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

Nikita Sorokin[1,4*], Anton Tikhonov[1,3*], Dmitry Abulkhanov[8], Ivan Sedykh[1], Irina Piontkovskaya[2], Sergey Nikolenko[5,6], Valentin Malykh[1,6,7]
[1]MTS AI, Moscow, Russia; [2]Luzin Research Center, Moscow, Russia;
[3]ITMO University, Saint-Petersburg, Russia; [4]Higher School of Economics, Moscow, Russia;
[5]St. Petersburg Department of the Steklov Institute of Mathematics, Saint-Petersburg, Russia;
[6]ISP RAS Research Center for Trusted Artificial Intelligence, Moscow, Russia;
[7]IITU University, Almaty, Kazakhstan; [8]Independent Researcher

Abstract

We consider the well-known and important tasks of clone detection and information retrieval for source code. The most standard setup is to search clones inside the same language code snippets. But it is also useful to find code snippets with identical behaviour in different programming languages. Nevertheless multi- and cross-lingual clone detection has been little studied in literature. We present a novel training procedure, cross-consistency training (CCT) leveraging cross-lingual similarity, that we apply to train language models on source code in various programming languages. We show that this training is effective both for encoder- and decoder-based models. The trained encoder-based CCT-LM model achieves a new state of the art on POJ-104 (monolingual C++ clone detection benchmark) with 96.73% MAP and AdvTest (monolingual Python code search benchmark) with 47.18% MRR. The decoder-based CCT-LM model shows comparable performance in these tasks. In addition, we formulate the multi- and cross-lingual clone detection problem and present XCD, a new benchmark dataset produced from CodeForces submissions.

1 Introduction

Clone detection is crucial in software development for identifying semantically similar code, aiding in unification, refactoring, and side effect control. Originally formulated for C/C++ by Mou et al. (2016), the task has since expanded to other languages, with the next step being multilingual clone detection. This work introduces a new multilingual dataset XCD and establishes baseline models.

Early clone detection relied on algorithmic methods (Baker, 1993; Krinke, 2001), later evolving into machine learning-based approaches (Li et al., 2017; Thaller et al., 2020; Gotmare et al., 2021) that embed code snippets for similarity-based retrieval. We propose CCT, a novel training technique that enhances code embeddings, achieving state-of-the-art results on both POJ-104 (Mou et al., 2016) and our new XCD dataset. Additionally, we demonstrate that CCT-LM, trained with CCT, is also effective for code search, as formulated by Lu et al. (2021b).

Main Contributions CCT – A pretraining method for aligning multilingual code snippets. XCD – A novel multilingual clone detection dataset from CodeForces. State-of-the-art results on POJ-104 and XCD with CCT-LM. CCT-LM achieves state-of-the-art on *AdvTest* for code search.

2 Related Work

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

Datasets. Husain et al. (2019) 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. (2021) 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. There are two main datasets for clone detection: POJ-104 (Mou et al., 2016) and *BigCloneBench* (Wang et al., 2020). POJ-104 represents a comparatively small corpus of C++ solutions from a student judging system. *BigCloneBench* comprises a vast dataset containing automatically mined data in the Java language.

Code Search. Gu et al. (2018) introduced

*Equal contribution.

Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 4: Student Research Workshop), pages 178–185

dense vector representations for code search, training two recurrent neural networks for source code and text. Feng et al. (2020) used a language model to produce these representations. Gotmare et al. (2021) employed three Transformer-based models for hierarchical encoding but found parameter sharing reduced quality. In contrast, our model uses a single Transformer decoder to embed queries and documents, omitting the classifier.

Clone Detection. One of the first successful deep learning approaches was CClearner (Li et al., 2017) 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 (Wang et al., 2020) and employ pretrained language models Villmow et al. (2022).

Language models for source code. BERT-like models, initially successful in natural language processing, have been adapted for programming languages. Several pre-trained models have emerged, including CodeBERT (Feng et al., 2020), a bimodal model trained on masked language modeling (MLM) and replaced token detection; GraphCodeBERT (Guo et al., 2021), which incorporates abstract syntax trees for training; and SynCoBERT (Wang et al., 2021), which leverages multimodal contrastive learning with identifier and AST edge prediction. More recently, autoregressive decoder models like DeepSeek-Coder (Guo et al., 2024) have gained prominence, focusing on source code generation tasks such as code completion and documentation generation.

3 Datasets

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

Code Search. For code search we use the *CodeSearchNet* dataset introduced by Husain et al. (2019). 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

AdvTest is a Python-only dataset derived from the *CodeSearchNet* corpus by Lu et al. (2021b), pairing functions with text where the first documentation paragraph serves as the query (Husain et al., 2019).

