Learn To Align: A Code Alignment Network For Code Clone Detection

Aiping Zhang
Nanjing University of Aeronautics and Astronautics
Nanjing, Jiangsu, P.R.China
zhangap@nuaa.edu.cn

Kui Liu
Nanjing University of Aeronautics and Astronautics
Nanjing, Jiangsu, P.R.China
brucekuiliu@gmail.com

Liming Fang†
Nanjing University of Aeronautics and Astronautics
Nanjing, Jiangsu, P.R.China
fangliming@nuaa.edu.cn

Qianjun Liu
Zhejiang University
Hangzhou, Zhejiang, P.R.China
liuqj0522@zju.edu.cn

Xinyu Yun
Nanjing University of Aeronautics and Astronautics
Nanjing, Jiangsu, P.R.China
yunxinyu@nuaa.edu.cn

Shouling Ji
Zhejiang University
Hangzhou, Zhejiang, P.R.China
sji@zju.edu.cn

***Abstract*—Deep learning techniques have achieved promising results in code clone detection in the past decade. However, existing techniques merely focus on how to extract more discriminative features from source codes, while some issues, such as structural differences of functional similar codes, are not explicitly addressed. This phenomenon is common when programmers copy a code segment along with adding or removing several statements, or use a more flexible syntax structure to implement the same function. In this paper, we unify the aforementioned problems as the problem of code misalignment, and propose a novel code alignment network to tackle it. We design a bi-directional causal convolutional neural network to extract feature representations of code fragments with rich structural and semantical information. After feature extraction, our method learns to align the two code fragments in a data-driven fashion. We present two independent strategies for code alignment, namely attention-based alignment and sparse reconstruction-based alignment. Both two strategies strive to learn an alignment matrix that represents the correspondences between two code fragments. Our method outperforms state-of-the-art methods in terms of F1 score by 0.5% and 3.1% on BigCloneBench and OJClone, respectively[1].**

***Index Terms*—Code clone detection, Bi-directional causal convolutional neural network, Code alignment.**

I. Introduction

Code clone detection aims at making decisions by measuring the similarity of two code snippets. It is valuable throughout the software development lifecycle and fundamental in many software engineering tasks, *e.g.*, code classification, code refactoring, bug detection, and malicious code detection [1]. In the past decades, a substantial amount of research effort has been devoted to detect clones. Some early methods used hand-crafted lexical and syntactic program features to identify similar code pairs, showing promising performance in detecting lexically and syntactically similar code pairs. However, in terms of detecting structurally flexible (*e.g.*, adding or removing several statements) and semantically similar code pairs, their performance is compromising.

To handle these difficult code clones, recently, more and more methods [2]–[4] seek to exploit the powerful ability of deep neural networks to detect either syntactically similar or semantically similar code fragments. Existing deep learning-based methods usually adopt a three-stage learning manner: (1) transform code fragments into abstract syntax trees (ASTs) [3], [5], [6] or program dependence graphs (PDGs) [7], (2) use neural networks to extract feature representations for each code fragment, (3) calculate the similarity of two code fragments. Despite their progress on overall performance, they did not show an absolute advantage compared with those traditional methods and there are still many challenges to be solved. In this paper, we focus on an important issue which was neglected by most existing methods, namely *code misalignment*.

The code misalignment problem mainly occurs when programmers copy all or part of one code file into another while adding or removing several statements, or using a more flexible structure to implement the same function. This phenomenon is common in the software development process, and can be also found in some benchmarks, such as BigCloneBench [8], where such kind of clones accounts for more than 95% of the total number. Even though the code fragments are usually transformed into ASTs or PDGs, the issue of code misalignment still exists because the original structure of the code is maintained. Therefore, we argue that code alignment is important for code clone detection, but it is usually ignored or not considered by existing approaches.

Nevertheless, aligning two codes is non-trivial. Intuitively, code alignment need to align similar code regions, while keeping the rest of the code regions unmatched. However, we have no cue about which part of the code is similar with another code. To address this issue, we present a novel code alignment network to perform code alignment in a data-driven

† Liming Fang is corresponding author.
[1]Our code is available at https://github.com/ArcticHare105/Code-Alignment

way. Specifically, we follow ASTNN [3] to parse source code fragments into ASTs, which are further divided into finer-granularity statement trees that are encoded into vectors by a small statement encoder. We then adopt a novel bi-directional causal convolutional neural network (BiC-CNN) to process the sequence of statement vectors into more representative and abstract representations. Finally, we propose two independent code alignment strategies, namely attention-based alignment and sparse reconstruction-based alignment, to deal with the issue of code misalignment.

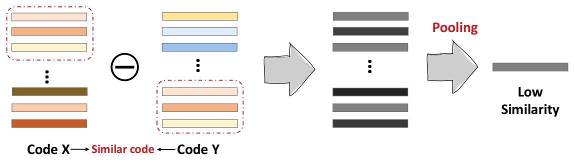
Formally, both of the two code alignment strategies strive to generate an alignment matrix, which shows the correspondence or relation between the statements of two code fragments. There are some differences between the two code alignment strategies. First, the attention-based alignment requires additional parameters, while the sparse reconstruction-based alignment does not. Besides, the attention-based alignment seeks to explore the relation between statements of two code fragments, while the sparse reconstruction-based alignment attempts to find the linear combination of statements. With the designed BiC-CNN and code alignment strategies, our method achieves the state-of-the-art performance on two public benchmark, *i.e.*, OJClone [2] and BigCloneBench [8]. The experimental results show the superiority of our method over existing methods. For example, on OJClone, our approach improves the results of F1 values from 98.4% to 99.1%, and on BigCloneBench, our method surpasses state-of-the-art methods by 1.3% on recall values.

The contributions of this paper are three-fold: (1) We propose a bi-directional causal convolutional neural network to extract feature representations for code clone detection. (2) To our limited knowledge, we are the first to investigate the issue of code misalignment in code clone detection, and we propose two effective code alignment strategies to improve performance. (3) Our method achieves the state-of-the-art performance on two public benchmarks.

```
\\ code x
public void xtest1() throws Exception {
        InputStream input = new
    FileInputStream("C:/Documentos/j931.pdf");
        InputStream tmp = new
    ITextManager().cut(input, 3, 8);
         FileOutputStream output = new
    FileOutputStream("C:/temp/split.pdf");
        IOUtils.copy(tmp, output);
        input.close();
        tmp.close();
        output.close();
    }

\\ code y
public void xtestFile2() throws Exception {
        InputStream inputStream = new
    FileInputStream(IOTest.FILE);
        OutputStream outputStream = new
    FileOutputStream("C:/Temp/testFile2.mp4");
        IOUtils.copy(inputStream, outputStream);
        inputStream.close();
        outputStream.close();
    }
```

(a) An example of two similar code fragments.



(b) An example of the pipeline for code clone detection.

Fig. 1: The motivation of our method. The similar regions of two code fragments are usually different in position, which hinders the model to detect similar codes. The residual of two codes would always show low similarity whenever they are similar or dissimilar. Therefore, code alignment is necessary.

