Early Experience with Transformer-Based Similarity Analysis for DataRaceBench

Winson Chen[1,2] Tristan Vanderbruggen[1] Pei-Hung Lin[1] Chunhua Liao[1] Murali Emani[3]

[1]Lawrence Livermore National Laboratory, Livermore, CA 94550, USA
[2] University of California Santa Cruz, Santa Cruz, CA 95064, USA
[3] Argonne National Laboratory, Lemont, IL 60439, USA

***Abstract*—DataRaceBench (DRB) is a dedicated benchmark suite to evaluate tools aimed to find data race bugs in OpenMP programs. Using microbenchmarks with or without data races, DRB is able to generate standard quality metrics and provide systematic and quantitative assessments of data race detection tools. However, as the number of microbenchmarks grows, it is challenging to manually identify similar code patterns for DRB, within the context of identifying duplicated kernels or guiding the additions of new kernels. In this paper, we experiment with a transformer-based, deep learning approach to similarity analysis. A state-of-the-art transformer model, CodeBERT, has been adapted to find similar OpenMP code regions. We explore the challenges and the solutions when applying transformer-based similarity analysis to new source codes which are unseen by pre-trained transformers. Using comparative experiments of different variants of similarity analysis, we comment on the strengths and limitations of the transformer-based approach and point out future research directions.**

***Index Terms*—Benchmarks, OpenMP, Data Races, Tools, Deep Learning, Transformers**

I. Introduction

DataRaceBench (DRB) is a dedicated benchmark suite to evaluate tools aimed at finding data race bugs in OpenMP programs. Since its initial release in 2017 [1], DRB has incorporated various additions to have a richer set of microbenchmark programs to cover the latest OpenMP constructs, base programming languages (such as C/C++ and Fortran) and modern parallel hardware devices (e.g. GPUs) [2]–[4]. Using microbenchmarks with or without data races, DRB is able to generate standard quality metrics (such as accuracy and F-1 score) and provide systematic and quantitative assessments of data race detection tools. The existing workflow using DRB has several steps: compiling the microbenchmark source codes, running tools being evaluated, collecting the reports generated by the tools, and processing the report files against ground truth to calculate the quality metrics of a tool. Due to its good design and automated workflow, DRB has been widely adopted by tool developers [5]–[11].

As the number of microbenchmarks grows, it is increasingly challenging to manually maintain the benchmarks in DRB. In order to have a minimum collection of microbenchmark programs with maximum coverage of OpenMP code patterns with or without data races, similarity analysis has been attempted to help identify similar code patterns inside the set of microbenchmarks. The analysis can also help with another typical maintenance task: given a new code pattern suggested by users, we have to decide if it should be added into DRB or not. However, the similarity analysis methods in prior work [3], [4] are primitive. They rely on manually selected static and dynamic properties of OpenMP code regions. They also involve specialized compilers and tools to extract the selected properties. As we try to add more microbenchmarks into DRB, it becomes increasingly important to have enhanced, automated similarity analysis techniques to avoid duplicated microbenchmarks being collected in DRB.

In the meantime, a new type of deep learning model called transformer has become a mainstream choice for natural language problems since its introduction in the aptly named "Attention is All You Need" paper in 2017 [12]. Seeing the rise of transformer-based models such as BERT [13], DALL-E and GPT-3 [14], some researchers have predicted that transformer models may become a critical foundation for a wide range of scientific and technical domains [15]. Due to the similarity between natural languages and programming languages, transformers also have been extended to solve various programming language processing (PLP) tasks [16], including clone detection, code completion (e.g. OpenAI's CodeX [17]) and program synthesis (e.g. AlphaCode [18]). A recent paper [19] summarizes typical PLP tasks, model architectures, compiler tools and datasets in order to facilitate the assembling of machine learning pipelines to solve a given programming language processing task. However, it is unclear how transformers can be leveraged in the context of parallel programming languages such as OpenMP.

In this paper, we report our early experience with transformers when adapting a state-of-art transformer for similarity analysis of OpenMP code regions, within the context of maintaining DRB. This paper has the following contributions:

1) a machine learning pipeline designed for adapting pre-trained transformer models for similarity analysis of OpenMP code regions,
2) a summary of challenges of the transformer-based approach and a set of solutions, including two fine-tuning steps using new datasets.

This work is supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Program under contract number DE-AC02-06CH11357 and Award Number DE-SC0021293. Prepared by LLNL under Contract DE-AC52-07NA27344 (LLNL-CONF-838838).

3) a comparison of different ways of leveraging a pre-trained transformer model, and
4) a discussion about the strengths and limitations of transformer-based similarity analysis for parallel programming codes.

The remainder of this paper is organized as follows. In the next section, we give an overview of DRB. Section III presents the background of similarity analysis and relevant prior implementations used by DRB. A new similarity analysis using transformers is described in Section IV. Experiments are discussed in Section V. Finally, related work is mentioned in Section VI and conclusions are drawn in Section VII.

II. DataRaceBench

DataRaceBench is an open-source benchmark suite designed to systematically and quantitatively evaluate the effectiveness of data race detection tools. The design of DRB focuses on parallel programs written in OpenMP, the popular parallel programming model for multi-threaded applications. To enable support for data race tool evaluation, DRB includes a set of microbenchmark programs with or without data races. The microbenchmarks come from three main sources: manually written, extracted from real scientific applications, or automatically generated optimization variants.

As shown in Fig. 1, DRB has grown gradually through joint developments from the DRB development team and community contributions since its release in 2017 [1]–[4]. Early versions (1.1 and 1.2) only support C/C++. Later versions (since 1.3) contain Fortran support. In the latest version 1.4.0, DRB collects 181 C/C++ microbenchmarks and 172 Fortran microbenchmarks. The latest suite includes parallel programs supporting OpenMP 3.x to OpenMP 5.x, both CPU and GPU hardware devices, both static and dynamic tools, as well as dockerized workflows and fine-grain correctness checking.

