-III are relevant to textual similarity and Type IV is about functionally equivalent.

Various traditional methods [12, 13, 14] tried to analyze and detect Type I-III. Sajnani et al. [12] proposed SourcererCC, which compares tokens and subsequences to detect clones. Jiang et al. [14] consider only syntactic information and ignore lexical similarities of programs. These approaches have limited success in detecting semantic clones and considering functional behaviors. To detect semantic clones (Type IV), researchers applied deep learning approaches [15, 16, 17, 18]. These methods use deep neural networks to acquire code representations, then detect clones through their representations' similarities. Among these works, however, most of them can only detect clones in the monolingual scenario. In this paper, the zero-shot cross-language code clone task belongs to Type IV, which is difficult to detect clones since source codes have different implementations and further obey specific syntactic rules among diverse programming languages.

B. Zero-Shot Learning

Though having achieved satisfactory performances on various downstream tasks, deep learning methods [7, 19, 20, 21, 22], especially pre-trained language models, are data-hungry. The performance of these models is strongly dependent on a large amount of parallel corpus. Models usually achieve suboptimal performance without sufficient examples. In practice, parallel data is rare and collecting them is time-consuming. To address the above issue, researchers [23] begin to investigate to zero-shot learning. It is a machine-learning technology that aims to adapt a model to a new task or a new domain without any labeled parallel examples.

In the software system, some PLMs [24, 25, 26, 27, 28] utilize large-scale networks and a large amount of data to learn satisfactorily semantic representations for programs. Then, they are directly applied to detect clones based on the representational space without fine-tuning. Although performing
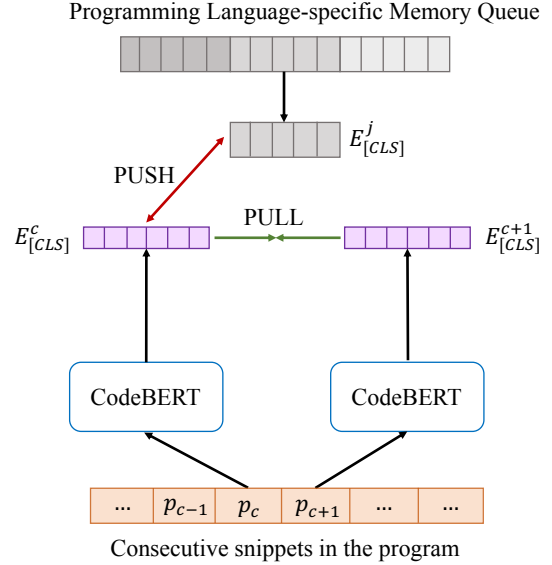

Fig. 2. Schematic depiction of the contrastive snippet prediction (CSP).

in the zero-shot setting, most of these models usually achieve poor performance or can only detect clones in the same programming language. Compared with these models, ZC^3 faces more challenges since it needs to align semantically similar representations among different languages without any annotated cross-language parallel corpus.

III. METHODOLOGY: ZC^3

ZC^3 aims to align representations of different languages on the high-dimensional space as much as possible, then detect cross-language clones based on the aligned vector space. Our proposed ZC^3 designs the contrastive snippet prediction to construct an isomorphic representation structure for different languages. Based on the isomorphic space, domain-aware learning and cycle consistency learning are exploited to empower the model to generate well-aligned representations among different languages that meanwhile are diacritical for different types of clones. In this section, we first briefly formulate code clone detection based on PLMs and then describe our proposed ZC^3 method.

A. Preliminary

Given a specific program as the query, we focus on retrieving semantically equivalent codes from a candidate pool, instead of classifying whether two programs are clones or not, since the retrieval scenario is more realistic in software systems. Thus, zero-shot cross-language code clone detection is defined as follows. It learns a robust matching model without any cross-language parallel data and retrieves the matched codes from target languages given a source-language program.

Recently, pre-trained programming language models, such as CodeBERT [8] and GraphCodeBERT [29], have been widely used in many code-related tasks and shown impressive results. In this paper, we also consider building our model based on PLMs and select CodeBERT [8] as the basic model since it is pre-trained on a large amount of unlabeled multilingual data and can map representations of different

languages to a unified space. Given a source-language program as the query s, we first concatenate $s = \{s_1, s_2, ..., s_n\}$ with special tokens $[CLS]$ and $[SEP]$, which can be formulated as $\{[CLS], s_1, s_2, ..., s_n, [SEP]\}$. It is tokenized into a sequence of tokens and fed into the CodeBERT. Then CodeBERT outputs a semantically embedding sequence $S = \{E^s_{[CLS]}, E^s_2, ..., E^s_{|S|}\}$ where $|S|$ is the length of tokens. Meanwhile, we use the same way to process target-language fragments t and acquire its representational sequence $T = \{E^t_{[CLS]}, E^t_2, ..., E^t_{|T|}\}$. In this paper, we treat $E^s_{[CLS]}$ and $E^t_{[CLS]}$ as the aggregated representations of s and t. Finally, the semantic similarity of the two programs is formulated as:

$$sim(s,t) = cos(E^s_{[CLS]}, E^t_{[CLS]}) = \frac{E^s_{[CLS]}{}^T E^t_{[CLS]}}{\|E^s_{[CLS]}\|\|E^t_{[CLS]}\|} \quad (1)$$

where $sim(s,t)$ denotes the cosine similarity between the query s and the candidate code t.

B. Contrastive Snippet Prediction

Cross-language dense retrieval tasks require the model to map semantically related sequences to similar positions in representational space, but existing approaches only focus on mapping multilingual sequence pairs with the same meaning to similar embeddings [30]. To mitigate this phenomenon, we propose a new pre-training task named Contrastive Snippet Prediction (CSP). The CSP task targets to construct isomorphic representation space among different languages by modeling the snippet relations in programs. In the isomorphic space, the code snippets with the same semantics in different languages have similar relationships to other monolingual code snippets. The CSP task is illustrated in Figure2.

CSP Objective. CSP task aims to establish the relations of code snippets. Formally, a functional program is a sequence of code snippets, where each snippet contains several consecutive lines. For each center code snippet, we define the snippets in the window centered on it as functional snippets. Given a center code snippet p_c, the CSP task needs to select the correct functional snippet p_f from a randomly sampled snippet pool. With the CSP task, the model could assess the mutual information $I(p_f|p_c)$ of snippet relations.

