# CCT-Code: Cross-Consistency Training for Multilingual Clone Detection and Code Search

Nikita Sorokin[1], Dmitry Abulkhanov[1], Sergey Nikolenko[2], and Valentin Malykh[1]

[1]Huawei Noah's Ark Lab, valentin.malykh@huawei.com,
[2]St. Petersburg Department of the Steklov Institute of Mathematics, sergey@logic.pdmi.ras.ru

May 22, 2023

## Abstract

We consider the clone detection and information retrieval problems for source code, well-known tasks important for any programming language. Although it is also an important and interesting problem to find code snippets that operate identically but are written in different programming languages, to the best of our knowledge multilingual clone detection has not been studied in literature. In this work, we formulate the multilingual clone detection problem and present XCD, a new benchmark dataset produced from the CodeForces submissions dataset. Moreover, we present a novel training procedure, called cross-consistency training (CCT), that we apply to train language models on source code in different programming languages. The resulting CCT-LM model, initialized with GraphCodeBERT and fine-tuned with CCT, achieves new state of the art, outperforming existing approaches on the POJ-104 clone detection benchmark with 95.67% MAP and AdvTest code search benchmark with 47.18% MRR; it also shows the best results on the newly created multilingual clone detection benchmark XCD across all programming languages.

## 1 Introduction

The clone detection problem stems from an important need arising in software development practice: to identify code with the same behaviour and effective output; this is useful, e.g., for code unification and refactoring and control of side effects. In automated language processing, Mou et al. [34] formulated the clone detection task for C/C++ source code, and later the problem was extended to other programming languages. The natural next step for this problem is to

detect the same behaviour for code across different programming languages.[1] In this work, we formulate the multilingual clone detection task, collect a dataset for it, and establish reasonable baselines on this dataset.

Various approaches for the clone detection task have been developed, starting from algorithmic methods [5, 24] and continuing with machine learning models [16, 29, 42]. Most machine learning approaches are based on learning the representations (embeddings) of code snippets; this approach allows to find duplicate code snippets by similarity between their embeddings in the latent space. The performance of such systems critically depends on the quality of embeddings. In this work, we present a novel training technique and pipeline called CCT for language models that allows them to embed code snippets in an efficient way. We demonstrate this by achieving state of the art on a preexisting clone detection dataset POJ-104 [34] and on the newly formulated multilingual clone detection dataset XCD.

Interestingly, we also show that the CCT technique allows a model to produce representations useful for code search as well. Code search, as formulated by Lu et al. [31], is the task where the query is a text description and the possible documents are code snippets.

Thus, the main contributions of our work are as follows:

(1) we present a pretraining method CCT that allows a model to align code snippets in different programming languages;

(2) we present a novel multilingual clone detection dataset XCD based on *Code-Forces* submissions;

(3) we present new state of the art results obtained by a language model CCT-LM trained with CCT on the clone detection datasets POJ-104 and XCD[2];

(4) we present state of the art results of CCT-LM for code search on the *AdvTest* dataset.

The paper is organized as follows. Section 2 surveys related work on data and approaches for source code analysis and multilingual natural language processing. Section 3 discusses code search datasets and introduces XCD. Section 4 describes the CCT pretraining approach. Section 5 shows our results on XCD and preexisting datasets, Section 6 analyzes them and presents an ablation study, Section 6.4 discusses the limitations of our work, and Section 7 concludes the paper.

---

[1]For an illustrative example, we refer here to the `http://helloworldcollection.de/` website that contains "Hello, world!" snippets in 603 programming languages.

[2]We will release the XCD dataset and the source code for CCT and CCT-LM upon acceptance.

## 2 Related Work

Our methods are inspired by natural language processing, thus related work includes both pure NLP and programming language processing.

### 2.1 Datasets

Husain et al. [21] presented the *CodeSearchNet* dataset constructed from a *GitHub* dump where the authors split method bodies into the code itself and a description. This dataset contains 2 million code snippet-description pairs in 6 programming languages, including *Python*. This dataset was partially used by Hasan et al. [19] who combined *CodeSearchNet* and three other datasets into a larger one. From *CodeSearchNet* they used the *Java* part and *Python* part translated automatically into *Java*. The resulting dataset contains 4 million code snippet-description pairs. *CodeXGLUE* presented by Lu et al. [31] is a machine learning benchmark collection of datasets for code understanding and generation tasks, which includes a modification of *CodeSearchNet*. *CodeXGLUE* provides a benchmark for various code-to-code, code-to-text, text-to-code tasks, including code search. This benchmark includes code in 10 programming languages.

There are two main datasets for clone detection: POJ-104 [34] and *BigCloneBench* [45]. POJ-104 represents a comparatively small corpus of C++ solutions from a student judging system. *BigCloneBench* is larger and includes automatically mined data in several programming languages.

### 2.2 Code Search

Early works on code search mostly relied on classic information retrieval [4,8–10]. Brandt et al. [8], Barzilay et al. [6], and Ponzanelli et al. [35] utilized existing industrial Web search engines. Gu et al. [17] used modern dense vector representations for information retrieval, training two recurrent neural networks to represent source code and text respectively. Feng et al. [13] presented a language model-based approach to produce these representations. Gotmare et al. [16] used three Transformer-based models, two as encoders and one as a classifier, for a hierarchical representation of code and text; although they experimented with sharing encoder parameters, it lowered the final quality of their model. Our model, in contrast, uses a single encoder for embedding both queries and documents and skips the classifier part.

### 2.3 Clone Detection

To the best of our knowledge, the attempt to detect similar program code was made by Baker [5] who proposed to find duplicate code in exact and parametrized forms. The latter approach could be reformulated as code obfuscation and search for an obfuscated exact match. Krinke [24] proposed to apply graph theory, namely to compare shortest paths of abstract syntactic trees

3

extracted from the code snippets. Early work on algorithmic clone detection was summarized by Roy and Cordy [39]. Last but not least, we note one of the most recent algorithmic approaches *SourcererCC* [40], which proposes a sophisticated algorithm that includes the construction of a search index for candidate clone snippets.