II. Backgroud and Motivation

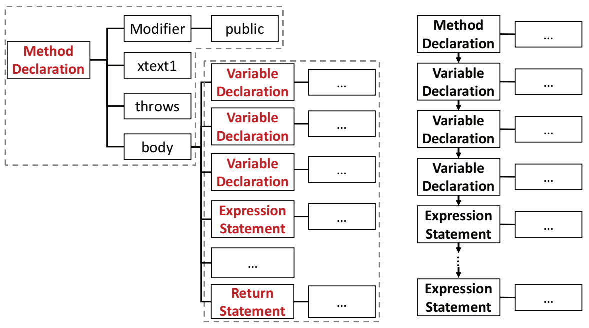
A. Code Clone Type Definition

Generally, similar codes are divide into four clones types according to their level of similarity [9]. Formally, type-1 denotes the clone type that two codes snippets are identical except for spaces, blanks, and comments, while type-2 means the same code snippets except for the variable name, type name, literal name, and function name. Type-3 means two codes are almost identical except for several additions and deletions on statements and name changes. Type-4 is usually named as functional clones, which share the same functionality but have different code structures or syntax. The existing methods can resolve the first two easy types effectively, but for type-3 and type-4, there are still many issues to be addressed.

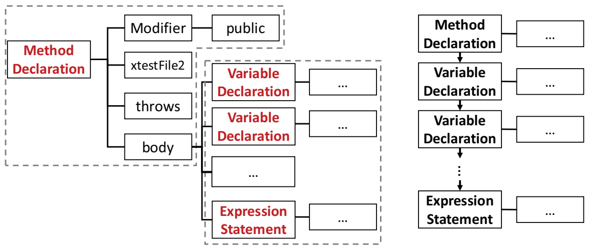
B. Code Representations

Abstract Syntax Tree (AST) is an abstract representation of the syntactic structure of the source code [10]. It represents the syntactic structure of the programming language as a tree, with each node in the tree representing a structure in the source code. The syntax is "abstract" because the syntax here does not represent every detail that occurs in the real syntax. Besides, Program Dependence Graph (PDG) is a usual code representation of control dependencies and data dependencies. However, generating PDGs is time-consuming, and there are many incomplete and uncompilable codes in some benchmarks whose PDGs cannot be obtained with existing tools.

To take advantage of the AST, tree-based methods have been developed to take ASTs as input. According to the used networks, we simply categorize these methods into three categories, *i.e.*, recursive neural networks [5], tree-based Convolutional Neural Networks (CNNs) [2] and tree-based Recurrent Neural Networks (RNNs) [3], [6].



(a) Code X



(b) Code Y

Fig. 2: Two examples of abstract syntax trees and the corresponding statement sequences.

C. Motivation Example

In our approach, we also take ASTs as input, however, transforming code fragments into ASTs cannot address the problem of code misalignment.

Fig. 1 (a) shows two similar code fragments. We can see that, the fifth line of the code *x* is similar with the fourth line of the code *y*, therefore, the similar regions of two code fragments are misalignment. We also present their abstract syntax trees and the statement sequences in Fig. 2. We note that even if we transform code fragments into ASTs and statement sequences, the problem of code misalignment still exists.

However, existing AST-based methods usually ignore this issue. Even though these methods are built on powerful neural networks, *e.g.*, RNNs and CNNs, their compromising performance on type-3 and type-4 demonstrates the necessity of explicit code alignment. More specifically, these methods usually adopt a siamese structure, that is, extracting feature representations of code fragments individually, and then calculate the similarity based on the learned feature representations. Among these processes, the most alignment-like operation is the global pooling operation [3], [7]. Due to the tree structure of AST, the feature representation of a code fragment, which are learned by CNNs or RNNs, is usually a two-dimensional tensor. In order to generate a vector for calculating similarity, a global pooling operation is adopted. However, global pooling is inherently weak in resolving code misalignment. As shown in Fig. 1 (b), if we directly calculate the residual of the feature representations of two similar codes, due to the code misalignment, the residual of each statement always show low similarities, and the global pooling also keeps the low similarity. In summary, code alignment is important for code clone detection.

III. Methodology

In this section, we first introduce an overview of our proposed approach (Sec. III-A). Next, we present the process of generating a code representation (Sec. III-B). Finally, we describe the two code alignment strategies (Sec. III-C).

A. Overview

Fig. 3 shows an illustration of our method. First, code fragments are parsed into ASTs. We follow [3] to split the large ASTs into finer-granularity statement trees (ST-trees), which are encoded into vectors by a small statement encoder. We then adopt a novel bi-directional causal CNN to process the sequence of statement vectors into more representative and abstract representations. Finally, we propose two independent code alignment strategies, namely attention-based code alignment and sparse reconstruction-based code alignment, to deal with the issue of code misalignment.

B. Code Representation Generation

a) Constructing ST-tree Sequences and Encoding: As concrete entities, code fragments should be first processed into abstract feature representations. By using existing syntax analysis tools, we can transform the whole code fragment into a large AST. However, as stated in [3], AST-based methods have their intrinsic limitations that usually destroys the original syntactic structure of source code and being vulnerable to the gradient vanishing. Therefore, we adopt the strategy of splitting AST into statement sequence.

Given a large AST T of one code fragment, we split it into a sequence of small ST-trees firstly. With a small ST-tree, a RvNN based statement encoder is then used for learning vector representations of statements [3]. Finally, we can obtain a set of feature vectors $\boldsymbol{X} \in \mathbb{R}^{T\times D}$ of ST-trees, where T denotes the number of small ST-trees and D represents the feature dimension. For more details, please refer to [3].

b) Generating discriminative features: The above operations could extract syntactical information of statements, however, there is an obvious limitation. Since each ST-tree is forwarded into the feature encoder individually, there is no interaction between ST-trees, or in other word, the extracted feature vectors $\boldsymbol{X}$ lacks structural and contextual information of the whole code fragment. In [3], they utilize a bi-directional recurrent neural network (Bi-RNN) to model the sequence of ST-trees' features. However, we argue that RNN is time-consuming due to its nature of serial processing. Prompted by the powerful ability of CNN in parallel computing and sequence modeling, we propose a CNN-based model to extract more discriminative representations.