Specifically, a program consists of a series of code snippets $(p_1, p_2, ..., p_n)$. For each center snippet p_c, its functional snippet set is $F(p_c) = \{p_f|c-w <= f <= c+w, f \neq c\}$. w is the radius of the window that represents the maximum snippet distance between the center snippet p_c and its functional snippets p_f. Next, CodeBERT is applied to encode p_c and p_f, and acquires $E^c_{[CLS]}$ and $E^f_{[CLS]}$ as their representations. We then model the relations of center snippet p_c and other snippets with the contrastive loss. Following SimCLR [31], the learning objective $\mathcal{L}_{CSP}$ of the CSP task is formulated as:

$$\mathcal{L}_{CSP} = -\left[log\frac{e^{s(E^c_{[CLS]}, E^f_{[CLS]})/\tau}}{\sum_{E^n_{[CLS]}\in\mathbb{N}_{CSP}} e^{s(E^c_{[CLS]}, E^n_{[CLS]})/\tau}}\right] \quad (2)$$
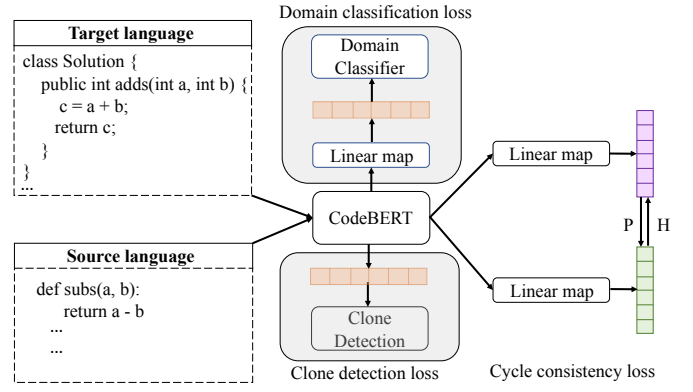

Fig. 3. The overview of adversarial clone detection.

where τ is a temperature parameter. $s(\cdot)$ calculates the cosine similarity of two vectors. $\mathbb{N}_{CSP}$ represents the randomly selected snippets out of the functional snippet set $F(p_c)$.

Based on the CSP task, our model constructs the isomorphic representation among different languages by learning the snippet relations. The learned representational space for different languages will have good cross-language properties after it became isomorphic by the task.

Programming Language-specific Queue. As demonstrated by [32], the bound of contrastive loss descends as the number of negative samples increases. We design a language-specific queue to expand the negative sample number under the limited batch size. In particular, the language-specific queue only stores the embedding of snippets within the same programming language. In the training process, given a language-specific query, negative samples are randomly selected only from the candidate queues that have the same language as the query. The queue is maintained in FIFO (First-In-First-Out) manner. The newest mini-batch embedding is put into the queue, meanwhile, the oldest mini-batch representations are taken out. With each iteration, language representations and parameters of the model are optimized.

C. Adversarial Clone Detection

After acquiring isomorphic distributions, in order to further constrain the model to generate aligned representations, we explore domain-aware learning and cycle consistency learning at the function level. Based on this, we introduce a clone detection task to teach the model for detecting clones. Finally, the representational distributions of ZC³ are well-aligned among different languages, meanwhile, are diacritical for different types of clones. Figure 3 illustrates the overview of the adversarial clone detection.

1) Domain-Aware Learning: To further align multilingual representations, we introduce the domain-aware learning module. In the procedure, we add a domain classifier layer on the encoder to predict the language labels of programs, and the parameters of the encoder are optimized to maximize the domain classifier's loss by adversarial training. Formally, we feed a source-language program or a target-language code into CodeBERT and obtain its aggregated representation $E_{[CLS]}$.
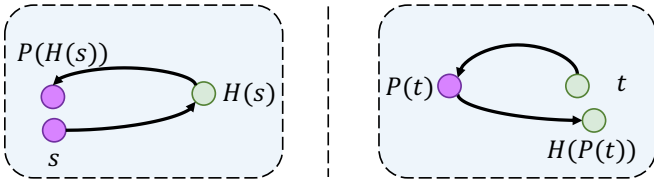
Fig. 4. The demonstration of cycle consistency learning that constrains that if the model translates a program from one language to the other and back again the program arrives where it starts: $s \rightarrow H(s) \rightarrow P(H(s)) \approx s$, and $t \rightarrow P(t) \rightarrow H(P(t)) \approx t$.

Then $E_{[CLS]}$ is fed into a gradient reversal layer (GRL) [33] to reduce the domain discrepancy of $E_{[CLS]}$ for the source domain and target domain. During the forward propagation, GRL acts as an identity function but during the backpropagation, GRL multiplies the incoming gradient by a negative factor $-\lambda$ which reverses the gradient direction. We can formulate GRL as a "pseudo-function" $G_\lambda(x)$ by the following two equations:

$$G_\lambda(x) = x \qquad (3)$$

$$\frac{\partial G_\lambda(x)}{\partial x} = -\lambda I \qquad (4)$$

where $\lambda = \frac{2}{1+exp(-\mu \frac{t}{T})}$ is a constant. T is the maximum optimizing step, t is the current training step, and μ is a hyper-parameter.

Then we feed GRL's outputs into the domain classifier layer which is a non-linear transformation layer parameterized as:

$$f_{dc}(x) = \text{softmax}(W_{dc}G_\lambda(E_{[CLS]}) + b_{dc}) \qquad (5)$$

The target is to minimize the cross-entropy for all data from both the source language and the target language:

$$\mathcal{L}_{dc} = -\frac{1}{N_s + N_t} \sum_{i=1}^{N_s+N_t} y_{dc}^i log f_{dc}(x^i) \\ +(1 - y_{dc}^i)log(1 - f_{dc}(x^i)) \qquad (6)$$

where $y_{dc}^i \in \{0, 1\}$ is the ground truth language label. N_s and N_t are the numbers of training examples in both languages.

Learning with GRL is adversarial such that the domain classifier is optimized to increase the ability to distinguish different languages, while the encoder learns representations to reduce the domain classification accuracy due to the reversal of the gradient. Through domain-aware learning, the multilingual representations are indistinguishable and further aligned.