Recent approaches are usually based on machine learning. Probabilistic approaches were used by Alomari and Stephan [2] who used latent Dirichlet allocation and Thaller et al. [42] who presented a novel graph probabilistic model based on density estimation. One of the first successful deep learning-based approaches was CClearner [29] that used text extracted from a program and its AST features and had a simplistic multilayer perceptron architecture for clone classification on a closed code base. More recent deep learning models include graph neural networks on ASTs [45]. Villmow et al. [44] based their approach on pretrained languages models.

## 2.4  Language models for source code

. After the success of BERT-like models for natural language, these approaches were also applied to programming languages. Several pretrained programming language models were presented recently, including: CodeBERT [13], which is a bimodal pretrained model for source code and natural language based on the RoBERTa Transformer architecture [30], trained on masked language modeling (MLM) and replaced token detection objectives; GraphCodeBERT [18] that uses data flow during pretraining to solve MLM, edge prediction, and node alignment tasks; SynCoBERT [46] that uses multimodal contrastive learning and is pretrained on identifier prediction and AST edge prediction.

## 2.5  Multilingual NLP

Open-domain question answering task assumes answering factoid questions without a predefined domain [26]. Since we propose a novel benchmark in cross-lingual code understanding, we also review multilingual NLP models, where recent research has been focused on non-English question answering (QA) datasets and multilingual transfer learning, usually from English to other languages.

Until recently, appropriate training and evaluation datasets have been scarce, but in recent years many works have collected loosely aligned data obtained through automatic translation or by parsing similar multilingual sources. Thus, Ahn et al. [1] presented an early attempt to multilingual question answering by translating an answer to the target language. Two years later, Bos and Nissim [7] presented another similar system using translation. Lee and Lee [27] showed successful transfer for multilingual QA with training on English data and evaluation in Chinese. Ferrández et al. [14] presented a work on multilingual knowledge base incorporating a system. Mitamura et al. [33] developed a system with the translation of keywords extracted from question to get an answer in the target language. Artetxe et al. [3] studied multilingual transfer of monolingual representations for a Transformer-based masked language model.

M'hamdi et al. [32] examined a multilingual optimization-based meta-learning approach to learn to adapt to new languages for QA. Gao and Callan [15] proposed unsupervised pretraining for dense passage retrieval, although the authors concentrated on retrieval itself and ignored the multilingual nature of the data.

Most approaches used extractive models, mostly due to the prevalence of datasets where, similar to SQuAD [37], labeled data includes an explicitly stated question, a passage containing an answer, and a span labeling for the answer. However, several works have considered generative QA: Kumar et al. [25] and Chi et al. [11] studied multilingual question generation, Riabi et al. [38] suggested a method to produce synthetic questions in a multilingual way by using multilingual MiniLM, and Shakeri et al. [41] proposed a method to generate multilingual question-answer pairs by a generative model (fine-tuned multilingual T5) based on samples automatically translated from English to the target language.

Generative QA was mostly considered for datasets with long answers, but the generative model FiD [23] achieved competitive results on SQuAD-like datasets, where an answer is supposed to be a short text span. For open domain QA, Lewis et al. [28] used generative models in their RAG approach that processes top $k$ passages from the retriever in the encoder simultaneously and uses their representations in the decoder for answer generation, in a process called fusion. Processing the passages independently in the encoder allows a model to scale to many contexts, as it only runs self-attention over one context at a time. The FiD model follows this paradigm, further improving the results in question generation. For QA over a knowledge graph, Zhou et al. [50] studied unsupervised bilingual lexicon induction for zero-shot multilingual transfer for multilingual QA in order to map training questions in the source language into questions in the target language for use as augmented training data, which is important for zero-resource languages.

In this work, we focus on available datasets for solving the multilingual Open Retrieval Question Answering (XOR) task. It is important because many languages are faced with both a lack of textual information and its asymmetry, in the case of concepts from other cultures.

## 3   Datasets

In this work we use two kinds of datasets, one for clone detection and another for code search.

### 3.1   Code Search

For code search we use the *CodeSearchNet* dataset introduced by [21]. The original version of *CodeSearchNet* consists of natural language queries paired with most relevant code snippets in six programming languages. Each snippet represents the code of a function collected from *GitHub* open source code.

**CodeSearchNet AdvTest** is a Python-only dataset constructed from the *CodeSearchNet* corpus by Lu et al. [31]. Each example, again, includes a function paired with a text document, and similar to Husain et al. [21] *AdvTest* takes the first paragraph of the documentation as the query. Lu et al. [31] make an interesting observation: after normalizing function and variable names with special tokens, the Mean Reciprocal Rank (MRR) scores of RoBERTa [30] and CodeBERT [13] models for code search on the original *CodeSearchNet* dataset drop from 0.809 to 0.419 and from 0.869 to 0.507 respectively (in *Python*). So, to improve the quality of the dataset and make it harder, they first filtered the data by removing examples for which the code could not be parsed into an abstract syntax tree, the document was shorter than 3 or longer than 256 tokens, the document contained special tokens such as "http://", or the document was empty or not written in English. The filtered dataset contains 251 820 / 9 604 / 19 210 examples in its training/validation/test sets respectively.