DRB is expected to continue to grow and collect more microbenchmarks representing OpenMP data race code patterns extracted from different domains. Ideally, DRB should contain a minimum collection to reduce the overhead of evaluating different tools, if possible. At the same time, it must represent the maximum coverage in terms of code patterns and the OpenMP language specifications. Version 1.3 and 1.4 of DRB leveraged two primitive similarity analysis methods to help identify duplicated code patterns and guide the addition of new microbenchmarks. We will explain the background of similarity analysis and details of the prior attempts in the following sections.

III. Similarity Analysis

A. Similarity analysis applications

Code similarity analysis aims to determine if two given code fragments share similar syntactic or semantic representations. Code semantic similarity analysis is a more popular type of code similarity analysis that can be applied to tasks such as code recommendation, bug detection, and language-to-language translation. Another variation of code similarity analysis addresses the code clone detection to determine if one code fragment is cloned from another source code fragment. This type of analysis can be helpful to identify code plagiarism, identify reusable code for library development, and detect fault/error propagation caused by code cloning.

To demonstrate the similarities among existing OpenMP programs in DRB, Fig. 2 shows three code fragments from different microbenchmarks in DRB. Parallel code regions extracted from DRB001 and DRB029 share the same OpenMP syntax to represent `parallel for` loops. Although there exist minor differences in the array subscripts and they have different types of loop-carried dependency causing the data race, we could still consider these two code fragments largely similar. But they differ in some critical aspects (e.g. dependence types) DRB cares about. DRB009 has several differences compared to the other two code fragments: (1) it has an additional OpenMP clause to represent its data-sharing attribute, (2) it has a different upper loop bound, and (3) it has a different statement in the loop body. With these differences, we would consider DRB009 to have much less similarity compared to the other two code fragments presented here.

It is interesting that a benchmark suite like DRB should contain both similar and different code regions, but not duplicated regions in the view of code patterns relevant to data races. A good similarity analysis should highlight the features important to a goal and help the maintainers to make a decision.

B. Definition of similarity analysis

In the nutshell, similarity analysis for source codes determines if the given source programs are considered similar. However, the definition of similarity has to be clearly specified to avoid confusion and misuse within a given context. Depending on the usage of similarity analysis, the similarity can be applied to syntax-based and semantic-based code representations, equivalence of program functions, source code patterns, or the mix of the above. For example, the definition of similarity in source code clone detection would focus on syntactic similarity of the source codes but allow minor variations, such as differences in variable names and the order of operands used in statements. On the other hand, similarity analysis applied for program understanding and functionality equivalence could determine syntactically dissimilar code blocks to be semantically similar.

Four similarity types are defined to distinguish code clone types to support the evaluation of clone detection tools [20]. The following summarizes the definitions of these types:

- Type-1: Source code fragments are identical except for differences in white spaces, layout and comments.
- Type-2: In addition to Type-1, source code fragments are identical except differences in identifier names and literal values.
- Type-3: In addition to Type-1 and Type-2, source code fragments are syntactically similar but with differences at the statement level. Code fragments can have statements added, modified and/or removed.

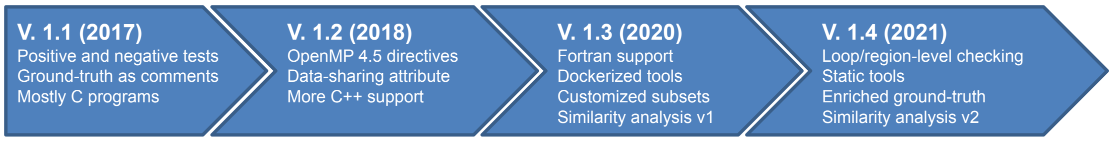

Fig. 1: History of DataRaceBench

```
#pragma omp parallel for
for (i=0;i< len -1 ;i++)
  a[i]=a[i+1]+1;
```

(a) DRB001-antidep1-orig-yes.c

```
#pragma omp parallel for
for (i=0;i<len-1;i++)
  a[i+1]=a[i]+1;
```

(b) DRB029-truedep1-orig-yes.c

```
#pragma omp parallel for private (i)
  for (i=0;i<len;i++)
    x=i;
```

(c) DRB009-lastprivatemissing-orig-yes.c

Fig. 2: Examples of DRB code regions

- Type-4: Code fragments are syntactically dissimilar but with the same functionality.

We leverage the definitions of similarity types to further define a new type of similarity used for the study of similarity analysis for DRB. The Type-5 similarity will be based on the Type-3 similarity but the code regions should share the same computational traits in the source codes. In this paper, we define the computational traits as the properties of an OpenMP code region, including its OpenMP directive and the code block following the OpenMP directive. Two similar code regions should carry similar OpenMP directives and clauses and have similar syntactic code representations, in a loop or in a statement, following the given OpenMP pragma. Ultimately, we would like the computational traits to include more detailed information such as data dependency information. Therefore different code regions should have either no loop-carried dependency or the same type of loop-carried dependency (true, anti, or control dependency).