2) Cycle Consistency Learning: CSP task targets learning snippet relations and constructing the isomorphic structure in the embedding space. At the function-individual level, however, the model is not constrained to generate representations that are both aligned and diacritical. Therefore, cycle consistency learning [34] is designed to restrict the model to produce aligned and discriminative embeddings at the function level among different languages. The cycle consistency contains two parts, namely forward cycle consistency and back cycle consistency. For a source-language sample, the cycle can be able to bring it back to the original semantic function in

the source language as much as possible, which is called forward cycle consistency. Similarly, we utilize the backward cycle consistency to map a target-language code back into its original meaning. The forward cycle consistency can be formulated as:

$$F_s : s \rightarrow H(s) \rightarrow P(H(s)) \approx s \qquad (7)$$

The backward cycle consistency is represented as:

$$B_t : t \rightarrow P(t) \rightarrow H(P(t)) \approx t \qquad (8)$$

The $H(\cdot)$ maps representations from the source language to the target language, and another linear operation $P(\cdot)$ maps representations in the reverse direction. $H(\cdot)$ and $P(\cdot)$ are both trainable linear transformations, which have the same structure but share no parameters. In the case of the $P(\cdot)$ module, it can be formulated as following:

$$P(x) = w_2(w_1x + b_1) + b_2 \qquad (9)$$

where w_k and b_k are training parameters.

We incentivize this behavior using a cycle consistency loss:

$$\mathcal{L}_{cyc} = \mathbb{E}_{s \in S}[|P(H(s)), s|] + \mathbb{E}_{t \in T}[|H(P(t)), t|] \qquad (10)$$

where $|\cdot|$ is the L1 loss. S and T are the samples of the source language and target language, respectively.

3) Code Clone Detection: Based on the aligned vector space, we use the labeled monolingual data to learn effective clone ability. The task requires the model to identify semantically equivalent programs by measuring the matching degree between the code query and the candidate code. Precisely, we feed the code query and a candidate snippet into the encoder, respectively, and obtain their aggregated representations $E_{[CLS]}^q$ and $E_{[CLS]}^r$. Then their matching degrees can be calculated by cosine similarity as formulated in Formula 1. Finally, the cross-entropy loss function is utilized as the training objective for the task:

$$\mathcal{L}_{cl} = -\frac{1}{N_m} \sum_{i=1}^{N_m} y^i log(sim(E_{[CLS]}^{q,i}, E_{[CLS]}^{r,i})) \qquad (11)$$

where N_m is the number of labeled monolingual examples.

In total, the adversarial code clone procedure is optimized by minimizing the following loss:

$$\mathcal{L} = \mathcal{L}_{cl} + \alpha \mathcal{L}_{dc} + \beta \mathcal{L}_{cyc} \qquad (12)$$

where α denotes the weight for domain classifier loss and β is the weight for the cycle consistency loss.

IV. STUDY DESIGN

To evaluate the effectiveness of our approach, we conduct a large-scale study to answer three research questions. In this section, we describe the details of our study, including datasets, metrics, and baselines.

	Snippet-level			Program-level		
	Python	Java	Avg	Python	Java	Avg
#num	81207	91089	86148	9263	9623	9443
#lines	2.41	3.71	3.06	20.54	34.93	27.74
#tokens	21.63	24.1	22.87	188.5	227.1	207.8

A. Research Questions

Our study aims to answer three research questions (RQ). In RQ1, we compare our ZC^3 to advanced code clone baselines on four representative cross-language datasets. In RQ2, we evaluate how ZC^3 performs in the unseen programming languages. In RQ3, we verify the performance of ZC^3 on the monolingual code clone detection scenario.

RQ1: How does ZC^3 perform compared to the SOTA unsupervised cross-language code clone baselines? We evaluate ZC^3 on four representative datasets. Considering that our approach aims to detect clones in the unsupervised setting, we compare our ZC^3 with the advanced unsupervised cross-language clone detection baselines.

RQ2: What is the performance of ZC^3 in the unseen programming languages? In this paper, we also assess whether ZC^3 can effectively detect cross-language clones in the unseen languages.

RQ3: How does ZC^3 perform on the monolingual code clone detection? Our approach is designed for zero-shot cross-language clone detection and aligns representational distributions for different languages. In this RQ, we evaluate whether ZC^3 performs well on the monolingual clone detection.

B. Datasets and Metrics

To construct isomorphic embedding space, we use the training set of XLCoST dataset [10] for contrastive snippet prediction. XLCoST is collected from GeeksForGeeks [1], which is a website containing many programming solutions for data structures and algorithm problems. XLCoST contains 7 different programming languages including C++, Java, Python, C#, Javascript, PHP, and C. Each program is divided into several consecutive code snippets, where each snippet describes a complete semantic meaning such as if-statement and for-statement. The statistics of XLCoST are given in Table I.

We use the dataset released by Guo et al. [9] for adversarial code clone in Section III-C. We name it CSN_{CC} in this paper. CSN_{CC} is collected from the CodeSearchNet corpus [35] and contains 11,744, 15,594, and 23,530 function-level programs in Ruby, Python, and Java languages, respectively. Each program can solve one of 4,053 problems. We utilize the first 3,500 problems for adversarial code clone and randomly select other 500 problems to evaluate ZC^3.

We evaluate our model on four cross-language code clone datasets including the Google CodeJam dataset [3], AtCoder

Multilingual	Java			Python		
	Train	Valid	Test	Train	Valid	Test
CodeJam	3461	390	433	2846	258	372
AtCoder	11824	1642	1408	11631	1337	1365
CSN_{CC}	15478	4952	3100	8081	3229	3197
XLCoST	9623	472	887	9263	472	887

Monolingual	C			Java		
	Train	Valid	Test	Train	Valid	Test
BigCloneBench	–	–	–	7,310	912	623
POJ-104	32000	8000	12000	–	–	–

dataset [3], XLCoST corpus [10], and CSN_{CC} dataset [9]. CodeJam comes from Google's programming competition [2]. There are 265 problems and each problem has several solutions with Python or Java languages. These problems are divided into train, valid, test sets in the ratio of 8:1:1. We only use the test set to evaluate ZC^3. The second dataset is the AtCoder corpus, which is collected from the AtCoder programming website [3] in Japan. There are 1,115 classes of problems in Python and Java. The test set has 115 problems with 1,365 programs. The third dataset is the XLCoST dataset [10]. Different from the above two datasets, there is only one solution for each problem in peer language. It contains 11,028, and 10,622 programs for Java and Python. We use the test set of CSN_{CC} [9] as the fourth dataset. There are 500 problems and all of them come from the public GitHub repositories [4].