Then, to better test the understanding and generalization abilities of the model, Lu et al. [31] normalized function and variable names in testing and development sets to nondescript tokens such as *func* for the function name and *arg_i* for the $i$-th variable name. The task remains to search for source code based on a natural language query. Moreover, in contrast to the testing phase of previous works [13,21] that only involved 1 000 candidates, Lu et al. [31] used the entire test set for each query, which makes the *AdvTest* dataset even more difficult. The training set comes from the filtered CodeSearchNet dataset [21] where the code is represented in a raw form in addition to tokenization native to its programming language. *AdvTest* uses MRR as the basic evaluation metric, defined as $\text{MRR@Q} = \frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{\text{rank}_i}$, where $Q$ is the number of queries and rank is the position of the ground truth answer document among ranked candidates.

## 3.2 Clone Detection

In the clone detection task, the problem is to retrieve semantically similar codes given a code as the query. To train and test models for clone detection, we use the **POJ-104** dataset introduced by Mou et al. [34]. It comes from a pedagogical programming open judge (OJ) system that automatically judges the validity of submitted source code for specific problems by running the code. The POJ-104 dataset consists of 104 problems and includes 500 student-written C/C++ programs for each problem. The clone detection here is, given a program's source code, to retrieve other programs that solve the same problem. The problems are grouped into three sets with 64/16/24 problems for training, validation, and testing respectively.

The default metric for the POJ-104 dataset is Mean Average Precision (MAP), where the average precision (AP) is defined as $\text{AP} = \sum_{i=1}^{100}(R_i - R_{i-1})\cdot P_i$, where $R_i$ and $P_i$ are the precision and recall at threshold $i$, i.e., computed taking into account only top $i$ items from the candidate list. MAP is the mean AP over all queries. It is important to mention that for POJ-104 the maximal possible $i$ is 499 since there are only 500 candidates in total.

## 3.3 The XCD dataset

In previous works, cross-lingual and multilingual abilities of code language models were not sufficiently investigated. To fill this gap, we introduce a new multilingual clone detection/code retrieval dataset XCD. The dataset is evaluated in three different settings: *full comparison* clone detection approach similar to the BUCC dataset [48], *retrieval style* clone detection similar to POJ-104 [34], and a *hybrid* approach.

**Construction**. We use the *CodeForces* submissions dump as the data source, randomly choosing 100 problems and 100 accepted solutions in one language for each problem. We also randomly chose another 40 000 solutions in the same language and added them to the dataset, getting 50 000 code snippets per language. We chose 9 languages, namely Python, Java, C#, Ruby, JavaScript (JS), Haskell, PHP, OCaml, and Perl. Thus, XCD consists of 450 000 code snippets in total.

**Full comparison evaluation setup**. Here the task is interpreted as binary classification, similar to Xu et al. [48]. We evaluate a model on each pair from test set, which gives rise to $n^2$ comparisons. Each pair could be either positive, if the pair consists of solutions for the same problem, or negative otherwise. We use the classic $F_1$ measure for evaluation in this setup: $F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$.

**Retrieval style evaluation setup**. For retrieval style evaluation, we follow the design of POJ-104. The task aims to retrieve 100 snippets per language solving the same problem from the test set. We evaluate ranking on 10 000 positive snippets, i.e., the model should rank 10 000 documents, bringing on top the snippets solving the same problem. Similar to Mou et al. [34], we use mean average precision on the top 100 results (MAP@100) for evaluation.

**Hybrid evaluation setup**. We also propose a setup where we evaluate the models not only on positive snippets but on all snippets in the same language. This task is harder and more similar to *AdvTest*. As the metric we chose MRR@20, following Lu et al. [31].

**Cross-lingual evaluation**. Evaluation setups shown above emphasize the *multilingual* setting, i.e., evaluation inside the set of solutions in a single language. But we also define a more complex *cross-lingual* setting designed to measure cross-lingual code understanding. To achieve this, we extend the three setups to all languages at once, extending the sets of both relevant and irrelevant snippets to the corresponding snippets in all languages.

## 3.4 Additional Labeling

In addition to the solution status ("Accepted" or not), we also mined error statuses for the solutions since the platforms used for problem solving often provide them. In total, we mined more than 97 million code snippets in more than 10 programming languages. The *CodeForces* platform can return 15 types of verdicts for a submitted solution. We split the verdicts into 4 groups: *Defect* (code has a defect), *Skip* (code cannot be judged), *Accepted* (no defects detected), and *Wrong* (code that fails some tests or constraints). Below we describe and classify
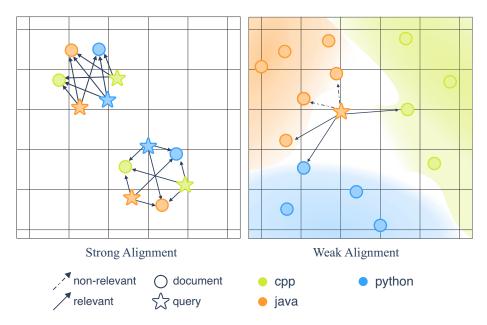
Figure 1: Strong and weak cross-lingual alignment.

some of the most common verdicts: (1) *Memory limit exceeded*: the program tries to consume more memory than allowed (*Wrong*); (2) *Time limit exceeded*: the program has not terminated in allotted time (*Wrong*); (3) *Runtime error*: the program terminated with a non-zero return code (e.g., due to division by zero, stack overflow etc.) (*Defect*); (4) *Wrong answer* on some tests (*Wrong*); (5) *Idleness limit exceeded*: the program did not use the CPU for a considerable time (*Defect*); (6) *Denial of judgement*: the solution was impossible to run due to a judging error or an error in the program, e.g., using extra large arrays (*Defect*); (7) *Rejected*: the program does not pass tests for an unclear reason (*Skip*); (8) *Accepted*: the program passed all tests.

## 4   Method

In this section, we introduce our pretraining approach CCT (Cross-Consistency Training). Its goal is to robustly learn the embedding space of code snippets and create a *strong* alignment between snippets solving the same problems across programming languages. The difference between strong and weak alignment is illustrated in Fig. 1: in a weakly aligned embedding space, the nearest neighbor might be a semantically similar snippet from a different language but generally most neighbors are in the same language, while in a strongly aligned space the similarity is purely semantic and does not care about the language at all.