C. *Previous Method: Cosine Similarity Analysis*

A Cosine Similarity analysis has been applied to DRB [3] [4]. The analysis relies on a set of manually picked features to represent the source code regions. The collected information has: first, the static information includes OpenMP directives and clauses used in source code (Table I: E0 through E122); second, statistics of the code region wrapped by OpenMP directives. (Table I: E123 through E134); and lastly, the dynamic information contains the data race analysis result (Table I: E135 through E140). A Clang-based plugin tool, the OpenMP Extractor [3], was applied to collect these features from Clang AST. The feature vector is defined using the following formula:

$$\vec{A} = (Directive, Clause, \{Extracted\ source\ info.\ list\},$$
$$\{Tool\ result\ list\}, Ground\ Truth)$$

Feature	Value Range	Encoding Fields	Description
Directive	[0,1]	[E0-E86]	87 directives are flattened into the first 87 elements in the vector. The existing directive's value is set to 1, else 0.
Clause	[0,1]	[E87-E122]	The next 36 integer elements are 36 flattened clauses. If the value is 1, the test case contains that clause.
AddOp	[0,n]	E123	Number of Add operators.
SubOp	[0,n]	E124	Number of Subtract operators.
MulOp	[0,n]	E125	Number of Multiply operators.
DivOp	[0,n]	E126	Number of divide operators.
CompOp	[0,n]	E127	Number of compare operators.
BitOp	[0,n]	E128	Number of bit operators.
LogicOp	[0,n]	E129	Number of logic operators.
AssignOp	[0,n]	E130	Number of assign operators.
CombOp	[0,n]	E131	Number of combined operators (+=, -=, etc.).
ConstOp	[0,n]	E132	Number of constant integer and floating values.
VariableRef	[0,n]	E133	Number of distinct variable names.
totalVarRef	[0,n]	E134	Total number of variable references.
Intel	[-2,1]	E135	Data race result by the tool. -2 represents time out. -1 represents the segmentation fault. 0 represents no data race. 1 represents the data race.
ROMP	[-2,1]	E136	
Tsan	[-2,1]	E137	
Coderrect	[-2,1]	E138	
LLOV	[-2,1]	E139	
Ground Truth	[0,1]	E140	Ground Truth, whether a loop has a data race or not.

TABLE I: Feature vector's definition and encoding methods

For the code regions, the Cosine Similarity analysis mainly considers OpenMP parallel loops under the OMPLoopDirective AST node and basic blocks under the OMPParallelDirective AST node. Some more additional code regions are also considered. For example the `DRB074-flush-orig-yes.c`, that has only `omp parallel reduction` in the source code without involving a loop.

The Cosine Similarity metric calculates the cosine of the angle between two non-zero vectors of an inner-product space and is more suitable for high-dimensional vectors. The Cosine Similarity is calculated by:

$$\cos(\theta) = \frac{\vec{A} \cdot \vec{B}}{|\vec{A}||\vec{B}|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \quad (1)$$

The similarity, range from [-1,1], is used to determine the similarity of two vectors. If two vectors are more similar, the cosine similarity will be closer to 1, and the degree for two vectors will be closer to 0°. Two vectors with a similarity value that is farther away from 1 are considered more dissimilar. There are only five fields (E135 through E139) in the feature vector which allow negative values. Therefore, it is unlikely to see negative similarity, representing two vectors pointing to opposite directions, with the feature vector used by DRB.

IV. Transformer-Based Similarity Analysis

In this section, we describe our approach to leverage transformer-based pretrained code encoders to compute similarity scores of OpenMP code regions. First, we introduce transformer models and transformer-based similarity analysis. We then describe the challenges and the methodology we use to investigate the applicability of CodeBERT to our purpose.

A. Transformers

Transformers are deep learning models that adopt the mechanism of self-attention to solve tasks in the fields of natural language processing and computer vision. Since its introduction in 2017 [21], Transformers have replaced recurrent neural networks (RNN) for language modeling tasks. Transformers have also shown the ability to outperform convolutional neural networks (CNN) for image processing [22], [23] and text-to-image [24].

The Transformer architecture uses attention [12], a deep-learning mechanism, where the dot-product of keys and queries measures the *attention* that should be given to a value. Initially the attention mechanism, which is the base of the transformer architecture, was used as part of RNN architectures. The nature of the attention mechanism makes transformers a set-to-set architecture. Usually, positional embedding are added to each token's embedding for sequence-to-sequence transformers. In the original transformers [12], both an encoder stack and a decoder stack are used to produce the output. However, the architecture can be split into encoder-only and decoder-only transformers such as: Bidirectional Encoder Representations from Transformers (BERT) [13], and Generative Pre-trained Transformer (GPT) [25].

B. Transformer-based Similarity Analysis

Due to the similarity between natural languages and programming languages, transformers have been extended to solve various programming language processing (PLP) tasks. A recent paper [19] summarizes typical PLP tasks, model architectures, compiler tools and datasets in order to facilitate the assembling of machine learning pipelines to solve a given programming language processing task.

We identify two relevant PLP tasks for our similarity analysis of OpenMP code regions: clone-detection [26] and code-search [27]. Based on the prior study [19], we decide to leverage CodeBERT [28] a pretrained encoder-only transformers for our similarity analysis. We select CodeBERT for two main reasons: tokenization is simple (without additional compiler tools) and its documentation and tutorials are extensive. Fine-tuning a pretrained ML model for a new task is a form of transfer-learning.

As shown in Fig. 3(a), CodeBERT is pretrained with positive and negative pairs of text-code and code-code from CodeSearchNet [27]. During this pretraining, CodeBERT is given tokenized sequences, prefixed by a classifier (CLS) token and separated by a separator (SEP) token, and returns a sequence of embeddings. These embeddings are used by two training tasks: (1) training a multi-layer perceptron (MLP) to predict whether or not the sequences are related using the embedding of the CLS token, and (2) masked language modeling (MLM) provides a self-supervised task for the other embeddings.

To generate similarity analysis scores, one can use CodeBERT to process two input code sequences separately and generate two embeddings. It is a common practice to fine-tune a pre-trained model with additional datasets or customized tasks so better results can be obtained. Finally, the embeddings can be used to compute a cosine similarity as the final similarity score, as shown in Fig. 3(c).

C. Challenges

Our goal in this paper is to repurpose CodeBERT, a pre-trained programming language encoder-only transformer, to evaluate similarities between a pair of C/C++ OpenMP code regions. This process of using a pretrained model for another objective is often called transfer learning.