To verify the monolingual clone ability, we further introduce two monolingual clone detection datasets, i.e., the BigCloneBench dataset [36] and the POJ-104 dataset [37]. BigCloneBench was first proposed by Svajlenko et al. [38] and we use the filtered version provided by Wang et al. [36]. It has 9,134 codes in total and 912 programs in the test set. We group the test programs according to their semantic functions and remove categories that only contain one program. Finally, 623 programs are remained and divided into 8 functional classes. POJ-104 is collected from a pedagogical programming open judge (OJ) system that automatically judges the correctness of the submitted source code for specific problems. It contains 104 problems and each problem has 500 student-written C programs. Following Lu et al. [39], we divide these problems into 64, 16, and 24 classes for training, validation, and testing.

Following previous works [9, 39], we use MAP as the metric. MAP is the mean of average precision scores, which is evaluated for retrieving similar samples given a query.

C. Compared Models

Considering that this paper investigates zero-shot cross-language clone detection, we compare ZC^3 with the unsupervised cross-language clone detection baselines as follows:

---

[1] https://www.geeksforgeeks.org/

[2] http://code.google.com/codejam
[3] https://atcoder.jp/
[4] https://github.com/

TABLE III
EVALUATION RESULTS OF ZERO-SHOT CROSS-LANGUAGE CLONE DETECTION ON THE PYTHON → JAVA SETTING AND THE JAVA → PYTHON SETTING OF THE ATCODER, CODEJAM, AND XLCOST DATASETS.

	AtCoder		CodeJam		XLCoST	
	Python→Java	Java→Python	Python→Java	Java→Python	Python→Java	Java→Python
CodeBERT	3.81	2.98	8.33	8.84	3.36	1.98
GraphCodeBERT	10.23	6.05	14.95	15.10	35.32	23.19
UniXcoder	28.81	20.86	24.42	23.28	84.25	79.93
CodeBERT$_{\text{MCC}}$	89.36	84.72	63.45	68.66	92.87	92.63
CodeBERT$_{\text{CSP}}$	89.84	85.56	67.17	69.83	93.62	94.05
CodeBERT$_{\text{DAL}}$	91.89	87.28	69.30	72.74	95.36	96.11
ZC^3	**92.25 (↑ 3.23%)**	**91.67 (↑ 8.21%)**	**73.92 (↑ 16.50%)**	**76.57 (↑ 11.52%)**	**96.96 (↑ 4.40%)**	**96.92 (↑ 4.63%)**

TABLE IV
EVALUATION RESULTS OF ZERO-SHOT CROSS-LANGUAGE CODE CLONE DETECTION ON THE X → Y SETTINGS OF CSN$_{\text{CC}}$ DATASET. THE X AND Y REPRESENT RUBY, JAVA, OR PYTHON LANGUAGES.

	Java		Python		Ruby		Avg
	Python	Ruby	Java	Ruby	Python	Java	
CodeBERT	0.83	0.96	1.23	1.30	1.47	1.03	1.14
GraphCodeBERT	2.01	2.04	4.37	4.78	5.03	4.54	3.79
UniXcoder	8.49	7.83	13.41	18.26	20.34	11.90	13.37
CodeBERT$_{\text{MCC}}$	54.26	50.97	58.13	67.04	65.52	55.78	58.62
CodeBERT$_{\text{CSP}}$	55.70	52.42	59.46	67.81	67.36	56.47	59.87
CodeBERT$_{\text{DAL}}$	57.92	55.12	61.54	69.60	67.85	58.22	61.71
ZC^3	**63.69**	**63.28**	**64.05**	**72.18**	**72.32**	**62.76**	**66.38**
	(↑ 17.38%)	**(↑ 24.15%)**	**(↑ 10.18%)**	**(↑ 7.67%)**	**(↑ 10.38%)**	**(↑ 12.51%)**	**(↑ 13.24%)**

- **CodeBERT** [8]: The model is an encoder-only structure, which is pre-trained in six programming languages with various advanced pre-training tasks.
- **GraphCodeBERT** [29]: The model considers the relation of various variables and proposes a graph-guided masked attention to incorporate structure information of codes.
- **UniXcoder** [9]: It is a unified pre-trained model that uses mask attention matrices to control behaviors. Besides, the model utilizes multi-modal contrastive learning and cross-modal generation tasks to align representations among different languages and provides SOTA results.
- **CodeBERT$_{\text{MCC}}$**: Fine-tuning CodeBERT on the monolingual code clone pairs in Python and Java languages.
- **CodeBERT$_{\text{CSP}}$**: Post-training CodeBERT with the contrastive snippet prediction task as described in Section III-B. Then it is continually fine-tuned on the monolingual clone pairs in the same way with CodeBERT$_{\text{MCC}}$.
- **CodeBERT$_{\text{DAL}}$**: The model executes the adversarial domain-aware learning based on the CodeBERT$_{\text{CSP}}$. The model can be regarded as an ablation model where the cycle consistency learning is removed from our ZC^3.

As described in Section VII-B, we notice some zero-shot cross-language methods [4, 5] utilize manually customized rules to detect cross-language clones at Type I-III. We think these studies have a different research focus from this paper (Type IV). Besides, previous works [1] have demonstrated that PLMs outperform them by a large margin. In this paper, we do not directly compare ZC^3 to these studies and leave them

for future work.

D. Implementation Details

We use CodeBERT-base [8] to initialize our model. It is an encoder-only backbone having a 12-layer Transformer with 12 attention heads, 64 head sizes, and 768 hidden sizes. For contrastive snippet prediction, we set the window size to 5, in other words, the largest distance between the snippet center and the functional snippet is 2. The size of the language-specific queue is 128. During adversarial clone detection, μ is set as 0.01. The number of source-language samples and target-language samples is a ratio of 1:2. α and β are set to 1, respectively. The batch size is 32. The maximum length of the source and target programs are both 512. The optimizer is an AdamW optimizer with an initial learning rate 2e-5.