Specifically, we design a bi-directional causal CNN to model the structural and contextual information of code fragments, which can also benefit our model in parallel computing. Causal convolutions are a type of convolution used for temporal data, which also ensures the model cannot violate the

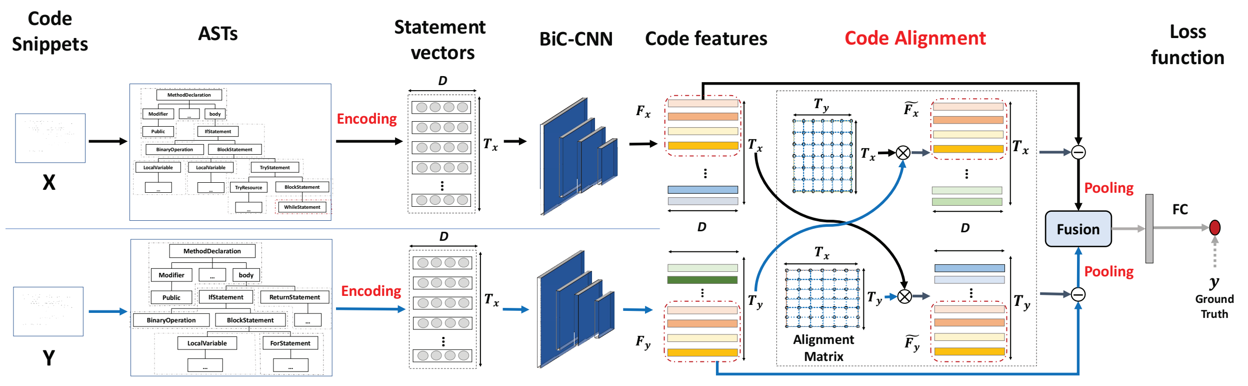


Fig. 3: The overview of our method. First, source code fragments are parsed into ASTs. We follow [3] to split the large ASTs into finer-granularity statement trees, which are encoded into vectors by a small statement encoder. We then adopt a novel bi-directional causal CNN to learn discriminative representations. Finally, we propose two independent code alignment strategies, namely attention-based alignment and sparse reconstruction-based alignment, to deal with the issue of code misalignment.

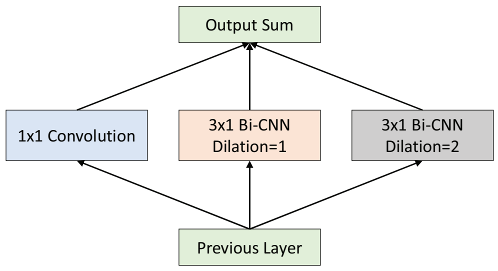


Fig. 4: Illustration of the details of the proposed bi-directional causal convolutional netowrk.

order. According to our experiments, causal CNN performs better than the vanilla CNN. We deem the reason is that the causal CNN can preserve the order of the input data, and thus retain the structural information of code fragments. Formally, the bi-directional causal convolutional layer (BiC-Conv Layer) can be formulated as:

$$\overrightarrow{\boldsymbol{F}}_t^l = \sum_{i=0}^{k} \boldsymbol{K}_i^f \cdot \boldsymbol{F}_{t-i}^{l-1}, \tag{1}$$

$$\overleftarrow{\boldsymbol{F}}_t^l = \sum_{i=0}^{k} \boldsymbol{K}_i^b \cdot \boldsymbol{F}_{t+i}^{l-1}, \tag{2}$$

$$\boldsymbol{F}_t = concat(\overrightarrow{\boldsymbol{F}}_t^l, \overleftarrow{\boldsymbol{F}}_t^l), \tag{3}$$

where l represent the l^{th} layer, $\boldsymbol{F}_{t-i}^{l-1} \in \mathbb{R}^{T\times D_i}$ is the input, $\boldsymbol{K}_i^*$ denote convolution kernels, f, b mean forward and backward, respectively. As Eq.1, after generating features along single direction, we concatenate bi-directional features to obtain the output feature $\boldsymbol{F} \in \mathbb{R}^{T\times D_o}$, so the dimension of feature $\overrightarrow{\boldsymbol{F}}_t^l$ and $\overleftarrow{\boldsymbol{F}}_t^l$ should be half of the output.

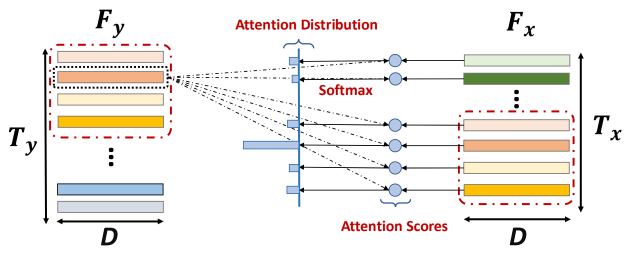
To capture information of large receptive field, based on the bi-directional causal convolutional layer and inspired by the Inception network [11], we present a bi-directional causal convolutional network. Formally, in our method, the network contains two bi-directional causal convolutional block (BiC-Conv block). As shown in Fig. 4, a BiC-Conv block includes a convolutional layer with kernel size 1×1, a BiC-Conv layer with kernel size 3×1 and a BiC-Conv layer with kernel size 3×1 and dilation 2. Therefore, one BiC-Conv block can capture the receptive field of 5×1. Finally, we summarize all the output features of the three operations as the output of the BiC-Conv block.

C. Code Alignment

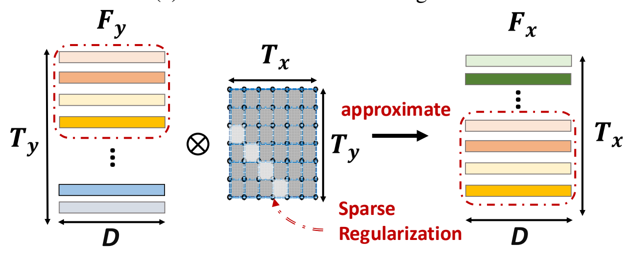
Even though the feature $\boldsymbol{F}$ is a good representation for the code fragment, it is still not enough to tackle code clone detection. For example, the similar parts in two code fragments are usually different in location, this phenomenon is severe in code clones of type-3 and type-4, which are syntactically similar and differ at the statement level. However, existing methods usually ignore this issue and utilize simple operations (*e.g.*, max or average pooling) to generate a holistic representation of the whole code fragment, which actually discards the useful structural information and thus distracts the network from accurately detecting code clones.

To address this issue, as shown in Fig. 5, we propose two strategies for code alignment, namely attention-based code alignment and sparse reconstruction-based code alignment. We show the process of aligning the code y to the code x below, the reverse process is easily achieved by swapping subscripts of notations. Suppose there are two codes: code x and code y, which may have different sequence lengths. We denote the encoded features of code x and code y as $\boldsymbol{F}_x \in \mathbb{R}^{T_x\times D}$ and $\boldsymbol{F}_y \in \mathbb{R}^{T_y\times D}$, where T_x and T_y denote the sequence lengths, as shown in Fig. 5. Note that, we not only align code y with code x, but also align code x with code x in a bi-directional way. The reason is align one code to another code must destroy the structure of the first code. Therefore, in order to maintain the structural information of two code fragments, we calculate two aligned features.

a) Attention-based code alignment (ACA): attention mechanism is widely used in various field. Actually, attention



(a) Attention-based code alignment.



(b) Sparse reconstruction-based code alignment.