To achieve strong alignment, we introduce a new contrastive learning objective $\mathcal{L}_{\mathrm{XCD}}$: for a randomly masked code snippet, we train the [CLS] vector

so that the [CLS] embedding will be closer to the snippet in another language that solves the same problem or the same snippet but masked in a different way than a random or similar but different (hard negative) snippet. We also use two additional loss functions: error detection loss $\mathcal{L}_{\mathrm{err}}$ and language modeling loss $\mathcal{L}_{\mathrm{LM}}$. As a result, our final loss function is a combination of the three losses:

$$\mathcal{L} = \mathcal{L}_{\mathrm{XCD}} + \mathcal{L}_{\mathrm{err}} + \mathcal{L}_{\mathrm{LM}}.$$

We describe the losses in details below.

## 4.1 Noise-contrastive estimation and losses

To learn a language-agnostic cross-lingual representation space, we propose a training procedure based on noise contrastive estimation (NCE). Let $\mathcal{X}$ and $\mathcal{Z}$ be some finite sets and $s_{\boldsymbol{\theta}} : \mathcal{X} \times \mathcal{Z} \to \mathbb{R}$ be a relevance score function differentiable in $\boldsymbol{\theta} \in \mathbb{R}^d$. The goal is to learn $\boldsymbol{\theta}$ such that the classifier $\mathbf{x} \mapsto \arg\max_{\mathbf{z} \in \mathcal{Z}} s_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$ has the optimal expected loss. This leads to conditional density estimation: for every $\mathbf{x} \in \mathcal{X}$

$$p_{\boldsymbol{\theta}}\left(\mathbf{z}|\mathbf{x}\right) = \frac{e^{s_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}}{\sum_{\mathbf{z}^- \in \mathcal{Z}} e^{s_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}^-)}} \tag{1}$$

with $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}, \mathbf{z}}\left[-\log p_{\boldsymbol{\theta}}\left(\mathbf{z}|\mathbf{x}\right)\right]$ being the optimum. In practice, optimizing this objective directly is infeasible: if $\mathcal{Z}$ is large the normalization term in (1) is intractable. Therefore, NCE uses subsampling, so (1) becomes

$$\pi_{\boldsymbol{\theta}}\left(\mathbf{z}|\mathbf{x}\right) = \frac{e^{s_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}}{\sum_{\mathbf{z}^- \in \mathcal{B}_{\mathbf{x}, \mathbf{z}}} e^{s_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}^-)} + e^{s_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}}, \tag{2}$$

where $\mathcal{B}_{\mathbf{x}, \mathbf{z}} = \{\mathbf{z}_1^-, \mathbf{z}_2^-, \ldots, \mathbf{z}_n^-\}$ is a set of *negatives* sampled from $\mathcal{Z}$ that do not match the *positive* answer $\mathbf{z}^+$ for this $\mathbf{x}$. NCE also often uses objectives similar to (2) but with $\pi_{\boldsymbol{\theta}}\left(\hat{\mathbf{z}}|\mathbf{z}\right)$ where $\mathbf{z}$ and $\hat{\mathbf{z}}$ come from the same space, and the objective corresponds to some similarity function.

**Cross-lingual objective**. Contrastive learning frequently employs pretext tasks to learn data representations without the need for labeled examples. In the context of learning from a multilingual set of documents, a possible pretext task would be to train a network to differentiate between documents with similar content but written in different languages (positive pairs) and those with dissimilar content (negative pairs). This leads to the loss function

$$\mathcal{L}_{\mathrm{XCD}}(\boldsymbol{\theta}) = \mathbb{E}_{(\hat{\mathbf{z}}, \mathbf{z}) \sim \mathcal{W}_{\mathrm{XCD}}}\left[-\log \pi_{\boldsymbol{\theta}}\left(\hat{\mathbf{z}}|\mathbf{z}\right)\right], \tag{3}$$

where $\mathcal{W}_{\mathrm{XCD}}$ is a distribution on the set of pairs of submissions in different programming languages from the XCD dataset (Section 3) that shows if the submissions are solving the same problem or not.

**Language modeling objective** is a standard implementation of noise contrastive estimation similar to Devlin et al. [12]:

$$\mathcal{L}_{\mathrm{LM}}(\boldsymbol{\theta}) = \mathbb{E}_{(\hat{\mathbf{z}}, \mathbf{z}) \sim \mathcal{W}_{\mathrm{LM}}}\left[-\log \pi_{\boldsymbol{\theta}}\left(\hat{\mathbf{z}}|\mathbf{z}\right)\right], \tag{4}$$

where $\mathcal{W}_{\mathrm{LM}}$ is a distribution of tokens over a vocabulary, in our case collected from XCD.

**Error detection objective**. For the error detection loss $\mathcal{L}_{err}$, we use binary cross-entropy for the classification of whether the code can raise an error during execution:

$$\mathcal{L}_{\mathrm{err}}(\boldsymbol{\theta}) = \mathbb{E}_{(\hat{\mathbf{z}}, \mathbf{z}) \sim \mathcal{W}_{\mathrm{BCE}}} \left[ -\log \pi_{\boldsymbol{\theta}} \left( \hat{\mathbf{z}} | \mathbf{z} \right) \right], \tag{5}$$

where $\mathcal{W}_{\mathrm{BCE}}$ is again a distribution on XCD but in this case reflecting if a submission was successful based on the verdicts (Section 3.4). $\mathcal{L}_{\mathrm{err}}$ is specific for source code data, intended to help the model understand code syntax and recognize errors.