CodeBERT is trained on Python, Java, JavaScript, PHP, Ruby, and Go to predict *semantic* similarities between pairs of text-code and code-code. We believe that this repurposing is possible because (1) finding *semantic* similarities require some understanding of the syntax, and (2) DRB's C/C++ kernels include statements (loops, conditionals, expressions, and code blocks) whose syntax is similar to those of Java, JavaScript, and PHP.

However, transfer learning is sensitive to changes of the data distributions in the dataset and modifications to the training objectives. There are two challenges to the applicability of vanilla CodeBERT to OpenMP kernel similarity: (1) improving the ability of CodeBERT to process C/C++, and (2) providing CodeBERT with some understanding of OpenMP directives. A common solution to these challenges is to add fine-tune steps using extra datasets presenting the additional code features which are not captured in the pre-trained model's training dataset.

Another challenge is that for our goal of finding similar OpenMP code regions, it is difficult to define a ground truth for both training and evaluation. The reason is that the similarity score we are interested in should be a real number and how similar two regions are can be subjective. To address this challenge, one can apply manual investigation within the context of concrete tasks such as identifying duplicated OpenMP code patterns or deciding if a new region should be added. This method of validating similarity results can be time and labor-consuming. Another solution is to leverage a similar third party tool, such as SourcererCC [29], that can provide quantitative referencing reports. Ultimately, a quantitative and standardized metric, like the BLEU score used in natural

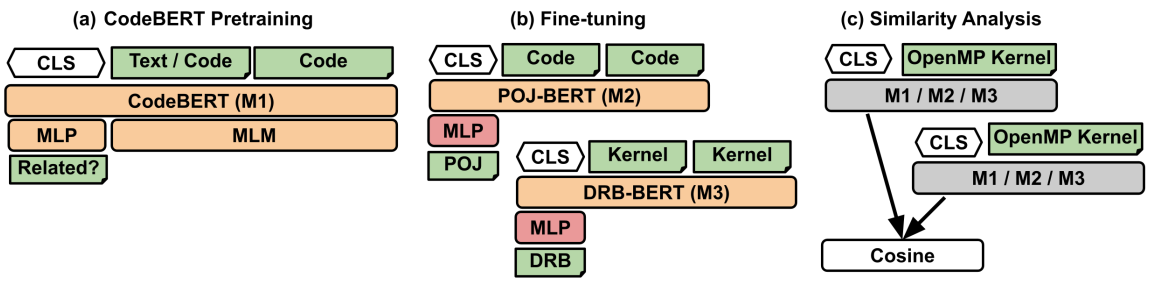

Fig. 3: Pipelines from left to right: (a) CodeBERT is a pretrained model (called M1 in this paper) to classify if a pair of text-code and code-code is related or not, (b) fine-tuning CodeBERT using pairs of C/C++ code from POJ or pairs of OpenMP kernels from DRB, resulting two models named M2 and M3 respectively, and (c) similarity analysis using either CodeBERT, or one of the two fine-tuned models.

language translation, should be defined to determine the trusted similarity score.

Finally, transformers have limitations for the length of input sequences, such as the 512-token limitation in CodeBERT. Real code regions may have longer sequences to be analyzed. Fortunately, DRB is designed to minimize code size while representing OpenMP code patterns. We have found that the length of the code regions of DRB are mostly loops with 50-175 tokens. For longer regions, we decided to drop code regions with more than 400 input tokens due to the memory limitation in the testing platform. These 400 tokens were tokenized by the tokenizer in sequential order of how the code is implemented. This is the area that we can look more into for compressing the input information to be encoded.

D. Fine-Tuning Pre-Trained Transformer Model

As shown in Fig. 3(b), we fine-tune original CodeBERT (called M1 in this paper) to make it more applicable to the similarity analysis of C/C++ OpenMP code regions. Two datasets are used for fine tuning CodeBERT. One is POJ-104 [30] and the other is DRB. The generated models are called M2 and M3 respectively as shown in Table II. POJ-104 provides a large number of C/C++ code to learn more about these languages while DRB includes some C/C++ but also OpenMP directives. We can then see how having C/C++ low-level code as a training set and adding additional OpenMP information would help to encode the embeddings.

The pipeline is used to identify either: pairs of POJ-104 codes found to be similar to generate M2, or pairs of DRB Kernel found to be similar to generate M3. The positive and negative pairs are constructed using near duplicate analysis (NDA). We use SourcererCC, a hand-crafted code clone detector, to compute syntactic similarities for pairs of codes. In both cases, any pair of code fragments with SourcererCC similarity score higher than 80% is marked as positive, whereas the rest are marked as negative.

V. Experiments

Table III lists the details of the experiment platform and the required software packages. The prerequisites to the experiments include collecting the baseline CodeBERT model, extracting the OpenMP code regions from DRB C/C++ microbenchmarks, and applying similarity labels, with SourcererCC code-clone detection analysis, to all pairs of code within POJ-104 and DRB extracted OpenMP code regions.

Label	Model Name	Description
M1	Vanilla CodBERT	Original pre-trained CodeBERT
M2	POJ-CodeBERT-Syntactic	M1 fine-tuned with POJ syntactic labels
M3	DRB-CodeBRT-Syntactic	M2 fine-tuned with DRB syntactic labels

TABLE II: CodeBERT Variants

	Dell Precision T7920 Workstation
CPU	Intel Xeon Gold 6238 CPU 2.10GHz
CPU Cores	2 sockets $\times$ 22 physical cores
Main Memory	128 GB (2x64GB) DDR4 ECC Memory
GPU	Nvidia Quadro RTX 6000
Device Memory	24 GB
OS	Red Hat Enterprise Linux 8.6 (Ootpa)
ML Configuration	Python=3.6.8, Pytorch=1.4.0, Transformers=2.5.0
Datasets	POJ-104, DataRaceBench 1.4

TABLE III: Software and Hardware Configurations

A. Training Dataset Construction

We label both POJ-104 and DRB extracted OpenMP dataset for the fine-tuning step to capture the features with syntactic similarity. SourcererCC is used to determine the Type-3 similarity for two given code fragments in the dataset (POJ-104 or DRB). An 80% similarity threshold is specified in SourcererCC to determine if the pair of code fragments is considered similar. For POJ-104, there are 64,278 pairs of codes, out of 1,352,026,000 ($\sum_{n=1}^{52000} n$) possible distinct pairs, considered similar. We further filter out the source codes with more than 400 tokens due to the system and CodeBERT constraints mentioned previously. We construct an overall 67,816 data samples including dissimilar pairs that are randomly generated. Similar pairs in the dataset are marked 1 in the label, whereas the rest are marked 0. We are using this to train the first fine-tuned CodeBERT (M2) and classification task of whether a pair of codes is syntactically similar or not.