Fig. 5: Illustration of two code alignment strategies. The two strategies aim to learn an alignment matrix, which represents the correspondence between the statements of two codes.

is an alignment operation [12] to some extent. To represent the attention mechanism in the form of alignment, we forward $\boldsymbol{F}_x$ and $\boldsymbol{F}_y$ into two fully connected networks respectively to obtain two compact features $\hat{\boldsymbol{F}}_x \in \mathbb{R}^{T_x \times N}$ and $\hat{\boldsymbol{F}}_y \in \mathbb{R}^{T_y \times N}$, where N represents the dimensionality. We then calculate an affinity matrix $\boldsymbol{A}_{xy} \in \mathbb{R}^{T_x \times T_y}$ as follow

$$\boldsymbol{A}_{xy} = softmax(\hat{\boldsymbol{F}}_x \hat{\boldsymbol{F}}_y^T, -1), \tag{4}$$

where $softmax(\cdot, -1)$ denotes the softmax operation along the last dimension. Intuitively, the i^{th} row and j^{th} column of matrix $\hat{\boldsymbol{F}}_x \hat{\boldsymbol{F}}_y^T$ manifests the relation between the i^{th} feature of the code x and the j^{th} feature of the code y. The higher of the value, the tighter of their relation. After normalization, the i^{th} row of $\boldsymbol{A}_{xy}$ represents the correspondences between the i^{th} feature of the code x and all features of the code y. It is natural to utilize the matrix $\boldsymbol{A}_{xy}$ to align code y to code x, which can be formulated as

$$\tilde{\boldsymbol{F}}_x = \boldsymbol{A}_{xy} \boldsymbol{F}_y. \tag{5}$$

It is obvious that the feature $\tilde{\boldsymbol{F}}_x \in \mathbb{R}^{T_x \times D}$ has the same size of the feature $\boldsymbol{F}_x$, but is formed by the feature $\boldsymbol{F}_y$ of code y. Intuitively, we can view the feature $\tilde{\boldsymbol{F}}_x \in \mathbb{R}^{T_x \times D}$ as the aligned feature corresponding to the feature $\boldsymbol{F}_x$ of code x. Similarly, we can calculate the alignment matrix $\boldsymbol{A}_{yx}$ to align code x to code y, and then obtain the aligned feature $\tilde{\boldsymbol{F}}_y$.

b) Sparse reconstruction-based code alignment (SRCA): Different from the ACA, SRCA has no extra parameter. The key idea behind SRCA is that if two code fragments are similar, some features in code y should be able to linearly reconstruct the feature of code x and the similarity between them can be computed as the reconstruction residual, and vice versa. Therefore, we attempt to obtain the linear coefficients $\boldsymbol{w}_i \in \mathbb{R}^{T_y}$ of $\boldsymbol{F}_x[i,:]$ (the i^{th} row of the feature $\boldsymbol{F}_x$) with respect to $\boldsymbol{F}_y$. With an ℓ_2-norm regularization imposed on $\boldsymbol{w}_i$, the linear representation can be formulated as

$$\min_{\boldsymbol{w}_i} \parallel \boldsymbol{F}_x[i,:] - \boldsymbol{w}_i^T \boldsymbol{F}_y \parallel_2^2 + \beta \parallel \boldsymbol{w}_i \parallel_2^2, \tag{6}$$

where $\parallel \cdot \parallel_2^2$ represents ℓ_2 normalization to impose sparse regularization. For the feature $\boldsymbol{F}_x$, Eq. 12 can be rewritten as

$$\min_{\boldsymbol{W}} \parallel \boldsymbol{F}_x - \boldsymbol{W} \boldsymbol{F}_y \parallel_2^2 + \beta \parallel \boldsymbol{W} \parallel_2^2, \tag{7}$$

where $\boldsymbol{W} \in \mathbb{R}^{T_x \times T_y}$, and β controls the smoothness of the vector $\boldsymbol{W}$. We use the least square algorithm to solve $\boldsymbol{W}$ as

$$\boldsymbol{W}^T = (\boldsymbol{F}_y \boldsymbol{F}_y^T + \beta \boldsymbol{I})^{-1} \boldsymbol{F}_y \boldsymbol{F}_x^T, \tag{8}$$

where $\boldsymbol{I} \in \mathbb{R}^{T_y \times T_y}$ is an identity matrix. Then the aligned feature $\tilde{\boldsymbol{F}}_x$ can be represented as

$$\tilde{\boldsymbol{F}}_x = \boldsymbol{F}_x \boldsymbol{F}_y^T (\boldsymbol{F}_y \boldsymbol{F}_y^T + \beta \boldsymbol{I})^{-1} \boldsymbol{F}_y. \tag{9}$$

We can also obtain the aligned feature $\tilde{\boldsymbol{F}}_y$ in the same way.

c) Calculating code similarity: With the aligned feature $\tilde{\boldsymbol{F}}_x$ generated from code y, given the extracted feature $\boldsymbol{F}_x$, we calculate the alignment residual as:

$$\boldsymbol{R}_{xy} = abs(\boldsymbol{F}_x - \tilde{\boldsymbol{F}}_x), \tag{10}$$

where $abs(\cdot)$ denotes element-wise absolution operation. Likewise, we can obtain the residual $\boldsymbol{R}_{yx}$ between the aligned feature $\tilde{\boldsymbol{F}}_y$ and the feature $\boldsymbol{F}_y$. Since the residual is a two-dimensional tensor, we apply a global max pooling along the sequence dimension to obtain two vectors $\boldsymbol{V}_{xy}, \boldsymbol{V}_{yx}$. Therefore, the similarity between code x and code y is calculated as

$$S = \mathrm{sigmoid}(\mathrm{FC}(\mathrm{Fusion}(\boldsymbol{V}_{xy}, \boldsymbol{V}_{yx}))), \tag{11}$$

where $\mathrm{Fusion}(,)$ denotes a fusion operation, *e.g.*, max, mean, *etc*. $\mathrm{FC}(\cdot)$ is a fully connected layer to map the dimensionality from D to 1, and $\mathrm{sigmoid}(\cdot)$ mimics the similarity with the value between 0 and 1. Finally, the loss function of our model is a binary cross entropy (BCE) loss as

$$\mathcal{L} = -\frac{1}{N} \sum\nolimits_{i=1}^{N} y^i log(S^i) + (1 - y^i) log(1 - S^i), \tag{12}$$

where y^i and S^i represent the ground truth and similarity of the i^{th} code pair in a mini-batch of size N.

IV. Experimental Settings

Empirical evaluation of our proposed methods is performed through several experiments. Before showing the experimental results, we first describe some research questions, the used benchmarks, and the overall experimental setup.

A. Research Questions

Based on the evaluation on the two benchmarks, we target at investigating the following research questions:

- **RQ1: How does our approach perform in code clone detection compared with state-of-the-art methods?** We aim to compare our method with the state-of-the-art methods to show the effectiveness of our method. Besides, it can also demonstrate the necessity of code alignment, which was mostly ignored by existing methods.
- **RQ2: What is the effect of the designed bi-directional causal convolutional neural network?** With this RQ, we aim to validate the effectiveness and efficiency of the proposed BiC-CNN, and show its superiority compared to vanilla RNNs.
- **RQ3: What is a suitable way to fuse the two residuals of two codes?** Calculating two residuals is essential for keeping the structural information. Therefore, we investigate which fusion strategy is suitable in our method.
- **RQ4: What are the effects of the two code alignment strategies?** With this RQ we aim to comprehensively evaluate the proposed code alignment strategies by showing both quantitative and qualitative results.