## 4.2  Hard negative mining

Previous works on contrastive learning show the importance of training on hard negative samples [22, 36]. These works use iterative training to get hard negatives; however, our data already contains some strong negative examples, namely preliminary solutions from the same users that solve the same problems but fail some of the tests (that is why the user would submit an updated solution to get the "Accepted" verdict). Thus, our hard negative examples are mined as failed solutions from the same user if they exist; if not then we use failed solutions from random users, and only if there are no such solutions (e.g., for an unpopular problem) then we use a random submission for a random problem.

# 5  Experiments

In this section, we describe the details about parallel data pretraining and our CCT pipeline for multilingual clone detection and code search tasks.

## 5.1  Pretraining

We initialize our model with a pretrained GraphCodeBERT$_{\mathrm{base}}$ model [18]; we call the resulting model CCT-LM$_{\mathrm{base}}$. For the full comparison setup we follow Ruder et al. [20] and Tien and Steinert-Threlkeld [43], where the similarity score cutoff distinguishing clones from non-clones is optimized separately for each language pair. Similarity scores are calculated based on dot products of [CLS] vector representations.

**Hyperparameters**. We use the AdamW optimizer with learning rate 1e-4, weight decay 0.01, and linear learning rate decay. Similar to Gao and Callan [15], we use gradient cashing for pretraining with accumulated batch size 500. We train our models on 4 NVIDIA Tesla V100 GPUs, spending 384 GPU hours in total for the base model pretraining.

| | Clone detection (**MAP**) | Code search (**MRR**) |
|---|---|---|
| RoBERTa-base [30] | 76.67 | 18.33 |
| CodeBERT [13] | 82.67 | 27.19 |
| SynCoBERT [46] | 88.24 | 38.10 |
| CodeRoBERTa | — | 42.35 |
| GraphCodeBERT [18] | 85.16 | — |
| CasCode [16] | — | 43.98 |
| [44] | 91.34 | — |
| CCT-LM$_{base}$ | **96.73** | **47.18** |

Table 1: Results on code clone detection on the POJ-104 dataset and code search on the *AdvTest* dataset.

## 5.2 Monolingual Results

Table 1 presents the results of our CCT-LM model (pretrained from a GraphCodeBERT checkpoint with CCT) in comparison to existing approaches. The results show that CCT-LM outperforms all previous models by a large margin, even in this monolingual setting. This shows that strong alignment enforced by the CCT pretraining is not only helpful for multilingual transfer but also improves the latent space structure in general.

## 5.3 Multilingual Results

Results in the multilingual setting on the proposed XCD dataset are shown in the top half of Table 2. In the full comparison setup, it is interesting that knowledge transfer from the POJ-104 dataset also does not help, and the metrics remain very low. However, CCT-LM shows much better results, obviously due to the multilingual design of CCT pretraining. We do not evaluate BM25 in this setup since it is not supposed to compare two documents.

For retrieval style evaluation, the results also show CCT-LM strongly outperforming all baselines and actually providing a viable solution for the problem while GraphCodeBERT does not yield reasonable results in any programming language. We note that BM25 (a strong baseline for natural language IR tasks) does not work for clone detection, which is natural since it relies on identical tokens which could be scarce even between code snippets solving the same problem.

Results for the hybrid evaluation setup show the same picture: BM25 still does not work, code language models can transfer knowledge across different solutions to some extent, and the improvement in metrics here between a fully unsupervised method and training on POJ-104 clone detection is noticeably greater. Still, CCT-LM far exceeds every other method here and generally sets a new baseline for multilingual code-related tasks.

| | Python | Java | C# | Ruby | JS | Haskell | PHP | OCaml | Perl | Avg |
|---|---|---|---|---|---|---|---|---|---|---|
| **Multilingual setting** | | | | | | | | | | |
| **Full Comparison, $F_1$ measure** | | | | | | | | | | |
| GraphCodeBERT$_{\text{base}}$ | 0.02 | 0.05 | 0.00 | 0.04 | 0.00 | 0.02 | 0.01 | 0.03 | 0.01 | 0.02 |
| GraphCodeBERT$_{\text{base}}^{\text{POJ}}$ | 0.04 | 0.00 | 0.01 | 0.06 | 0.07 | 0.08 | 0.06 | 0.06 | 0.06 | 0.05 |
| CCT-LM$_{\text{base}}$ | **22.24** | **18.39** | **17.33** | **23.33** | **10.46** | **17.64** | **21.43** | **17.01** | **16.40** | **18.24** |
| **Retrieval Style, MAP@100** | | | | | | | | | | |
| BM25 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| GraphCodeBERT$_{\text{base}}$ | 7.21 | 9.25 | 1.33 | 4.28 | 1.59 | 5.78 | 6.08 | 2.90 | 10.37 | 5.42 |
| GraphCodeBERT$_{\text{base}}^{\text{POJ}}$ | 30.12 | 24.63 | 23.54 | 32.78 | 36.64 | 24.45 | 37.21 | 33.94 | 45.33 | 32.07 |
| CCT-LM | **87.42** | **55.99** | **65.35** | **72.12** | **74.32** | **81.05** | **83.21** | **71.53** | **71.89** | **73.65** |
| **Hybrid, MRR@20** | | | | | | | | | | |
| BM25 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| GraphCodeBERT$_{\text{base}}$ | 2.08 | 5.42 | 0.22 | 2.59 | 0.80 | 1.99 | 2.90 | 1.40 | 5.23 | 2.51 |
| GraphCodeBERT$_{\text{base}}^{\text{POJ}}$ | 27.10 | 20.04 | 19.44 | 30.98 | 28.37 | 19.70 | 32.89 | 30.08 | 39.98 | 27.62 |
| CCT-LM | **74.97** | **62.08** | **58.77** | **80.60** | **74.56** | **62.27** | **81.21** | **72.64** | **79.16** | **71.80** |
| **Cross-lingual setting** | | | | | | | | | | |
| **Full Comparison, $F_1$ measure** | | | | | | | | | | |
| GraphCodeBERT$_{\text{base}}$ | 0.01 | 0.01 | 0.00 | 0.01 | 0.00 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| GraphCodeBERT$_{\text{base}}^{\text{POJ}}$ | 0.01 | 0.00 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| CCT-LM$_{\text{base}}$ | **8.92** | **9.46** | **4.78** | **6.01** | **7.33** | **5.82** | **6.47** | **5.33** | **3.56** | **6.40** |
| **Retrieval Style, MAP@100** | | | | | | | | | | |
| BM25 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| GraphCodeBERT$_{\text{base}}$ | 3.18 | 5.24 | 0.23 | 1.77 | 1.15 | 3.38 | 3.12 | 1.90 | 16.27 | 4.02 |
| GraphCodeBERT$_{\text{base}}^{\text{POJ}}$ | 12.83 | 14.75 | 9.33 | 12.78 | 17.16 | 15.94 | 19.53 | 16.01 | 23.88 | 15.80 |
| CCT-LM | **44.82** | **20.34** | **23.33** | **35.01** | **32.57** | **40.07** | **43.36** | **36.66** | **37.80** | **34.88** |
| **Hybrid, MRR@20** | | | | | | | | | | |
| BM25 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| GraphCodeBERT$_{\text{base}}$ | 1.24 | 2.42 | 0.34 | 1.28 | 0.82 | 0.93 | 1.43 | 0.76 | 2.15 | 1.26 |
| GraphCodeBERT$_{\text{base}}^{\text{POJ}}$ | 20.12 | 13.08 | 10.37 | 17.28 | 12.62 | 19.70 | 14.31 | 18.08 | 18.33 | 15.98 |
| CCT-LM | **30.83** | **22.77** | **19.32** | **32.66** | **31.64** | **20.80** | **31.59** | **40.42** | **39.40** | **29.93** |