For DRB dataset, we extend support for the Clang-based tool, OpenMP Extractor [3], to extract 658 OpenMP regions from DRB C/C++ microbenchmarks. 624 source code regions are left after filtering out code regions with more than 400 tokens. In the similarity labeling process, 86 pairs of codes, out of 195,000 ($\sum_{n=1}^{624} n$) pairs, are considered similar by SourcererCC.

B. *Fine-Tuning for POJ-CodeBERT-Syntactic (M2) and DRB-CodeBERT-Syntactic (M3)*

In this experiment, the CodeBERT base model from the HuggingFace[1] Transformer Package is applied to initialize the CobeBERT model to be fine-tuned. We also reference the BigCloneBench [31] model structure to have the CodeBERT with classification downstream tasks implemented with multi-layer perceptron (MLP).

The POJ-CodeBERT-Syntactic (M2) is generated by fine-tuning CodeBERT using POJ-104 with C/C++ syntax. We apply a split of 70% on training, 10% validating, and 20% testing on total of 67,816 data samples, randomly generated from POJ-104 labeled dataset, to run two epochs of training with block size of 400 and learning rate of $5\mathrm{e}^{-5}$. The training takes about 40 minutes for each epoch. During the training, we stored the best F1 score model from the pipeline to prepare for testing. In addition, the CodeBERT model weights are stored in order to generate embeddings of the code regions.

A second fine-tune, to generate DRB-CodeBERT-Syntactic (M3), is then applied using DRB with OpenMP Syntax. We reused the M2 that was saved and loaded it into CodeBERT. The training step for the second fine-tune uses 697 data samples that were randomly generated from the DRB labeled dataset. The training takes about 1.5 minutes per epoch. The data split, epoch number, block size and learning rate are the same as the training in the first fine-tune training. The pipeline for this step is shown in Fig. 3(b).

The results of the training are shown in Table IV, where we use the testing to receive the F1 score, Recall, and Precision. This result shows that our models perform well to understand the syntactic patterns to predict whether or not a pair of codes are similar. The small DRB dataset used to fine-tune M3 model is likely to lead to overfitted results in testing. As we are only using the encoder from the models to generate the embeddings to obtain the similarity score, these models are sufficient to demonstrate the learning for the models.

	M2	M3
F1	0.9490	1.0000
Recall	0.9563	1.0000
Precision	0.9418	1.0000

TABLE IV: Result of the Fine-Tune Training

C. *Feature-based Embedding Similarity*

In feature-based embedding, we tokenize the codes from DRB and pass the tokenized token stream into different variants of CodeBERT models to generate embeddings for each code. We can then match the embeddings generated with different models, representing the same source code, and apply cosine similarity to determine the similarity.

[1]https://huggingface.co/

D. *Comparison & Discussion*

To identify similar codes among the OpenMP code regions in DRB, we exploit heatmaps to identify the highly similar codes in the collected OpenMP regions. Four groups of heatmaps, shown in Fig. 4, of similarity among all pairs of extracted OpenMP regions in DRB are generated by applying the three variants of the transformer-based similarity analysis, using the models M1 through M3, and the cosine similarity model from the previous work. Fig. 4 also presents both heatmaps of raw similarity data (top row) and the standardized data (bottom row). The standardized data indicates that although the vanilla CodeBERT(M1) model reports high similarity scores for all pairs of code regions in DRB, but it is still capable to capture differences among the collected OpenMP code regions. The heatmap from the cosine similarity analysis represents the results from the previous work: 54% of the pairs are highly similar (similarity higher than 0.87), 44% are moderately similar (similarity between 0.5 and 0.87) and only 2.9% are distinct (similarity between -1 and 0.5). The heatmap generated using the Vanilla CodeBERT (M1) shows high similarity for most of the code pairs. With POJ-CodeBERT (M2), the generated heatmap reveals variances in similarity. This indicates that the fine-tuning with the POJ-104 dataset does improve the analysis by recognizing C/C++ code patterns. There are only a very few dissimilar pairs shown in the heatmap with the result generated by DRB-CodeBERT (M3). The model is based on the M2 model and fine-tuned with the DRB dataset. The purpose of the second fine-tuning is to provide OpenMP specific patterns to the training. However, the small number of code regions collected in the DRB does not seem to provide adequate code patterns for training data. The model has seen the whole dataset during its training session and might not be ideal to apply any meaningful task with the same dataset.

Due to the difficulty of defining the ground truth of the real similarity scores within DRB, we leverage again the SourcererCC tool to provide a similarity report as a reference in the comparison. The SourcererCC report has significant differences compared to the results by the transformer-based approaches. With a low similarity threshold of 10%, we applied SourcererCC to generate the list of pairs of codes with similarity higher than the threshold value. Only 196 pairs from all possible pairs are reported by SourcererCC with at least 10% of similarity.