Lu et al. (2021b) found that normalizing function and variable names significantly reduces Mean Reciprocal Rank (MRR) scores, dropping from 0.809 to 0.419 for RoBERTa (Liu et al., 2019) and 0.869 to 0.507 for CodeBERT (Feng et al., 2020). They improved dataset quality by filtering unparsable code, overly short/long documents, special tokens, and non-English or empty texts, resulting in 251 820 training, 9 604 validation, and 19 210 test examples.

To assess generalization, *AdvTest* normalizes function and variable names in the development and test sets, replacing them with generic tokens (e.g., *func*, *arg_i*). Unlike prior works (Husain et al., 2019; Feng et al., 2020), which evaluated on 1 000 candidates per query, *AdvTest* uses the entire test set, increasing difficulty. The training data, derived from the filtered CodeSearchNet (Husain et al., 2019), retains raw code and applies language-specific tokenization. Performance is measured using Mean Reciprocal Rank (MRR).

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. (2016). 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 $\mathrm{AP} = \sum_{i=1}^{100}(R_i - R_{i-1}) \cdot P_i$, 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.

3.1 XCD Dataset

Existing works have not thoroughly explored the multilingual capabilities of code language models. To address this gap, we introduce XCD, a new multilingual clone detection and code re-

trieval dataset. The dataset supports three evaluation settings: full comparison (binary classification like BUCC (Xu et al., 2018)), **retrieval-style clone detection** (similar to POJ-104 (Mou et al., 2016)), and a hybrid approach

We constructed XCD using CodeForces submissions, selecting 110 problems with 100 accepted solutions per problem in five languages (Python, Java, C#, C++, C), totaling 55,000 snippets.

Evaluation Setups Full Comparison Binary classification of test set pairs (n^2 comparisons). Each pair is positive if solving the same problem, otherwise negative. Evaluated using F1-score (Sasaki et al., 2007).

Retrieval Style Tasked with retrieving 100 snippets per language solving the same problem from 11,000 positive snippets. Evaluated using MAP@100 (Mou et al., 2016).

Hybrid Evaluation Includes all snippets in the same language, making it more challenging (similar to *AdvTest*). Evaluated using MRR@R (Lu et al., 2021b).

Cross-Lingual Evaluation Extends all setups across multiple languages to assess cross-lingual code understanding.

Additional Labeling Beyond solution status (Accepted/Not Accepted), we also mined error statuses from 97M code snippets across 10+ programming languages. CodeForces provides 15 verdicts, which we categorized into four groups:

1. Defect – Runtime errors (e.g., division by zero, stack overflow).
2. Skip – Judging errors (e.g., rejected due to unclear reasons).
3. Accepted – Passed all tests.
4. Wrong – Failed tests or constraints (e.g., time/memory limit exceeded).

This additional labeling enhances dataset utility for error prediction and robust code retrieval.

4 Method

In this section, we introduce our pre-training 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.

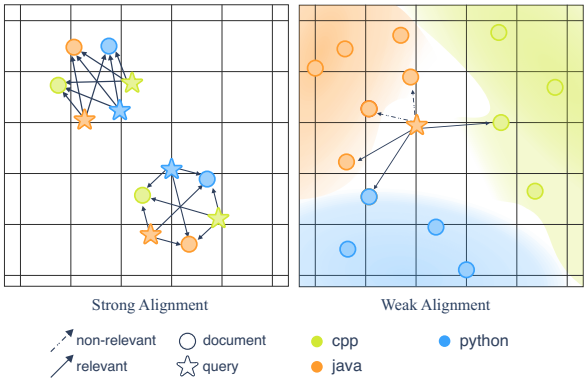


Figure 1: Strong and weak cross-lingual alignment.

To achieve strong alignment, we employ a contrastive learning objective $\mathcal{L}_{\text{XCD}}$: for a randomly code snippet, we train the vector representations of the source code tokens in such a way that their aggregation, for example, averaging or last token, is closer to the source code, which solves the same problem regardless of the programming language. This ensures that the embeddings of the source code differentiates between related snippets and random or similar but different (hard negative) snippets effectively.

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 $\mathbf{x} \in \mathcal{X}$

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

with $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x},\mathbf{z}} \left[-\log p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\right]$ 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

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

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 *pos-*

itive answer $\mathbf{z}^+$ for this $\mathbf{x}$. NCE also often uses objectives similar to (2) but with $\pi_{\boldsymbol{\theta}}(\hat{\mathbf{z}}|\mathbf{z})$ 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:

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

where $\mathcal{W}_{\text{XCD}}$ 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.