Table 2: Multilingual and cross-lingual clone detection in three evaluation setups on the XCD dataset.

## 5.4 Cross-lingual Results

Our results in this setting are presented in the bottom half of Table 2. All conclusions derived for the multilingual case (Section 5.3) apply here too, but in comparison to the multilingual setting, cross-lingual tasks are significantly harder and all values are lower. We suggest that the difference in the results across programming languages could be caused by the imbalance in the pre-training dataset.

| | Java | Ruby | PHP | Go | JS | Avg |
|---|---|---|---|---|---|---|
| CodeBERT$_{\text{base}}$ | 46.37 | 50.65 | 37.83 | 50.65 | 50.48 | 47.19 |
| GraphCodeBERT$_{\text{base}}$ | 47.33 | 59.95 | 37.47 | 60.28 | **52.04** | 51.41 |
| CCT-LM$_{\text{base}}$ | **48.71** | **62.25** | **42.78** | **61.44** | 51.06 | **53.24** |

Table 3: Zero-shot retrieval; $F_1$ score, *CodeSearchNet*.

| GraphCodeBERT | Clone Detection (**MAP**) | Code Search (**MRR**) | Defect detection (**Acc**) |
|---|---|---|---|
| Base | 85.16 | 45.80 | 62.51 |
| Base + $\mathcal{L}_{\text{XCD}}$ | 95.92 | 29.93 | 61.05 |
| Base + $\mathcal{L}_{\text{XCD}}$ + $\mathcal{L}_{\text{LM}}$ | 95.67 | 47.18 | 63.68 |
| Base + $\mathcal{L}_{\text{XCD}}$ + $\mathcal{L}_{\text{LM}}$ | 96.03 | 45.22 | 64.91 |
| Base + $\mathcal{L}_{\text{XCD}}$ + $\mathcal{L}_{\text{LM}}$ + SL | 96.46 | 47.33 | - |
| Base + $\mathcal{L}_{\text{XCD}}$ + $\mathcal{L}_{\text{LM}}$ + $\mathcal{L}_{\text{err}}$ + SL | **96.73** | **47.57** | **65.58** |

Table 4: GraphCodeBERT variations: clone detection on POJ-104, code search on *AdvTest*, defect detection on *Devign*; SL denotes the size limit.
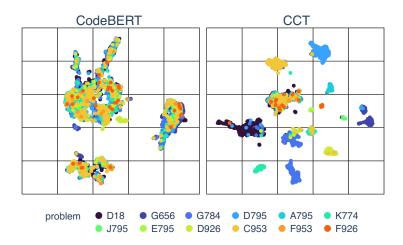
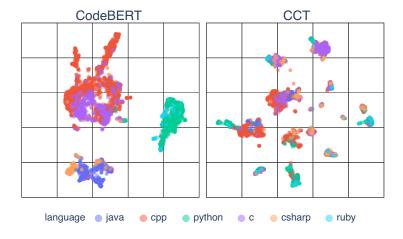# 6 Analysis

## 6.1 Zero-shot Results

We investigated zero-shot transfer from Python to Java, Ruby, PHP, Go, and JavaScript on the *CodeSearchNet* dataset for previously introduced code language models and our CCT-LM. The zero-shot results are presented in Table 3. As evidence for the power of pretrained language models, we see that existing approaches show rather good results even though they have not been trained on the retrieval task. By leveraging its multilingual ability, CCT-LM improves over the baselines in the zero-shot setup for all languages except *JavaScript* (JS).

## 6.2 Latent space structure

Figure 1 showed an abstract representation of the basic CCT idea of semantically aligned language-agnostic embedding space. Figure 2 turns this theory into practice with projections of actual embeddings for sample code snippets before and after CCT training. The snippets represent solutions for 12 sample tasks in six programming languages. We see that after CCT, representations of code snippets are not aligned by language but rather by problem (Fig. 2b), while their alignment had been language-dependent before CCT (Fig. 2a). This illustrates that CCT training significantly improves the multilingual latent space for code snippets, making it semantic and language-agnostic.