B. Dataset

We evaluate our method on two benchmarks: BigCloneBench [8] and OJClone [2]. The reason we choose BigCloneBench is because it is a widely used benchmark for code clone detection, and the type-3 and type-4 clone types account for the majority. For OJClone, programs have the same functionality if they aim to solve the same problem, so it is also appropriate to evaluate our method. Besides, the BigCloneBench and OJClone are based on Java and C, respectively, which allows us to evaluate the generalization ability of our approach in detecting code clones.

a) BigCloneBench: This dataset contains over 6,000,000 positive clone pairs and 260,000 negative clone pairs. In BigCloneBench, all the code fragments are based on Java language. Due to the ambiguity between the definitions of type-3 and type-4, the code pairs of type-3 and type-4 are further partitioned by a similarity score on statement-level: strongly type-3 (ST3), moderately type-3 (MT3) and weakly type-3/type-4 (WT3/T4) are defined with similarity in [0.7, 1.0), [0.5, 0.7) and [0.0, 0.5), respectively. The majority of code clone pairs are weakly type-3/type-4, so BigCloneBench is suitable to be used for validating semantic clone detection.

b) OJClone: This dataset contains 104 programming problems together with different source codes students submit for each problem [2]. In OJClone, two different source codes that solve the same programming problem are considered as a code clone pair, due to their same functionality, which belongs to type-3 or type-4.

C. Implementation Details.

The convolutional neural network used in our model contains two layers, both consisting of 256 channels. Our method is implemented by PyTorch [14]. During training, we loop through each code pair in the current training batch and accumulate the gradient to deal with code fragments of different lengths. We use Adam [15] to optimize the model. The training procedure stops at 8 epochs with the learning rates 0.01. We set batch size as 10 for OJClone and 16 for BigCloneBench, and the weight decay is chosen as 0.0005.

V. Experimental Results and Analyses

We present the experimental results in this section to answer the research questions.

A. RQ1: Comparison with the State-of-the-art Methods

In this question, we would like to find out whether our method is effective on code clone detection. First, we briefly introduce several code clone approaches as follows: (1) Deckard [1]: a code clone approach based on syntax tree, which adopts Euclidean distance to calculate the similarity between code fragments. (2) DLC [5]: a deep-learning-based method that uses a recursive auto-encoder to extract deep features. (3) CDLH [6]: a deep learning approach to learn syntactic features of code clone. (4) Deepsim [7]: a semantic-based approach that utilizes supervised deep learning to measure functional code similarity. (5) ASTNN [3]: a state-of-the-art model, which splits a large AST into a sequence of small statement trees, and encodes the statement trees into lexical and syntactical vectors. (6) FA-AST+GMN [13]: a state-of-the-art model, which augments original ASTs with explicit control and data flow edges, and then Graph Matching Networks (GMN) is utilized to measure the similarity of code pairs. (7) SSFL [4]: a state-of-the-art model, which learns a novel joint code representation to learn hidden syntactic and semantic features of source codes.

As shown in Tab. I, on BCB dataset, all the methods obtain promising performance in detecting similar code fragments in type-1 and type-2, since both code fragments are almost the same except some different function and variable names. While for other types, the effectiveness of our method can be represented. Specifically, Deckard only utilizes hand-crafted features as code representations and calculated similarity by using euclidean distance. Therefore, it considers each dimension the same for calculating similarity, which is not enough and thus obtains a low recall value. Moreover, even though the RtvNN and CDLH adopt an RNN language model to learn code embeddings, they treat the AST as a binary tree, which destroy the original structure of the code, leading to inferior performance. Even though the ASTNN [3] explicitly model sequential naturalness of statements in code fragments. However, they ignore the issue of code misalignment, which is essential for code clone detection. Therefore, it is unsurprising that our method significantly outperforms ASTNN in the clone types of MT3 and T4, which can be viewed as semantic clones that extremely require code alignment to alleviate the influence of other factors. Moreover, compared with the state-of-the-art method FA-AST+GMN [13], our methods still achieves better performance, demonstrating the state-of-the-art performance of our method on BigClonebench dataset.

TABLE I: Code clone detection results on BigCloneBench. P, R, F1 denote precision, recall and F1 score, respectively. Note that, the column of *ALL* is a weighted sum result according to the percentage of various clone types [6].

Method	T1			T2			ST3			MT3			T4			ALL		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Deckard [1]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	93.0	2.0	3.0
RtvNN [5]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	95.0	1.0	1.0
CDLH [6]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	92.0	74.0	82.0
ASTNN [3]	100.0	100.0	100.0	100.0	100.0	100.0	100.0	99.6	99.8	100.0	97.9	98.9	93.3	92.2	92.8	93.4	92.3	92.9
FA-AST+GMN [13]	100.0	100.0	100.0	100.0	100.0	100.0	100.0	99.6	99.8	100.0	96.5	98.2	95.7	93.5	94.6	95.8	93.6	94.7
Ours (Attention)	100.0	100.0	100.0	100.0	100.0	100.0	99.8	**100.0**	**99.9**	99.9	98.2	99.0	95.2	94.3	94.7	95.3	94.4	94.8
Ours (Sparse)	100.0	100.0	100.0	100.0	100.0	100.0	99.8	**100.0**	**99.9**	100.0	**98.5**	**99.2**	**95.8**	**94.8**	**95.3**	**95.9**	**94.6**	**95.2**

TABLE II: Code clone detection results on OJClone. P, R, F1 denote precision, recall and F1 score, respectively.

Method	Metric		
	P	R	F1
Deckard [1]	99.0	5.0	10.0
DLC [5]	52.5	68.3	59.4
CDLH [6]	47.0	73.0	57.0
PDG+HOPE [16]	76.2	7.0	12.9
PDG+GGNN [17]	77.3	43.6	55.8
Deepsim [7]	70.0	83.0	76.0
ASTNN [3]	98.9	92.7	95.5
SSFL [4]	97.0	95.0	96.0
Ours (Attention)	99.4	97.5	98.4
Ours (Sparse)	**99.7**	**98.5**	**99.1**

TABLE III: Code clone detection results on OJClone. *None* denotes we do not use any network to proces the input, that is, the features are directly used to calculate similarities. Note that, all of these methods are without code alignment.