Hard negative mining. Previous works on contrastive learning show the importance of training on hard negative samples (Qu et al., 2021; Izacard and Grave, 2020). They used iterative training to get hard negatives, but our data already contains strong negative examples as preliminary solutions from the same users that solve the same problems but fail some tests (that is why a user would submit an updated solution to get the "Accepted" verdict). Thus, we mine hard negative examples as failed solutions from the same user; if there are none we use failed solutions from random users, and only if there are none (e.g., for an unpopular problem) we use a random submission for a random problem.

5 Experiments

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

Pretraining. We train two models, one is encoder-based, which is initialized with pretrained GraphCodeBERT$_{\text{base}}$ (Guo et al., 2021); we call the resulting model CCT-LM$_{\text{enc}}$. Another one is decoder-based, which is initilized with a pretrained DeepSeek-Coder-1.3B model (Guo et al., 2024); we call the resulting model CCT-LM$_{\text{dec}}$. Similarity scores are calculated based on dot products of the last token vector representations, but we also researched using various types of poolings and allowing bidirectional attention.

Hyperparameters. We use the AdamW optimizer with learning rate 5e-5, weight decay 0.01, and linear learning rate decay. We use gradient accumulation for pretraining with an effective batch size of 2000.

	Clone detection (MAP)	**Code search (MRR)**
Endcoder-only		
RoBERTa-base (Liu et al., 2019)	76.67	18.33
CodeBERT (Feng et al., 2020)	82.67	27.19
SynCoBERT (Wang et al., 2021)	88.24	38.10
CodeRoBERTa	—	42.35
GraphCodeBERT (Guo et al., 2021)	85.16	—
CasCode (Gotmare et al., 2021)	—	43.98
Villmow et al. (2022)	91.34	—
CCT-LM$_{\text{enc}}$	**96.73**	**47.18**
Decoder-only		
CodeGen (Nijkamp et al., 2023)	89.68	—
CodeGPT (Lu et al., 2021a)	87.96	—
SantaCoder (Allal et al., 2023)	83.98	—
Phi-1 (Gunasekar et al., 2023)	92.72	—
CCT-LM$_{\text{dec}}$	**95.84**	37.61

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

Monolingual Results. Tab. 1 presents the results of CCT-LM models compared to existing approaches, showing that CCT-LM outperform all previous models by a large margin in this monolingual setting. Thus, strong alignment enforced by CCT pretraining is not only helpful for multilingual transfer but also improves the latent space structure in general. It is important to mention, that CCT pretraining works for both encoder- and decoder-based models, improving the results.

5.1 Multi- and Cross-lingual Evaluation

For these types of evaluation on XCD we use several setups described in Sec. 3.1. Since these setups are computationally intensive we work only with encoder-based models.

Multilingual Results

The top half of Tab. 2 presents multilingual results on the proposed XCD dataset. Interestingly, knowledge transfer from the POJ-104 dataset does not improve performance, and metrics remain low. However, CCT-LM significantly outperforms others, likely due to its multilingual pretraining approach. BM25 is not evaluated in this setup, as it is unsuitable for document comparison.

For retrieval-based evaluation, CCT-LM$_{enc}$ outperforms all baselines, providing a viable solution, while GraphCodeBERT fails across all programming languages. BM25, a strong baseline for nat-

	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$^{\text{POJ}}_{\text{base}}$	0.04	0.00	0.01	0.06	0.07	0.08	0.06	0.06	0.06	0.05
CCT-LM$_{\text{enc}}$	**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$^{\text{POJ}}_{\text{base}}$	30.12	24.63	23.54	32.78	36.64	24.45	37.21	33.94	45.33	32.07
CCT-LM$_{\text{enc}}$	**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$^{\text{POJ}}_{\text{base}}$	27.10	20.04	19.44	30.98	28.37	19.70	32.89	30.08	39.98	27.62
CCT-LM$_{\text{enc}}$	**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$^{\text{POJ}}_{\text{base}}$	0.01	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
CCT-LM$_{\text{enc}}$	**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$^{\text{POJ}}_{\text{base}}$	12.83	14.75	9.33	12.78	17.16	15.94	19.53	16.01	23.88	15.80
CCT-LM$_{\text{enc}}$	**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$^{\text{POJ}}_{\text{base}}$	20.12	13.08	10.37	17.28	12.62	19.70	14.31	18.08	18.33	15.98
CCT-LM$_{\text{enc}}$	**30.83**	**22.77**	**19.32**	**32.66**	**31.64**	**20.80**	**31.59**	**40.42**	**39.40**	**29.93**

Table 2: Multilingual clone detection in two evaluation setups on the XCD dataset.

ural language information retrieval, does not work for clone detection, as it relies on identical tokens, which are often sparse even in similar code snippets.