V. RESULTS AND ANALYSES

In our first research question, we evaluate the performance of our ZC^3 with respect to advanced code clone approaches.

RQ1: How does ZC^3 perform compared to the SOTA unsupervised cross-language code clone baselines?

Setup. We evaluate baselines (Section IV-C) and our ZC^3 on four cross-language datasets (Section IV-B). The evaluation metric is described in Section IV-B, i.e., the MAP score. For the metric, higher scores represent better performance.

Results. Table III reports the performances of all methods on the AtCoder, CodeJam, and XLCoST datasets. Table IV shows the results on CSN$_{\text{CC}}$. The percentages in parentheses are the relative improvements compared to CodeBERT$_{\text{MCC}}$.

	Python					Java				
	C++	C#	JavaScript	PHP	C	C++	C#	JavaScript	PHP	C
CodeBERT	2.95	3.48	2.73	4.25	1.79	3.67	3.09	2.44	5.24	15.93
CodeBERT$_{\text{MCC}}$	91.37	93.18	88.92	92.36	94.49	93.78	97.12	84.02	94.92	96.59
ZC^3	**94.61**	**96.65**	**95.35**	**95.92**	**95.88**	**95.81**	**98.82**	**95.67**	**97.70**	**98.83**
	(↑ **3.55%**)	(↑ **3.72%**)	(↑ **6.74%**)	(↑ **3.85%**)	(↑ **1.47%**)	(↑ **2.16%**)	(↑ **1.75%**)	(↑ **12.68%**)	(↑ **3.98%**)	(↑ **2.32%**)

Analyses. (1) ZC^3 achieves the best performance among all baselines with significant improvements. Concretely, compared with the SOTA baseline, UniXcoder, ZC^3 improves the MAP score from 24.83% to 91.96%, 23.85% to 75.24%, and 82.09% to 96.94% on the three datasets, respectively. (2) Our approach effectively aligns representational distributions among different languages. Despite significant improvements of training on the monolingual labeled corpus, the gap between CodeBERT$_{\text{MCC}}$ and ZC^3 is still up to 6.1% on average of the three datasets. That demonstrates our designed contrastive snippet learning, domain-adversarial learning, and cycle consistency learning accelerate to construct a well-aligned vector space. (3) Constructing an isomorphic representation distribution is beneficial to multilingual clone detection. By introducing contrastive snippet learning, CodeBERT$_{\text{CSP}}$ further improves CodeBERT$_{\text{MCC}}$ on the MAP score. The CSP task effectively grasps the snippet relations in different languages. (4) Domain-aware learning and cycle consistency learning improves performance significantly. Compared to CodeBERT$_{\text{CSP}}$, ZC^3 acquires 4.70% absolute improvements on MAP and achieves the best performance. That verifies that constraining the model to generate aligned and diacritical representations at the function level is essential and effective for cross-language clone detection.

To illustrate the generalization ability of our ZC^3, we further conduct experiments on CSN$_{\text{CC}}$ dataset that is collected from GitHub. From Table IV, we can find that ZC^3 gets 53.01% average absolute improvements on the six settings in terms of MAP, and the trends are generally consistent with the results in Table III, which further demonstrates the effectiveness and generalization ability of our method. In addition, although the CSP task is only trained in Java and Python languages, the procedure can also improve the performance on the Java → Ruby setting and the Python → Ruby setting.

> **Answer to RQ1:** ZC^3 achieves the best results among all baselines. In particular, ZC^3 acquires 67.13%, 51.30%, 14.85%, and 53.01% absolute improvements on four datasets at MAP. The significant improvements prove our approach can produce well-aligned representational distributions and is a promising approach for zero-shot cross-language code clone detection.

RQ2: What is the performance of ZC^3 in the unseen programming languages?

	POJ-104	BigCloneBench
CodeBERT$_{\text{FT}}$	86.42	65.37
ZC^3	86.27	62.59

Setup. We evaluate how ZC^3 performs in the unseen languages. Precisely, we use the model trained on Python and Java as described in Section III to evaluate the Python → X and Java → X settings, where X ∈ {C++, C#, JavaScript, PHP, C} is never seen during the training procedure.

Results. The results are shown in Table V. The percentages in parentheses mean the relative improvements from CodeBERT$_{\text{MCC}}$ to ZC^3.

Analyses. (1) Our ZC^3 can work well in the unseen programming languages. As shown from results, ZC^3 achieves 95.68% on the Python → X setting and 97.37% on the Java → X setting in terms of MAP. Besides, ZC^3 outperforms CodeBERT$_{\text{MCC}}$ by a large margin in all settings. It brings 3.87% and 4.58% relative improvements on the two settings compared with CodeBERT$_{\text{MCC}}$. That demonstrates that the aligned representation space constructed by contrastive snippet prediction and adversarial clone detection is universal and can be translated into other unseen languages. (2) The improvements on the Java → X setting is higher than the Python → X setting. The reason might be that Java, C++, C#, JavaScript, and PHP are object-oriented programming languages, and their syntax and code morphology are similar.

> **Answer to RQ2:** Our ZC^3 achieves impressive performances in unseen programming languages. That demonstrates that our approach works well and can be applied to other unseen languages.

RQ3: How does ZC^3 perform on the monolingual code clone detection?

Setup. We assess the performance of our ZC^3 on the monolingual clone detection scenario. For a fair comparison, we train CodeBERT on the training set of POJ-104 and BigCloneBench, respectively, and name it CodeBERT$_{\text{FT}}$.

Results. We present experimental results of CodeBERT$_{\text{FT}}$ and ZC^3 in Table VI.