Method		Metric		
		P	R	F1
None		92.1	81.9	86.7
CNN	vanilla (128-128)	98.3	93.8	96.2
	128-128	**99.5**	**96.2**	**97.8**
	128-64	98.4	94.8	97.0
	128-32	98.1	93.6	95.8
	64-64	99.2	94.7	96.9
	64-32	99.2	94.7	96.9
	32-32	99.0	94.4	96.6
RNN	vanilla RNN	97.8	92.6	95.1
	Bi-RNN	98.9	92.7	95.7

As shown in Tab. II, on OJClone, similar results can be also observed. It can be observed that CDLH, DeepSim, ASTNN, SSFL and our approach obtain better results than those unsupervised methods Deckard, DLC, and PDG in terms of F1 value. Since the code types in OJClone are almost functional clones, this phenomenon demonstrates that unsupervised methods cannot capture the similarity of functional clones. Compared with other supervised method: CDLH, Deepsim and SSFL, our method still achieves better performance. Especially, compared with the recent method ASTNN [3], our method significantly surpass it by **0.8%**, **5.8%** and **3.6%** on the metrics of precision, recall and F1 score, respectively, demonstrating the state-of-the-art performance on OJClone.

B. RQ2: Effectiveness of the BiC-CNN

In our method, we adopt a bi-directional causal convolutional neural network to extract more discriminative and abstract features of code fragments. Generally speaking, for a sequential input, an recurrent neural network is usually used (*e.g.*, ASTNN [3] utilizes a bi-directional recurrent neural network). However, we argue that the convolutional neural network is more appropriate for its powerful ability of sequence modeling and parallel computing.

TABLE IV: Evaluation of the number of parameters and inference time on OJClone.

Method	Parameters	Inference Time	Metric		
			P	R	F1
Bi-RNN (2 layers) [3]	0.89M	38.33ms	98.9	92.7	95.7
Bi-RNN + Attention	0.92M	53.21ms	99.2	97.1	98.1
Bi-RNN + Sparse	0.89M	48.53ms	99.3	97.9	98.6
Bi-CNN	**0.62M**	**32.59ms**	97.0	95.0	96.0
Bi-CNN + Attention	0.65M	43.55ms	99.4	97.5	98.4
Bi-CNN + Sparse	**0.62M**	38.22ms	**99.7**	**98.5**	**99.1**

In Tab. III, we show the results of different methods, which are based on RNNs or CNNs, on the OJClone dataset. Note that, for fair comparisons, we do not adopt code alignment in these experiments. As we can see, when we do not use CNN or RNN, the performance is very low, demonstrating the importance of introducing an additional module to extract discriminative features. Among RNN-based methods, the bi-directional RNN achieves better performance compared to the vanilla RNN. It makes sense because the bi-directional RNN can capture richer sequential naturalness of the code. Compared with the RNN, CNN shows its superior ability in code clone detection. Specifically, compared with the bi-directional RNN, our proposed bi-directional causal CNN has improvements on precision, recall and F1 score by 0.6%, 3.5% and 2.3%, respectively. Similar to the results of RNNs, the proposed bi-directional causal CNN obtains better performance compared to the vanilla CNN. Therefore, we can obtain such a conclusion: our bi-directional causal CNN can capture sequential information and simultaneously integrates the advantages of CNNs.

To obtain the optimal setting, we also evaluate different models with different output channels. For a fair comparison, we fix the number of layers as 2 in these experiments. We can see that the model with output channels of 128 and 128 achieves the best performance.

In Tab. IV, we also evaluate the number of parameters as well as the inference time of different methods. As we can see, compared with the bi-directional RNN, our proposed bi-directional causal CNN has less parameters and requires less

TABLE V: Evaluation of different fusion strategies of two residuals on OJClone. The *single residual* means we only calculate a single residual and thus require no fusion strategy.

Method		Metric		
		P	R	F1
Attention	single residual	98.8	97.0	97.9
	sum	99.3	97.2	98.2
	max	99.5	97.4	98.4
	mean	99.3	97.6	98.3
	concat	**99.4**	**97.5**	**98.4**
Sparse Reconstruction	single residual	99.1	97.8	98.4
	sum	99.3	98.1	98.7
	max	99.5	98.4	98.9
	mean	99.6	98.2	98.9
	concat	**99.7**	**98.5**	**99.1**

time for inference, but achieves better detection performance.

C. RQ3: Evaluation of Different Fusion Strategies

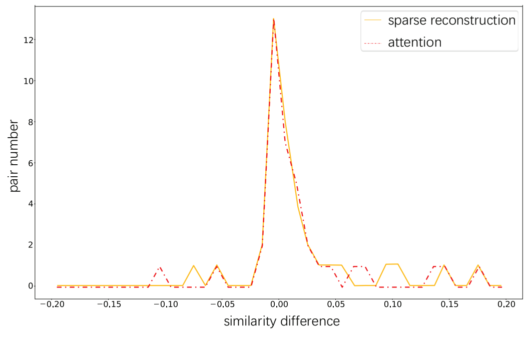
As stated in the section of the code alignment, we calculate two residuals of code x and code y to achieve the bi-directional alignment, so as to maintain the structural information of the two codes. In Tab. V, we first validate the necessity of calculating two residuals. We note that there is an obvious performance degradation when we only calculate a single residual. For example, in terms of the attention-based code alignment, the performance drops by 0.5%, showing the importance of calculating two residuals.

To figure out which fusion strategy is suitable for aggregating two residuals, in Tab. V, we further evaluate four fusion strategies (*i.e.*, *sum*, *max*, *mean* and *concat*) on the OJClone dataset. Specifically, the *sum* operation means adding the corresponding elements of the two feature vectors to build the fusion vector, *i.e.*, $\boldsymbol{V} = \boldsymbol{V}_{xy} + \boldsymbol{V}_{yx}$. Likewise, *max* and *mean* operation denote element-wise max and element-wise mean, respectively. The *concat* means concatenating two residual vectors, *i.e.*, $\boldsymbol{V} = [\boldsymbol{V}_{xy}, \boldsymbol{V}_{yx}]$. As we can see, all of the four strategies can achieve promising performance. Among the four strategies, the *concat* achieves the best performance due to its better ability to retain information. Besides, since the dimension of the concatenated feature is twice as large as the features generated by other strategies, more parameters are required, which may lead to better performance.

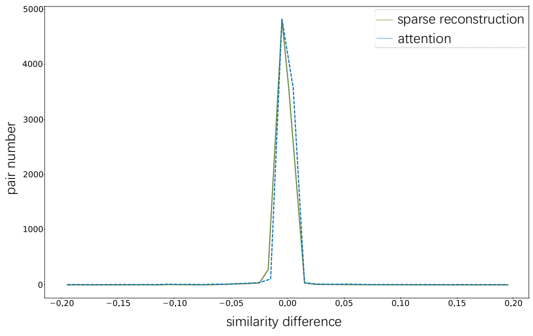
D. RQ4: Evaluation of Two Code Alignment Strategies

In our method, code alignment plays an important role. The two types of code alignment can both work in code clone detection. As shown in Tab. I and Tab. II, the attention-based code alignment and sparse-reconstruction-based code alignment can significantly improve the results. For example, on OJClone, they improves precision, recall and F1 score by 0.5%, 4.8%, 2.9% and 0.8%, 5.8%, 3.6%, respectively. Besides, from Tab. IV, we can find that the two code alignment strategies can consistently improve the performance upon the baseline models, suggesting that both code alignment strategies are valid for any baseline model.