(a) Projected embeddings of 12 coding problems.



(b) The same embeddings shown by programming language.

Figure 2: Sample multilingual embeddings.

## 6.3 Ablation Study

In this section, we study the effects of various parts of CCT. Table 4 shows the results of several GraphCodeBERT-based models on clone detection, code search, and defect detection tasks. We decided to add defect detection in this analysis since one component of our loss, $\mathcal{L}_{\text{err}}$, is aligned with this task; it is evaluated on the *Devign* dataset [49] with $21\,854/2\,732/2\,732$ snippets in its train/dev/test split and accuracy as the metric. We compare the GraphCodeBERT

base model with different pretraining losses and size limits. Changes in metrics on different tasks are far from uniform; for instance, the pure contrastive loss $\mathcal{L}_{\text{XCD}}$ gives a significant quality boost for code retrieval but hinders code search even more significantly. Adding the masked LM loss $\mathcal{L}_{\text{LM}}$, however, compensates for this, probably because representations from the general GraphCodeBERT pretraining are preserved better. Next, $\mathcal{L}_{\text{err}}$ significantly improves quality for defect detection and yields small improvements for clone detection. Limiting the size to GraphCodeBERT's maximum input length 512 (SL) significantly reduces the number of code snippets and problems used for pretraining but improves the metrics for all tasks. We note that state of the art results for defect detection on this dataset are at 65.78% accuracy [47], only slightly better than our model.

## 6.4   Limitations

We have studied several programming languages, including Python and Java, in our XCD setup; although all our methods seem to be language-agnostic, a further study for other languages would be interesting, especially since all considered languages are interpreted rather than compiled (like C/C++). Many inputs exceed 512 tokens; we used standard truncation for evaluation (taking into consideration only the beginning of the code), which may be suboptimal, and more suitable input representations could be found. Since we apply truncation, the code's AST can be (and often is) damaged. We skipped ASTs in our approach, but it should benefit from their usage once there is a method for correct AST extraction. While in training we skipped long code snippets entirely, we expect our model to improve with training on long documents. We also suppose that the model would benefit from increasing the batch size by using more powerful hardware with more memory. Note also that while CCT-LM significantly improved state of the art in clone detection and code search, defect detection proved to be harder; we hypothesize that it may need a different approach for code representation.

## 7   Conclusion

Understanding semantic similarity is an important aspect of language processing that opens up ways to solve many different tasks, for both natural and programming languages. In this work, we have presented a new method CCT-LM that improves this ability via a novel CCT pretraining approach and demonstrated its viability on the clone detection and code search tasks. We have also formulated a novel task of multilingual clone detection, presented the XCD dataset for multilingual and cross-lingual source code analysis, formalized in three evaluation setups. The proposed CCT-LM model has outperformed strong baselines in all presented tasks, proving that CCT pretraining provides better semantic similarity understanding for a language model. The cross-lingual setup proved to be much harder to solve with current language models, leaving a lot of room

for improvement. We hope that our method will be helpful in other source code processing tasks that we have not covered and left as future work. Moreover, we believe that modifications of our approach can be useful for NLP and perhaps other fields of machine learning.

# References

[1] Kisuh Ahn, Beatrice Alex, Johan Bos, Tiphaine Dalmas, Jochen L Leidner, and Matthew B Smillie. Cross-lingual question answering using off-the-shelf machine translation. In *Workshop of the Cross-Language Evaluation Forum for European Languages*, pages 446–457. Springer, 2004.

[2] Hakam W Alomari and Matthew Stephan. Towards slice-based semantic clone detection. In *2018 IEEE 12th International Workshop on Software Clones (IWSC)*, pages 58–59. IEEE, 2018.

[3] Mikel Artetxe, Sebastian Ruder, and Dani Yogatama. On the cross-lingual transferability of monolingual representations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4623–4637, Online, July 2020. Association for Computational Linguistics.

[4] Alberto Bacchelli, Michele Lanza, and Romain Robbes. Linking e-mails and source code artifacts. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 375–384, 2010.

[5] Brenda S Baker. A program for identifying duplicated code. *Computing Science and Statistics*, pages 49–49, 1993.

[6] Ohad Barzilay, Christoph Treude, and Alexey Zagalsky. Facilitating crowd sourced software engineering via stack overflow. In *Finding Source Code on the Web for Remix and Reuse*, pages 289–308. Springer, 2013.

[7] Johan Bos and Malvina Nissim. Cross-lingual question answering by answer translation. In *CLEF (Working Notes)*. Citeseer, 2006.

[8] Joel Brandt, Mira Dontcheva, Marcos Weskamp, and Scott R Klemmer. Example-centric programming: integrating web search into the development environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 513–522, 2010.

[9] Brock Angus Campbell and Christoph Treude. Nlp2code: Code snippet content assist via natural language tasks. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 628–632. IEEE, 2017.

[10] Wing-Kwan Chan, Hong Cheng, and David Lo. Searching connected api subgraph via text phrases. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, pages 1–11, 2012.

[11] Zewen Chi, Li Dong, Furu Wei, Wenhui Wang, Xian-Ling Mao, and Heyan Huang. Cross-lingual natural language generation via pre-training, 2019.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[13] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online, 2020. Association for Computational Linguistics.

[14] Sergio Ferrández, Antonio Toral, Oscar Ferrández, Antonio Ferrández, and Rafael Munoz. Applying wikipedia's multilingual knowledge to cross–lingual question answering. In *International Conference on Application of Natural Language to Information Systems*, pages 352–363. Springer, 2007.

[15] Luyu Gao and Jamie Callan. Unsupervised corpus aware language model pre-training for dense passage retrieval. 2021.