Analyses. (1) Our ZC^3 performs well on the monolingual code clone detection. In particular, ZC^3 achieves 86.27%
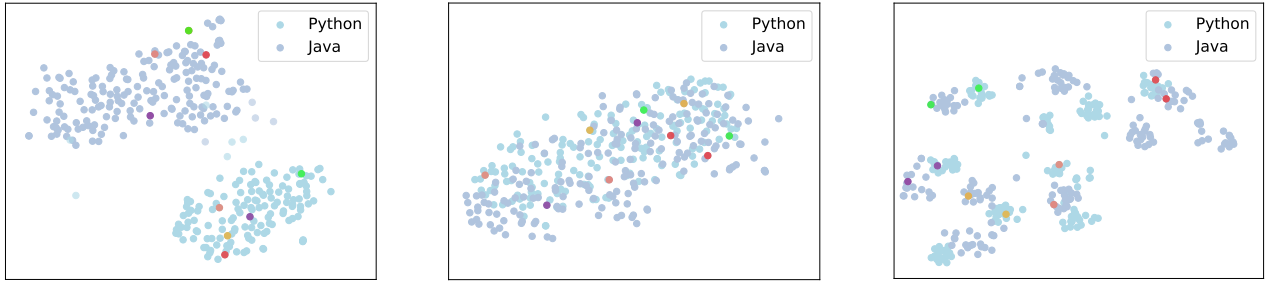
Fig. 5. Comparision of t-SNE visualization in CodeBERT (left), CodeBERT$_{DAL}$ (middle), and ZC^3 (right). Each data point in the figure is an example. Data points with the same color represent a clone pair. Different colors mean different types of clones.
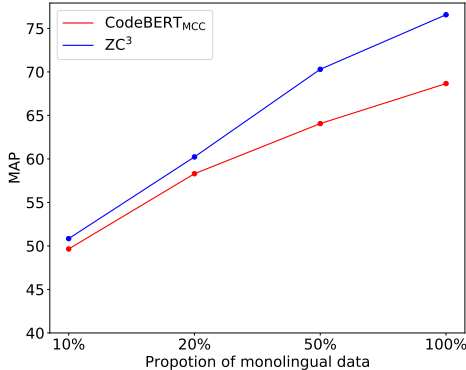

Fig. 6. Effects of the number of monolingual labeled clone pairs about CodeBERT$_{MCC}$ and ZC^3 on the Python $\rightarrow$ Java setting of CodeJam dataset.

and 62.59% on MAP at the POJ-104 dataset and the Big-CloneBench dataset, respectively. This means that our approach not only achieves SOTA results on zero-shot cross-language clone detection but also performs well in detecting monolingual clones. (2) The performances of ZC^3 are competitive to CodeBERT$_{FT}$. CodeBERT$_{FT}$ is trained on the training set of the two datasets and then detects clones on the test set, which is a supervised procedure and can be treated as the upper bound. Compared to CodeBERT$_{FT}$, the results of our ZC^3 are only slightly lower, which demonstrates ZC^3 is effective enough in the monolingual clone detection scenario. In addition, considering that ZC^3 is also trained on the monolingual clone corpus, we also explore why ZC^3 is lower than CodeBERT$_{FT}$. The reason might be that our labeled example size is smaller than the counterpart of CodeBERT$_{FT}$ and the representational alignment losses affect the model to focus on learning Java clone detection ability.

> **Answer to RQ3: ZC^3 performs well on the monolingual clone detection scenario, and its performance is close to the counterpart of the supervised method, e.g., CodeBERT$_{FT}$. In particular, ZC^3 acquires 86.27% and 62.59% at MAP on both datasets.**

VI. Discussion

A. Impacts of the Number of Monolingual Clone Pairs

We further study the impacts of the number of labeled clone samples in the monolingual language. Figure 6 reports how the performances of ZC^3 and CodeBERT$_{MCC}$ change with respect to different proportions of labeled monolingual clone samples on the CodeJam dataset. We can find that the performances of both models increase monotonically with the increase of monolingual training samples. In addition, ZC^3 always outperforms CodeBERT$_{MCC}$ in all cases, and the improvements become more significant with the increase of labeled training samples.

B. Visualization of Features

We further perform the visualization of example representation vectors on CodeBERT, CodeBERT$_{DAL}$, and ZC^3. We randomly select ten classes of Python-Java clone pairs from the test set of the CodeJam dataset. Then we feed them into models and obtain their representations. The dimension of these representations are reduced from 768 to 2 by t-SNE [40], and each sample is mapped into a point at Figure 5. From these samples, we randomly highlight five clone pairs on different classes, and the two highlighted cycles in the same color belong to a clone pair. Different colors mean different types of clones. As shown in the left subfigure, the distribution of two languages has a clear border, which demonstrates that CodeBERT is sensitive to the language type. By introducing the CSP task and domain-aware learning, the features of Python and Java languages are well entangled but the same type of clones are not aligned as shown in the middle subfigure. After cycle consistency learning, both languages not only entangle together but also each type of code clone is well clustered, which verifies that the model is constrained to generate aligned and diacritical features as much as possible.

C. Effects of Window Size and the Queue

The window size and the queue are important elements in the contrastive snippet prediction procedure. We investigate how the two elements affect the performance of our model. As shown in Table VII, with the increasing of window size and the queue size, ZC^3 acquires consistent improvements. Between the two factors, queue size has a greater influence on the performance of our model.

	Python→Java	Java→Python
Window Size		
2	73.19	76.15
3	73.37	76.24
5	73.92	76.57
Queue		
32	70.81	75.40
64	71.54	75.86
128	73.92	76.57

D. Threats to Validity

There are two main threats to the validity of our work.

The generalizability of our experimental results. To mitigate this threat, we carefully design our experimental datasets, baselines, and metrics. For the datasets, we follow previous studies [3, 9, 10] and use four representative cross-language clone detection datasets. The four datasets cover different programming languages (i.e., Java, Python, and Ruby, etc.) and come from different domains. In addition, we also introduce two popular monolingual code clone corpora, POJ-104 and BigCloneBench, to assess our approach's ability on monolingual clone detection. To verify the superiority of ZC^3, we select a series of advanced zero-shot cross-language clone detection models as our baselines for the comparison. They are the representative PLMs for understanding programs in the past three years. For the metric, following existing works [9, 39], we select a widely used MAP metric to evaluate all methods. It is defined as the mean of average precision scores and is applied to evaluate for retrieving similar samples given a query. Besides, we execute each method three times and report the average experimental results.