Comparing two different ways of code alignment, the sparse reconstruction-based strategy achieves better performance. The reason may because the attention-based strategy only allows non-negative linear coefficients but the sparse-reconstruction-based strategy allows both positive and negative coefficients, which would enable a more powerful ability of alignment.



(a) Positive clone pairs



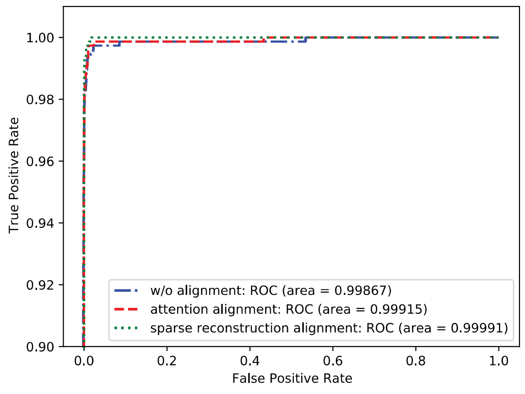
(b) Negative clone pairs

Fig. 6: Illustration of the statistical distribution of similarity difference before and after the alignment operation. Here, we utilize histograms of 50 bins to show the results. The *horizontal* axis represents the similarity after alignment minus the similarity before alignment. The *vertical* axis represents the number of code pairs. For the sparse reconstruction-based code alignment, the similarities of 649 positive clone pairs become larger while only 19 pairs become smaller. For the attention-based code alignment, the similarities of 623 positive clone pairs become larger while only 45 pairs become smaller.

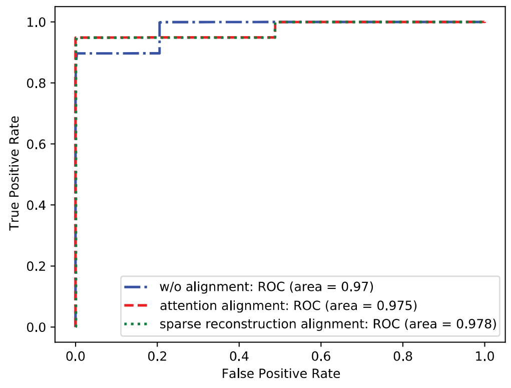
To attain an insight into the two types of code alignment, we show the similarities of positive code pairs and negative code pairs in Fig. 6. As we can see, after applying code alignment, the similarities of positive code pairs are increased, meanwhile the similarities of negative code pairs are decreased, demonstrating the effectiveness of code alignment in code clone detection that heavily requires accurate similarity measurement.

We further draw the ROC curve of different variants of our method on two datasets, the results are shown in Fig. 7. The ROC curve is a graphical plot that illustrates the diagnostic ability of a binary classifier as its discrimination threshold is varied[2]. Similar to the results shown in Tab. II and Tab.

[2]https://en.wikipedia.org/wiki/Receiver_operating_characteristic



(a) OJClone



(b) Type-4 of BCB

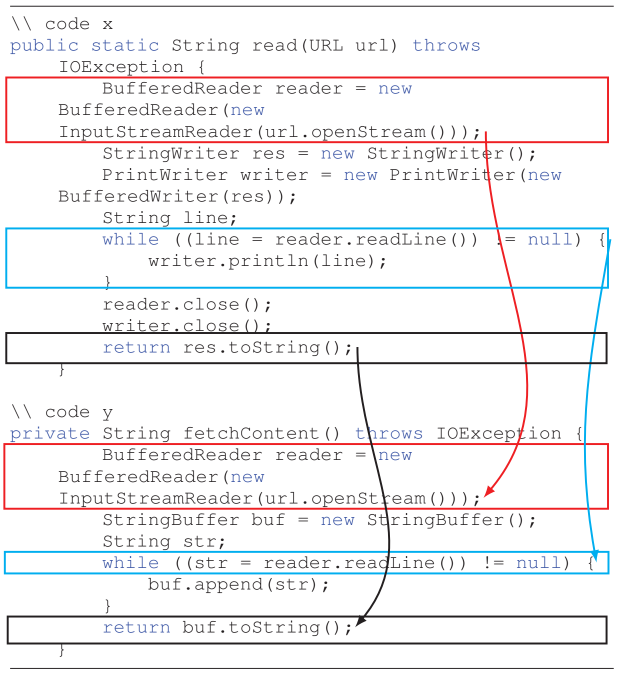
Fig. 7: The ROC curve and AUC score of different methods.

I, the sparse reconstruction-based code alignment achieves the highest AUC score among the three approaches, and the attention-based coda alignment also obtains a better AUC score compared with the methods without code alignment.

To attain intuitive insights into our code alignment, we also show some qualitative results. During code alignment, the alignment matrix (*i.e.*, $\boldsymbol{A}_{xy}$ of the attention-based code alignment and $\boldsymbol{F}_x\boldsymbol{F}_y^T(\boldsymbol{F}_y\boldsymbol{F}_y^T+\beta\boldsymbol{I})^{-1}$ of the sparse reconstruction-based code alignment) show the correspondence between two codes, so we can figure out which part of code X corresponds to which part of code Y from the alignment matrix. In Fig. 8, we show the two similar codes, where the alignment matrix is visualized by the arrows between two codes, which connect the code lines with high alignment scores. From Fig. 8, we can find that our proposed strategy obtains astonishing code alignment performance, therefore, it is unsurprising that the code alignment would prompt code clone detection, since it eliminates other influence factors such as locations and allows the model to intently focus on detecting similar codes.

VI. Threats to Validity

There are three main threats to the validity. First, our method is based on a shallow feature extraction module similar to the previous methods. However, if we build our method on a powerful network such as ResNet [18], it remains to be seen



```
\\ code x
public static String read(URL url) throws
    IOException {
        BufferedReader reader = new
    BufferedReader(new
    InputStreamReader(url.openStream()));
        StringWriter res = new StringWriter();
        PrintWriter writer = new PrintWriter(new
    BufferedWriter(res));
        String line;
        while ((line = reader.readLine()) != null) {
            writer.println(line);
        }
        reader.close();
        writer.close();
        return res.toString();
    }

\\ code y
private String fetchContent() throws IOException {
        BufferedReader reader = new
    BufferedReader(new
    InputStreamReader(url.openStream()));
        StringBuffer buf = new StringBuffer();
        String str;
        while ((str = reader.readLine()) != null) {
            buf.append(str);
        }
        return buf.toString();
    }
```

Fig. 8: Illustration of the alignment results. We show three statement pairs that have alignment scores greater than 0.5. It is obvious that the code alignment is accurate and effective.

whether the proposed code alignment will have any effect. Second, our method are evaluated on only two programming languages, *i.e.*, Java and C. Actually, our method can be potentially used to detect code clones for other programming languages, whose ASTs can be also extracted. However, since we have not evaluated this, we cannot claim the effectiveness on these programming languages. Third, the effectiveness of our approach is limited by the quality of current benchmarks. With the introduction of deep learning and the addition of various strategies, such as code alignment, the existing datasets have entered performance saturation and cannot well reflect the effectiveness of the method. Therefore, we plan to collect a larger dataset.