We manually inspect the similarity reports by applying a Top-K analysis to select the top three pairs of code regions that have the highest similarity reported by SourcererCC and our three transformer-based models (Table V). The selected top three pairs do not include highly similar code pairs that are known to the DRB developers, such as these pairs differing by only fixed or varying arrays. Similarity scores, by cosine similarity and transformer-based models, are provided in the table as references. With the manual inspection, DRB001, DRB002, DRB026, DRB029, and DRB030 all have very similar code regions as shown in Fig. 2. Mostly, they share similar code patterns but with different dependence types. Other pairs like (DRB028, DRB035) and (DRB013, DRB104) are also reported to have highly similar code patterns. DRB028 and DRB035 differ by only the order of two statements. But the difference causes different numbers of dependence pairs. DRB013 and DRB104 differ by only one OpenMP barrier directive. All these pairs should be kept since they represent subtle differences related to OpenMP. High similarity is reported by all selected similarity analysis for the selected pairs in Table V.

To evaluate the effectiveness of the models in determining the similarity for new OpenMP programs, we manually select two OpenMP parallel regions from the NAS parallel benchmark [32] and eight code regions of the Rodinia benchmark suite [33] (from bfs, find_ellipse, hotspot, kmean_clustering, lud, nn and pre_euler3d) as the unseen OpenMP programs for testing. All the selected codes have less than 10% of similarity, checked by SourcererCC, compared to all extracted code regions from DRB. Each unseen OpenMP region is compared against every extracted OpenMP code region from DRB using the three models aforementioned. Four of the ten OpenMP code regions from the collected unseen OpenMP dataset are eliminated from the experiment due to the 400-token constraint in the experiment platform. For each selected transformer-based model, we list the three extracted code regions from the DRB that are identified to have the highest similarity compared to the unseen OpenMP regions. Table VI lists all code regions picked by the three transformer-based models. The M1 model picks exactly the same list for four out of the six selected unseen OpenMP codes. The selected codes from the list are small code regions with OpenMP pragma and a single statement. Code regions picked by the M2 model tend to have very simple loop structures. Instead, the M3 model, which is fine-tuned with the DRB dataset, picks code regions with larger loop structures, with single or nested loops. The codes picked by M3 are still considered dissimilar after manual inspection but seem to present a closer loop structure that is present in the unseen code.

The ranges of top one scores for the six code regions in Table VI are: 0.13 to 0.3 from M1, 0.72 to 0.9 from M2, and 0.96 to 1 from M3. Given a task to consult these three models to decide if these unseen code regions should be included into DRB collection, the M1 model suggests all the six regions as candidates to DRB collection due to very low similarity found in existing benchmark suite. Whereas all six code regions should not be considered to be added into DRB collection based on the M3 model. Scores reported by M2 model sit in the middle of results from the other two but its answer to the task should be closer to the answer from M3 model. Given all six unseen codes are with low similarity reported by SourcererCC. M1 model, the vanilla CodeBERT, provides better suggestions for the unseen OpenMP codes selected in this experiment.

A manual inspection is applied, following the definition of the proposed Type-5 similarity, as a more trustworthy ground truth to determine if the six unseen OpenMP codes should be included into DRB collection. The result is listed in the following:

- NPB-IS-1 is likely to be added due to unseen OpenMP clause: `schedule(dynamic)`
- NPB-IS-2 will not be considered due to similar pattern in DRB104-nowait-barrier-orig-no.c.
- rodinia-bfs-1 is likely to be added with a new pattern that has the if statement inside the parallel loop body.
- rodinia-lud_omp-1 will not be selected due to many existing code regions with `omp simd`.
- rodinia-nn_openmp-1 will not be selected for seen code patterns with both `shared` and `private` OpenMP clauses in DRB181-SmithWaterman-yes.c.
- rodinia-hotspot_openmp-1 is likely to be included due to a more complex statement in SIMD loop body, and calculation appearing in the loop statement's test expression.

VI. Related Work

There are many approaches to similarity analysis. Checking checksum of digital content is simple and effective to identify the exact duplication but it is insufficient to identify partial duplication. Fingerprints of digital contents [34] can be used to identify partial duplication and applied for similarity analysis. Another effective approach, implemented for SoucererCC, tokenizes the code blocks of the input code and builds a partial index for the subset of the tokens in a block. The analysis iterates through the code blocks and retrieves their candidate clone blocks from the index to inspect if the similarity score is higher than a given threshold [29]. However, this approach is not suitable for the the study with DRB as it reports only if the similarity passing the threshold and doesn't provide the exact similarity score.

Machine learning has become a popular approach to code similarity analysis. There are various existing similarity analysis systems based on RNN or transformer models. The RNN-based models vary in the code representation structures, code2vec [35] and code2seq [36] utilize abstract syntax trees, Neural Code Comprehension [37] exploits LLVM intemediate representation to generate conteXtual flow graph (XFG); Aroma [38] relies on the construction of simplified parsed tree for the source code; MISIM [39] uses context-aware semantics structure (CASS) as the representation. The transformer-based models include CodeBERT [28] and AlphaCode [18]. Compared to the existing similarity analysis using machine learning, our similarity is a transformer-based approach but with focus to determine similarity among OpenMP code regions extracted from DRB.

Large scale dataset, CodeNet [40], uses similarity analysis, through SourcererCC, to identify near-duplicated codes to avoid redundant collection. Meanwhile, CodeNet also picks source code similarity analysis as a task to demonstrate its usefulness as a dataset for machine learning applied to code-to-code and code-to-text learning.