The implementation of models. It is widely known that deep neural models are sensitive to the implementation details, including network architectures and hyper-parameters. In this paper, we need to execute all baselines and our approach. For the baselines, we apply the source code and parameters published by their original papers [8, 9, 29]. Then, we ensure the models' performances are comparable with their reported results. For our approach, we use the mainstream neural network (e.g., CodeBERT) as our backbone as described in Section III-A. To select suitable hyper-parameters, we implement a small-range grid search on several hyper-parameters (i.e., snippet window size w, adversarial clone detection parameter μ, etc), and leave other hyper-parameters including batch size and learning rate the same with baselines. Thus, there might be room to tune network architectures and more hyper-parameters of our approach for more improvements.

VII. RELATED WORK

A. Code Clone Detection

Existing clone detection approaches can mainly be divided into traditional methods and learning-based methods. Traditional approaches aim to solve Type I-III clone detection by considering token and syntactic information. Token-aware works treat the code into a sequence of tokens and match tokens between the query and candidate code. CCFinder [41] and SourcerCC [12] convert codes into a regular form and identify clones through a token-by-token matching algorithm. They only consider textual information and rely heavily on language-related rules and lexical vocab, which limits its practical application. Syntactic clone methods use Abstract Syntax Trees (AST) to compute the similarity of codes. Deckard [14] maps the subtrees into numerical vectors in Euclidean space and clusters them. Clones are detected by comparing the whole tree. Similar to CCFinder, Deckard also needs to define specific rules for different languages.

Learning-based approaches mainly focus on Type IV clones, which detect semantically equivalent programs. The early deep neural-based clone detection method is CCLearner [42]. CCLearner trains a classifier and then utilizes it to detect clones from a candidate pool. Wei and Li [18] propose a neural framework that exploits the lexical and syntactical information for acquiring representations and computes semantical similarities between programs. Zhang et al. [15] design an AST-based model, named ASTNN, that splits the whole AST into a number of small subtrees. Then RNN encodes these subtrees and produces a vector for detecting clones. Recently, various PLMs are proposed and applied to detect clones. CodeBERT [8] is an early-designed PLMs that is pre-trained on 6 programming languages and computes the similarity of learned code representations for clone detection. Graph-CodeBERT [29] designs a graph-guided masked attention to encode the syntactic structure of codes and achieves impressive on the task. Inspired by contrastive learning [31], a series of approaches [43, 44] utilize it for learning function-level code representations. These methods augment a given sample by constructing a similar counterpart, then force models to recognize similar programs from a candidate pool. However, these methods mainly focus on detecting monolingual clones.

B. Cross-lingual Representation Alignment

Cross-lingual representation alignment aims to map semantically equivalent but linguistically different contents into similar positions in a unified vector space. Recently, multilingual representation has been studied in many natural language tasks [45, 46, 47, 48]. XLM [47] and XLM-R performs cross-lingual pre-training by introducing the translation language task on multilingual datasets.

When it comes to cross-language alignment for programs, existing models are mainly divided into two kinds. One group of models uses labeled cross-lingual clones for supervised learning. CLCDSA [3] proposes a parallel dataset that contains labeled cross-lingual clones. Then it uses the dataset to train a neural network for detecting clones. C4 [1] uses a large amount of parallel multilingual data for contrastive learning, which is a competitive method for supervised cross-language clone detection. Another school of research is unsupervised cross-language clone detection models that contain rule-based and learning-based methods. The rule-based methods contain LICCA [4] and CLCMiner [5]. LICCA [4]

depends on SSQSA's high-level representation and meanwhile requires programs to be the same length. CLCMiner [5] needs the clones from revision histories. These requirements limit their practical application. The learning-based models refer to PLMs, such as CodeBERT [8], GraphCodeBERT [29], and UniXcoder [9]. They are pre-trained in different languages and directly applied to detect multilingual clones. Despite these methods can perform zero-shot clone detection on multilingual languages, their performances are sub-optimal since they either focus on lexical similarity or ignore modeling the semantic relations among different languages.

VIII. Conclusion

During software development, there is an increasing demand for zero-shot cross-language clone detection. This paper proposes a novel model ZC^3. We design a contrastive snippet prediction task to learn snippet relations and construct an isomorphic structure for different languages. Based on the isomorphic space, we exploit domain-aware learning and cycle consistency learning to further constrain the model to generate aligned representations among various languages that meanwhile are diacritical for different types of clones. Experimental results on four cross-language clone datasets demonstrate that our proposed approach achieves excellent performance on zero-shot cross-language clone detection.

References

[1] C. Tao, Q. Zhan, X. Hu, and X. Xia, "C4: Contrastive cross-language code clone detection," 2022.

[2] X. Cheng, Z. Peng, L. Jiang, H. Zhong, H. Yu, and J. Zhao, "Mining revision histories to detect cross-language clones without intermediates," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, pp. 696–701.

[3] K. W. Nafi, T. S. Kar, B. Roy, C. K. Roy, and K. A. Schneider, "Clcdsa: cross language code clone detection using syntactical features and api documentation," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 1026–1037.

[4] T. Vislavski, G. Rakić, N. Cardozo, and Z. Budimac, "Licca: A tool for cross-language clone detection," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2018, pp. 512–516.

[5] X. Cheng, Z. Peng, L. Jiang, H. Zhong, H. Yu, and J. Zhao, "Clcminer: detecting cross-language clones without intermediates," *IEICE TRANSACTIONS on Information and Systems*, vol. 100, no. 2, pp. 273–284, 2017.

[6] G. Rakić, "Extendable and adaptable framework for input language independent static analysis," Ph.D. dissertation, University of Novi Sad (Serbia), 2015.

[7] K. Zhang, W. Wang, H. Zhang, G. Li, and Z. Jin, "Learning to represent programs with heterogeneous graphs," in *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, 2022, pp. 378–389.

[8] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.

[9] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "Unixcoder: Unified cross-modal pre-training for code representation," *arXiv preprint arXiv:2203.03850*, 2022.

[10] M. Zhu, A. Jain, K. Suresh, R. Ravindran, S. Tipirneni, and C. K. Reddy, "Xlcost: A benchmark dataset for cross-lingual code intelligence," *arXiv preprint arXiv:2206.08474*, 2022.