[16] Akhilesh Deepak Gotmare, Junnan Li, Shafiq Joty, and Steven CH Hoi. Cascaded fast and slow models for efficient semantic code search. *arXiv preprint arXiv:2110.07811*, 2021.

[17] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. Deep code search. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 933–944. IEEE, 2018.

[18] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. GraphCodeBERT: Pre-training Code Representations with Data Flow. In *ICLR*, 2021.

[19] Masum Hasan, Tanveer Muttaqueen, Abdullah Al Ishtiaq, Kazi Sajeed Mehrab, Md Mahim Anjum Haque, Tahmid Hasan, Wasi Ahmad, Anindya Iqbal, and Rifat Shahriyar. Codesc: A large code–description parallel dataset. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 210–218, 2021.

[20] Junjie Hu, Sebastian Ruder, Aditya Siddhant, Graham Neubig, Orhan Firat, and Melvin Johnson. Xtreme: A massively multilingual multi-task benchmark for evaluating cross-lingual generalization. arXiv, 2020.

[21] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. September 2019.

[22] Gautier Izacard and Edouard Grave. Distilling knowledge from reader to retriever for question answering. 2020.

[23] Gautier Izacard and Édouard Grave. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880, 2021.

[24] J. Krinke. Identifying similar code with program dependence graphs. In *Proceedings Eighth Working Conference on Reverse Engineering*, pages 301–309, 2001.

[25] Vishwajeet Kumar, Nitish Joshi, Arijit Mukherjee, Ganesh Ramakrishnan, and Preethi Jyothi. Cross-lingual training for automatic question generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4863–4872, Florence, Italy, July 2019. Association for Computational Linguistics.

[26] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

[27] Chia-Hsuan Lee and Hung-Yi Lee. Cross-lingual transfer learning for question answering. *arXiv preprint arXiv:1907.06042*, 2019.

[28] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. 2021.

[29] Liuqing Li, He Feng, Wenjie Zhuang, Na Meng, and Barbara Ryder. Cclearner: A deep learning-based clone detection approach. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 249–260. IEEE, 2017.

[30] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[31] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. Codexglue: A machine learning benchmark dataset for code understanding and generation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[32] Meryem M'hamdi, Doo Soon Kim, Franck Dernoncourt, Trung Bui, Xiang Ren, and Jonathan May. X-METRA-ADA: Cross-lingual meta-transfer learning adaptation to natural language understanding and question answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3617–3632, Online, June 2021. Association for Computational Linguistics.

[33] Teruko Mitamura, Mengqiu Wang, Hideki Shima, and Frank Lin. Keyword translation accuracy and cross-lingual question answering inchinese and japanese. In *Proceedings of the Workshop on Multilingual Question Answering-MLQA '06*, 2006.

[34] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. Convolutional neural networks over tree structures for programming language processing. 2016.

[35] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. Mining stackoverflow to turn the ide into a self-confident programming prompter. In *Proceedings of the 11th working conference on mining software repositories*, pages 102–111, 2014.

[36] Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. Rocketqa: An optimized training approach to dense passage retrieval for open-domain question answering. *ArXiv*, abs/2010.08191, 2021.

[37] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.

[38] Arij Riabi, Thomas Scialom, Rachel Keraron, Benoît Sagot, Djamé Seddah, and Jacopo Staiano. Synthetic data augmentation for zero-shot cross-lingual question answering. *arXiv preprint arXiv:2010.12643*, 2020.

[39] Chanchal Roy and James Cordy. A survey on software clone detection research. *School of Computing TR 2007-541*, 01 2007.

[40] Hitesh Sajnani, Vaibhav Saini, Jeffrey Svajlenko, Chanchal K Roy, and Cristina V Lopes. Sourcerercc: Scaling code clone detection to big-code. In *Proceedings of the 38th International Conference on Software Engineering*, pages 1157–1168, 2016.

[41] Siamak Shakeri, Noah Constant, Mihir Sanjay Kale, and Linting Xue. Multilingual synthetic question and answer generation for cross-lingual reading comprehension. *arXiv preprint arXiv:2010.12008v1*, 2020.

[42] Hannes Thaller, Lukas Linsbauer, and Alexander Egyed. Towards semantic clone detection via probabilistic software modeling. In *2020 IEEE 14th International Workshop on Software Clones (IWSC)*, pages 64–69. IEEE, 2020.

19

[43] Chih-chan Tien and Shane Steinert-Threlkeld. Bilingual alignment transfers to multilingual alignment for unsupervised parallel text mining. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8696–8706, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[44] Johannes Villmow, Viola Campos, Adrian Ulges, and Ulrich Schwanecke. Addressing leakage in self-supervised contextualized code retrieval, 2022.

[45] Wenhan Wang, Ge Li, Bo Ma, Xin Xia, and Zhi Jin. Detecting code clones with graph neural network and flow-augmented abstract syntax tree. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 261–271. IEEE, 2020.

[46] Xin Wang, Yasheng Wang, Fei Mi, Pingyi Zhou, Yao Wan, Xiao Liu, Li Li, Hao Wu, Jin Liu, and Xin Jiang. SynCoBERT: Syntax-Guided Multi-Modal Contrastive Pre-Training for Code Representation, September 2021. Number: arXiv:2108.04556 arXiv:2108.04556 [cs].

[47] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8696–8708, 2021.

[48] Ruochen Xu, Yiming Yang, Naoki Otani, and Yuexin Wu. Unsupervised cross-lingual transfer of word embedding spaces. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2465–2474, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[49] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In *Advances in Neural Information Processing Systems*, pages 10197–10207, 2019.

[50] Yucheng Zhou, Xiubo Geng, Tao Shen, Wenqiang Zhang, and Daxin Jiang. Improving zero-shot cross-lingual transfer for multilingual question answering over knowledge graph. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5822–5834, Online, June 2021. Association for Computational Linguistics.