Model	Rank	Pair		DB Score	M1 Score	M2 Score	M3 Score
DB	1	DRB001-1	DRB030-1	1.0	0.999934	0.999985	0.999992
	2	DRB001-1	DRB029-1	1.0	0.999934	0.999985	0.999992
	3	DRB028-1	DRB035-1	1.0	0.999805	0.999541	0.999942
M1	1	DRB013-1	DRB104-1	0.996063	1.0	1.0	1.0
	2	DRB001-1	DRB026-1	0.981769	1.0	1.0	1.0
	3	DRB002-1	DRB026-1	0.981769	1.0	1.0	1.0
M2	1	DRB013-1	DRB104-1	0.996063	1.0	1.0	1.0
	2	DRB002-1	DRB026-1	0.981769	1.0	1.0	1.0
	3	DRB001-1	DRB026-1	0.981769	1.0	1.0	1.0
M3	1	DRB013-1	DRB104-1	0.996063	1.0	1.0	1.0
	2	DRB002-1	DRB026-1	0.981769	1.0	1.0	1.0
	3	DRB001-1	DRB026-1	0.981769	1.0	1.0	1.0

TABLE V: Top-3 similar pairs and corresponding similarity scores in DRB reported by the similarity analysis models

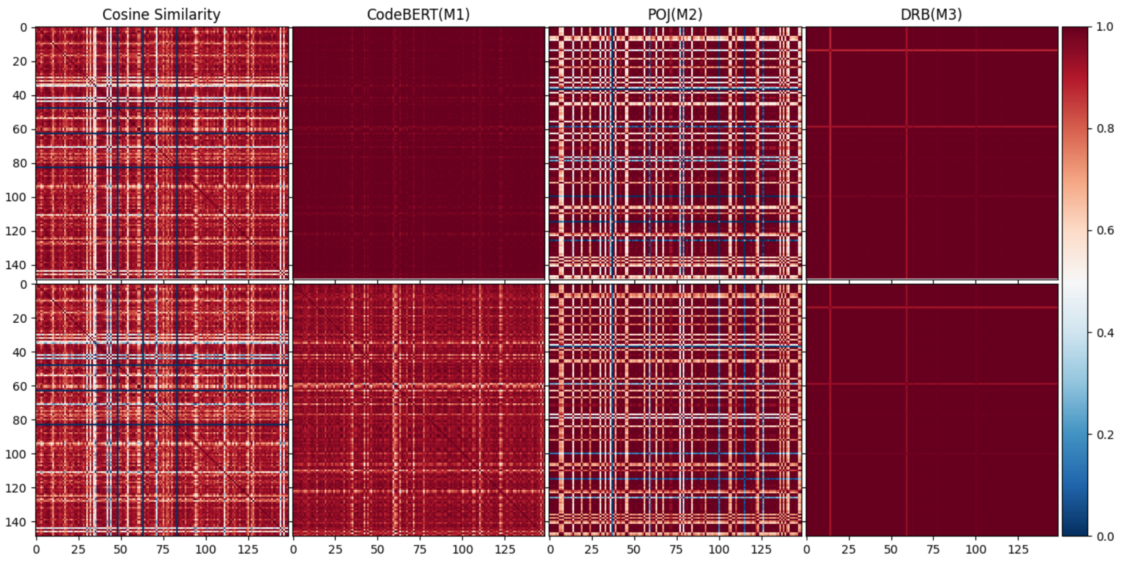

Fig. 4: Heatmaps of similarity for DataRaceBench using cosine similarity, M1, M2 and M3 models. Top row: raw similarity data; bottom row: standardized data

Unseen OMP code	Top1			Top2			Top3		
	M1	M2	M3	M1	M2	M3	M1	M2	M3
NPB-IS-1	DRB143-2	DRB086-1	DRB092-2	DRB142-2	DRB083-1	DRB084-2	DRB143-1	DRB082-1	DRB091-2
NPB-IS-2	DRB143-2	DRB083-1	DRB137-1	DRB142-2	DRB082-1	DRB138-1	DRB142-1	DRB088-1	DRB095-1
rodinia-bfs-1	DRB143-2	DRB094-1	DRB181-1	DRB142-2	DRB068-1	DRB159-3	DRB143-1	DRB088-1	DRB058-2
rodinia-lud_omp-1	DRB143-2	DRB083-1	DRB170-1	DRB142-2	DRB088-1	DRB041-9	DRB143-1	DRB086-1	DRB141-2
rodinia-nn_openmp-1	DRB094-2	DRB041-31	DRB041-20	DRB143-2	DRB094-1	DRB041-8	DRB142-2	DRB147-1	DRB041-25
rodinia-hotspot_openmp-1	DRB094-2	DRB086-1	DRB041-25	DRB096-2	DRB083-1	DRB041-8	DRB142-2	DRB082-1	DRB041-15

TABLE VI: Top-K Similar OpenMP Regions in DRB for unseen OpenMP codes

VII. Conclusion

In this paper, we present our initial work to leverage deep-learning transformers to conduct similarity analysis of DRB. Our experiments show that fine-tuning pretrained transformers are essential to adapt them for the domain of parallel computing due to new syntax and semantics of parallel language constructs such as those of OpenMP. Although transformers can alleviate the burden of manually picking code features, labeled datasets with suitable training tasks are hard to find. Another major limitation is that the input sequences of the transformer models have to fit into a small length, such as 512. This limits their applicability to real applications using large code regions. It is challenging to evaluate the effectiveness of a similarity analysis without a quantitative and standard metric. For Type-3 and the Type-5 similarity proposed in this paper, defining the ground truth and developing a standard metric for evaluation can greatly assist the future work in developing similarity analysis.

Future work will experiment with more methods of fine-tuning CodeBERT using training datasets with more OpenMP semantics. We will also use the analysis pipelines to scan more open source OpenMP codes, including those written in Fortran, to find new code patterns to be added into DRB. Second, we will also look into how to enable transformers to analyze large code regions. And finally, we will survey and design quantitative metric for the similarity analysis.

References

[1] C. Liao, P.-H. Lin, J. Asplund, M. Schordan, and I. Karlin, "Dataracebench: a benchmark suite for systematic evaluation of data race detection tools," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–14.

[2] C. Liao, P.-H. Lin, M. Schordan, and I. Karlin, "A semantics-driven approach to improving dataracebench's openmp standard coverage," in *International Workshop on OpenMP*. Springer, 2018, pp. 189–202.