VII. Related Work

A. Code Clone Detection

Traditional code clone detection techniques meanly use representations such as text [19], [20], tokens [21], syntax indicators or abstract syntax trees [1] for measuring the similarity between code snippets. With the breakthrough of deep learning in Natural Language Processing (NLP), deep learning has been applied in the field of software engineering due to the similarities between programs and natural languages. DLC [5] introduces a language model to detect clones, where a recursive learning process are proposed to learn fragment representations. CDLH [6] propose to transform ASTs into binary hash codes by leaning hash functions, and then utilizes an AST-based LSTM to capture the lexical and syntactical information of source codes. Deepsim [7] attempt to encode the control flow and data flow into a semantic matrix, which are used to detect functional clones with deep neural networks. FA-AST [13] present two graph-based networks to detect clones. Apart from the graphs generated from ASTs, they also introduce the the control flow and data flow information into the graph. The most related approach with ours is ASTNN [3], it splits a large AST into a sequence of small statement trees, and encodes the statement trees to vector representations, which are further forwarded into a bi-directional recurrent neural network and finally produce the vector representation of a code fragment. Our method also follow the ASTNN [3] to split a AST into a sequence of small statement trees, however, we design a more powerful and efficiency bi-directional convolutional neural network to obtain code representations. Moreover, different from the max pooling operation used in ASTNN, which is prone to loss some structural information, we utilize code alignment to first align code, so as to balance the structural information preservation and similarity measurement.

B. Alignment-based Methods

Alignment is widely adopted in various fields. Usually, there are two or more objects, which are from multiple sources (*e.g.*, modalities) or a single source, we need to find their common points but hindered by some factors, *e.g.*, domain gap [22]–[24], occlusion [25], [26], pose variance [27], [28], alignment strategies can be used. For example, Karpathy *et al.* [22] use a structural ranking loss to align image-sentences pairs. DANs [23] jointly leverages visual and textual attention mechanisms to align visual regions and words. He *et al.* [25] propose an alignment-free approach, which utilize sparse reconstruction to handle the difficulty of person occlusion in person re-identification. To better recognize human faces, some methods [27] propose to first align faces according to some key points. Similar idea is adopted in P2RN [28] to identify human actions. For code clone detection, there are also some factors that impede accurate detection, *e.g.*, different structures or orders of two codes, demonstrating the necessity of introducing some code alignment strategies.

VIII. Conclusion and Future Work

In this paper, we proposed to detect clone code fragments with code alignment strategies. We presented a novel bi-directional causal CNN to process the sequence of statement vectors into discriminative representations. Based on the learned features, we designed two code alignment strategies, namely attention-based code alignment and sparse reconstruction-based code alignment, to mitigate the influence of statement differences and enable more accurate detection.

Currently, our proposed code alignment strategies can be only apply to the features generated by CNNs or RNNs. However, the graph representation has shown its effectiveness for code clone detection in many approaches. In the future, we aim to focus on how to make the code alignment compatible with graph representations.

Acknowledgements

This research was supported by the National Key R&D Program of China (Grant No.2020AAA0107704), the National Natural Science Foundation of China (Grant 62172214, 62076125, U20B2049, U20B2050), the Natural Science Foundation of Jiangsu Province, China (Grant No. BK20210279), the Open Project Program of the State Key Laboratory of Mathematical Engineering and Advanced Computing (No. 2020A06) and the Open Project Program of the Key Laboratory of Safety-Critical Software (NUAA), Ministry of Industry and Information Technology (No. XCA20026).

References

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

[2] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, "Convolutional neural networks over tree structures for programming language processing," in *AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[3] 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 *International Conference on Software Engineering*. IEEE, 2019, pp. 783–794.

[4] C. Fang, Z. Liu, Y. Shi, J. Huang, and Q. Shi, "Functional code clone detection with syntax and semantics fusion learning," in *ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020, pp. 516–527.

[5] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, "Deep learning code fragments for code clone detection," in *IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2016, pp. 87–98.

[6] H. Wei and M. Li, "Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code." in *International Joint Conference on Artificial Intelligence*, 2017, pp. 3034–3040.

[7] G. Zhao and J. Huang, "Deepsim: deep learning code functional similarity," in *Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 141–151.

[8] 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 *International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 476–480.

[9] Q. U. Ain, W. H. Butt, M. W. Anwar, F. Azam, and B. Maqbool, "A systematic review on code clone detection," *IEEE Access*, vol. 7, pp. 86 121–86 144, 2019.

[10] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," in *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*. IEEE, 1998, pp. 368–377.

[11] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2015, pp. 1–9.

[12] Y. Chen, Y. Kalantidis, J. Li, S. Yan, and J. Feng, "A2-nets: Double attention networks," *arXiv preprint arXiv:1810.11579*, 2018.

[13] 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 *International Conference on Software Analysis, Evolution and Reengineering*. IEEE, 2020, pp. 261–271.

[14] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, vol. 32, 2019, pp. 8026–8037.

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[16] M. Tufano, C. Watson, G. Bavota, M. Di Penta, M. White, and D. Poshyvanyk, "Deep learning similarities from different representations of source code," in *IEEE/ACM 15th International Conference on Mining Software Repositories*. IEEE, 2018, pp. 542–553.

[17] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," *arXiv preprint arXiv:1711.00740*, 2017.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 630–645.

[19] B. S. Baker, "A program for identifying duplicated code," *Computing Science and Statistics*, pp. 49–49, 1993.

[20] ——, "Parameterized pattern matching: Algorithms and applications," *Journal of computer and system sciences*, vol. 52, no. 1, pp. 28–42, 1996.

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

[22] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2015, pp. 3128–3137.

[23] H. Nam, J.-W. Ha, and J. Kim, "Dual attention networks for multimodal reasoning and matching," in *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2017, pp. 299–307.

[24] X. Xu, T. Wang, Y. Yang, L. Zuo, F. Shen, and H. T. Shen, "Cross-modal attention with semantic consistence for image–text matching," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 12, pp. 5412–5425, 2020.

[25] L. He, Y. Wang, W. Liu, H. Zhao, Z. Sun, and J. Feng, "Foreground-aware pyramid reconstruction for alignment-free occluded person re-identification," in *IEEE International Conference on Computer Vision*. IEEE, 2019, pp. 8450–8459.

[26] J. Miao, Y. Wu, P. Liu, Y. Ding, and Y. Yang, "Pose-guided feature alignment for occluded person re-identification," in *IEEE International Conference on Computer Vision*. IEEE, 2019, pp. 542–551.

[27] A. Jourabloo, M. Ye, X. Liu, and L. Ren, "Pose-invariant face alignment with a single cnn," in *IEEE International Conference on Computer Vision*. IEEE, 2017, pp. 3200–3209.

[28] L. Huang, Y. Huang, W. Ouyang, and L. Wang, "Part-aligned pose-guided recurrent network for action recognition," *Pattern Recognition*, vol. 92, pp. 165–176, 2019.