[11] C. K. Roy and J. R. Cordy, "A survey on software clone detection research," *Queen's School of computing TR*, vol. 541, no. 115, pp. 64–68, 2007.

[12] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "Sourcerercc: Scaling code clone detection to big-code," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 1157–1168.

[13] C. K. Roy and J. R. Cordy, "Benchmarks for software clone detection: A ten-year retrospective," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2018, pp. 26–37.

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

[15] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, "A novel neural source code representation based on abstract syntax tree," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 783–794.

[16] H. Yu, W. Lam, L. Chen, G. Li, T. Xie, and Q. Wang, "Neural detection of semantic code clones via tree-based convolution," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 2019, pp. 70–80.

[17] H. Wei and M. Li, "Positive and unlabeled learning for detecting software functional clones with adversarial training." in *IJCAI*, 2018, pp. 2840–2846.

[18] ——, "Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code." in *IJCAI*, 2017, pp. 3034–3040.

[19] J. A. Li, Y. Li, G. Li, X. Hu, X. Xia, and Z. Jin, "Editsum: A retrieve-and-edit framework for source code summarization," in *36th IEEE/ACM International*

Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021. IEEE, 2021, pp. 155–166. [Online]. Available: https://doi.org/10.1109/ASE51524.2021.9678724

[20] J. A. Li, G. Li, Z. Li, Z. Jin, X. Hu, K. Zhang, and Z. Fu, "Codeeditor: Learning to edit source code with pre-trained models," *ACM Trans. Softw. Eng. Methodol.*, may 2023, just Accepted. [Online]. Available: https://doi.org/10.1145/3597207

[21] Z. Tao, Z. Jin, X. Bai, H. Zhao, Y. Feng, J. Li, and W. Hu, "Eveval: A comprehensive evaluation of event semantics for large language models," *arXiv preprint arXiv:2305.15268*, 2023.

[22] J. Li, Y. Zhao, Z. Jin, G. Li, T. Shen, Z. Tao, and C. Tao, "Sk2: Integrating implicit sentiment knowledge and explicit syntax knowledge for aspect-based sentiment analysis," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 2022, pp. 1114–1123.

[23] M. Wu, M. Mosse, N. Goodman, and C. Piech, "Zero shot learning for code education: Rubric sampling with deep learning inference," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 782–790.

[24] J. A. Li, Y. Li, G. Li, Z. Jin, Y. Hao, and X. Hu, "Skcoder: A sketch-based approach for automatic code generation," in *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 2023, pp. 2124–2135. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00179

[25] Y. Dong, X. Jiang, Z. Jin, and G. Li, "Self-collaboration code generation via chatgpt," *CoRR*, vol. abs/2304.07590, 2023.

[26] Y. Dong, G. Li, and Z. Jin, "CODEP: grammatical seq2seq model for general-purpose code generation," in *ISSTA*. ACM, 2023, pp. 188–198.

[27] K. Zhang, Z. Li, Z. Jin, and G. Li, "Implant global and local hierarchy information to sequence based code representation models," *arXiv preprint arXiv:2303.07826*, 2023.

[28] K. Zhang, G. Li, and Z. Jin, "What does transformer learn about source code?" *arXiv preprint arXiv:2207.08466*, 2022.

[29] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu *et al.*, "Graphcodebert: Pre-training code representations with data flow," *arXiv preprint arXiv:2009.08366*, 2020.

[30] N. Wu, Y. Liang, H. Ren, L. Shou, N. Duan, M. Gong, and D. Jiang, "Unsupervised context aware sentence representation pretraining for multi-lingual dense retrieval," *arXiv preprint arXiv:2206.03281*, 2022.

[31] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607.

[32] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[33] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," *The journal of machine learning research*, vol. 17, no. 1, pp. 2096–2030, 2016.

[34] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[35] R. Puri, D. S. Kung, G. Janssen, W. Zhang, G. Domeniconi, V. Zolotov, J. Dolby, J. Chen, M. Choudhury, L. Decker *et al.*, "Project codenet: A large-scale ai for code dataset for learning a diversity of coding tasks," *arXiv preprint arXiv:2105.12655*, vol. 1035, 2021.

[36] W. Wang, G. Li, B. Ma, X. Xia, and Z. Jin, "Detecting code clones with graph neural network and flow-augmented abstract syntax tree," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2020, pp. 261–271.

[37] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, "Convolutional neural networks over tree structures for programming language processing," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[38] J. Svajlenko, J. F. Islam, I. Keivanloo, C. K. Roy, and M. M. Mia, "Towards a big data curated benchmark of inter-project code clones," in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 476–480.

[39] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. Clement, D. Drain, D. Jiang, D. Tang *et al.*, "Codexglue: A machine learning benchmark dataset for code understanding and generation," *arXiv preprint arXiv:2102.04664*, 2021.

[40] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[41] T. Kamiya, S. Kusumoto, and K. Inoue, "Ccfinder: A multilinguistic token-based code clone detection system for large scale source code," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 654–670, 2002.

[42] L. Li, H. Feng, W. Zhuang, N. Meng, and B. Ryder, "Cclearner: A deep learning-based clone detection approach," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2017, pp. 249–260.

[43] Y. Ding, L. Buratti, S. Pujar, A. Morari, B. Ray, and S. Chakraborty, "Contrastive learning for source code with structural and functional properties," *arXiv preprint arXiv:2110.03868*, 2021.

[44] P. Jain, A. Jain, T. Zhang, P. Abbeel, J. E. Gonzalez, and I. Stoica, "Contrastive code representation learning," *arXiv preprint arXiv:2007.04973*, 2020.

[45] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[46] J. Li, R. He, H. Ye, H. T. Ng, L. Bing, and R. Yan, "Unsupervised domain adaptation of a pre-trained cross-lingual language model," *arXiv preprint arXiv:2011.11499*, 2020.

[47] G. Lample and A. Conneau, "Cross-lingual language model pretraining," *arXiv preprint arXiv:1901.07291*, 2019.

[48] Z. Jiang, A. El-Jaroudi, W. Hartmann, D. Karakos, and L. Zhao, "Cross-lingual information retrieval with bert," *arXiv preprint arXiv:2004.13005*, 2020.