[3] G. Verma, Y. Shi, C. Liao, B. Chapman, and Y. Yan, "Enhancing dataracebench for evaluating data race detection tools," in *2020 IEEE/ACM 4th International Workshop on Software Correctness for HPC Applications (Correctness)*. IEEE, 2020, pp. 20–30.

[4] P.-H. Lin and C. Liao, "High-precision evaluation of both static and dynamic tools using dataracebench," in *2021 IEEE/ACM 5th International Workshop on Software Correctness for HPC Applications (Correctness)*. IEEE, 2021, pp. 1–8.

[5] S. Thayer, G. L. Gopalakrishnan, I. Briggs, M. Bentley, D. H. Ahn, I. Laguna, and G. L. Lee, "Archergear: data race equivalencing for expeditious hpc debugging," in *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2020, pp. 425–426.

[6] S. Atzeni, G. Gopalakrishnan, Z. Rakamaric, I. Laguna, G. L. Lee, and D. H. Ahn, "Sword: A bounded memory-overhead detector of openmp data races in production runs," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2018, pp. 845–854.

[7] A. Schmitz, J. Protze, L. Yu, S. Schwitanski, and M. S. Müller, "Dataraceonaccelerator–a micro-benchmark suite for evaluating correctness tools targeting accelerators," in *European Conference on Parallel Processing*. Springer, 2019, pp. 245–257.

[8] U. Bora, S. Das, P. Kureja, S. Joshi, R. Upadrasta, and S. Rajopadhye, "Llov: A fast static data-race checker for openmp programs," *arXiv preprint arXiv:1912.12189*, 2019.

[9] Y. Gu and J. Mellor-Crummey, "Dynamic data race detection for openmp programs," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 767–778.

[10] J. Gao, X. Yang, Y. Jiang, H. Liu, W. Ying, and X. Zhang, "Jbench: a dataset of data races for concurrency testing," in *Proceedings of the 15th International Conference on Mining Software Repositories*, 2018, pp. 6–9.

[11] M. Jasper, M. Mues, M. Schlüter, B. Steffen, and F. Howar, "Rers 2018: Ctl, ltl, and reachability," in *International Symposium on Leveraging Applications of Formal Methods*. Springer, 2018, pp. 433–447.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[13] 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.

[14] L. Floridi and M. Chiriatti, "Gpt-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, no. 4, pp. 681–694, 2020.

[15] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[16] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. B. Clement, D. Drain, D. Jiang, D. Tang, G. Li, L. Zhou, L. Shou, L. Zhou, M. Tufano, M. Gong, M. Zhou, N. Duan, N. Sundaresan, S. K. Deng, S. Fu, and S. Liu, "Codexglue: A machine learning benchmark dataset for code understanding and generation," *CoRR*, vol. abs/2102.04664, 2021.

[17] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[18] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. D. Lago *et al.*, "Competition-level code generation with alphacode," *arXiv preprint arXiv:2203.07814*, 2022.

[19] P. Flynn, T. Vanderbruggen, C. Liao, P.-H. Lin, M. Emani, and X. Shen, "Finding reusable machine learning components to build programming language processing pipelines," *2nd International Workshop on Software Architecture and Machine Learning*, 2022. [Online]. Available: https://arxiv.org/abs/2208.05596

[20] H. Sajnani, *Large-scale code clone detection*. University of California, Irvine, 2016.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[22] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2020. [Online]. Available: https://arxiv.org/abs/2010.11929

[23] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," 2020. [Online]. Available: https://arxiv.org/abs/2005.12872

[24] J. Yu, Y. Xu, J. Y. Koh, T. Luong, G. Baid, Z. Wang, V. Vasudevan, A. Ku, Y. Yang, B. K. Ayan, B. Hutchinson, W. Han, Z. Parekh, X. Li, H. Zhang, J. Baldridge, and Y. Wu, "Scaling autoregressive models for content-rich text-to-image generation," 2022. [Online]. Available: https://arxiv.org/abs/2206.10789

[25] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[26] 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*, 2014, pp. 476–480.

[27] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "Codesearchnet challenge: Evaluating the state of semantic code search," 2019. [Online]. Available: https://arxiv.org/abs/1909.09436

[28] 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.

[29] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "Sourcerercc: Scaling code clone detection to big-code," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, 2016, pp. 1157–1168.

[30] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, "Convolutional neural networks over tree structures for programming language processing," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[31] 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.

[32] D. H. Bailey, E. Barszcz, L. Dagum, and H. D. Simon, "Nas parallel benchmark results," *IEEE Parallel & Distributed Technology: Systems & Applications*, vol. 1, no. 1, pp. 43–51, 1993.

[33] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE international symposium on workload characterization (IISWC)*. Ieee, 2009, pp. 44–54.

[34] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: local algorithms for document fingerprinting," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 2003, pp. 76–85.

[35] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning distributed representations of code," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–29, 2019.

[36] U. Alon, S. Brody, O. Levy, and E. Yahav, "code2seq: Generating sequences from structured representations of code," *arXiv preprint arXiv:1808.01400*, 2018.

[37] T. Ben-Nun, A. S. Jakobovits, and T. Hoefler, "Neural code comprehension: A learnable representation of code semantics," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[38] S. Luan, D. Yang, C. Barnaby, K. Sen, and S. Chandra, "Aroma: Code recommendation via structural code search," *Proceedings of the ACM on Programming Languages*, vol. 3, no. OOPSLA, pp. 1–28, 2019.

[39] F. Ye, S. Zhou, A. Venkat, R. Marcus, N. Tatbul, J. J. Tithi, N. Hasabnis, P. Petersen, T. Mattson, T. Kraska *et al.*, "Misim: A neural code semantics similarity system using the context-aware semantics structure," *arXiv preprint arXiv:2006.05265*, 2020.

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