The hybrid evaluation setup confirms these findings: BM25 remains ineffective, code language models demonstrate some knowledge transfer across solutions, and training on POJ-104 clone detection leads to a noticeable performance boost. However, CCT-LM$_{enc}$ consistently outperforms all methods, establishing a new benchmark for multilingual code-related tasks.

Cross-lingual Results. Our results in this setting are presented in the bottom half of Tab. 2. All conclusions derived for the multilingual case (above) 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 pretraining dataset.

6 Conclusion

Understanding semantic similarity is crucial for language processing, enabling solutions for various tasks in natural and programming languages. In this work, we presented CCT-LM, a new method that enhances this capability via a novel CCT pretraining approach, demonstrating its effectiveness in clone detection and code search. We introduced a novel task of multilingual clone detection and the XCD dataset for multilingual source code analysis, formalized in two evaluation setups.

The proposed CCT-LM models (encoder- and decoder-based) outperformed strong baselines in clone detection and code search. CCT-LM$_{enc}$ excelled across all setups for multi- and cross-lingual evaluation, showing that CCT pretraining improves semantic similarity understanding in language models.

We hope our method benefits other source code processing tasks, left for future work, and believe modifications of our approach could aid NLP and other machine learning fields.

7 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. 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.

Acknowledgments

The work of Sergey Nikolenko and Valentin Malykh was supported by a grant for research centers in the field of artificial intelligence, provided by the Analytical Center for the Government of the Russian Federation in accordance with the subsidy agreement (agreement identifier 000000D730321P5Q0002) and the agreement with the Ivannikov Institute for System Programming of the Russian Academy of Sciences dated November 2, 2021 No. 70-2021-00142.

References

Loubna Ben Allal, Raymond Li, Denis Kocetkov, Chenghao Mou, Christopher Akiki, Carlos Munoz Ferrandis, Niklas Muennighoff, Mayank Mishra, Alex Gu, Manan Dey, Logesh Kumar Umapathi, Carolyn Jane Anderson, Yangtian Zi, Joel Lamy Poirier, Hailey Schoelkopf, Sergey Troshin, Dmitry Abulkhanov, Manuel Romero, Michael Lappert, Francesco De Toni, Bernardo García del Río, Qian Liu, Shamik Bose, Urvashi Bhattacharyya, Terry Yue Zhuo, Ian Yu, Paulo Villegas, Marco Zocca, Sourab Mangrulkar, David Lansky, Huu Nguyen, Danish Contractor, Luis Villa, Jia Li, Dzmitry Bahdanau, Yacine Jernite, Sean Hughes, Daniel Fried, Arjun Guha, Harm de Vries, and Leandro von Werra. 2023. Santacoder: don't reach for the stars!

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

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

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

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

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. 2023. Textbooks are all you need.

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. 2021. GraphCodeBERT: Pre-training Code Representations with Data Flow. In *ICLR*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming – the rise of code intelligence.

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

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

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

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

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

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

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021a. Codexglue: A machine learning benchmark dataset for code understanding and generation.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. 2021b. 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)*.

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

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. Codegen: An open large language model for code with multi-turn program synthesis.

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

Yutaka Sasaki et al. 2007. The truth of the f-measure. 2007. *URL: https://www. cs. odu. edu/mukka/cs795sum09dm/Lecturenotes/Day3/F-measure-YS-26Oct07. pdf [accessed 2021-05-26]*, 49.

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

Johannes Villmow, Viola Campos, Adrian Ulges, and Ulrich Schwanecke. 2022. Addressing leakage in self-supervised contextualized code retrieval. *arXiv preprint*.

Wenhan Wang, Ge Li, Bo Ma, Xin Xia, and Zhi Jin. 2020. 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.

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

Ruochen Xu, Yiming Yang, Naoki Otani, and Yuexin Wu. 2018. 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. Association for Computational Linguistics.

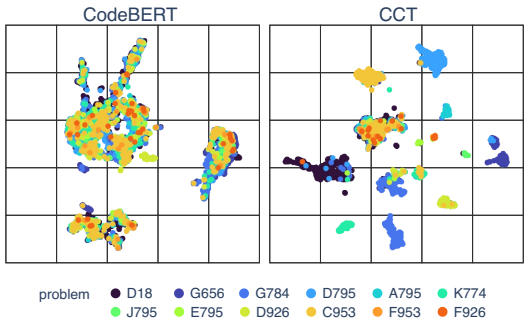
	Java	Ruby	PHP	Go	JS	Avg
$\text{CodeBERT}_{\text{base}}$	46.37	50.65	37.83	50.65	50.48	47.19
$\text{GraphCodeBERT}_{\text{base}}$	47.33	59.95	37.47	60.28	**52.04**	51.41
CCT-LM_{enc}	**48.71**	**62.25**	**42.78**	**61.44**	51.06	**53.24**

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

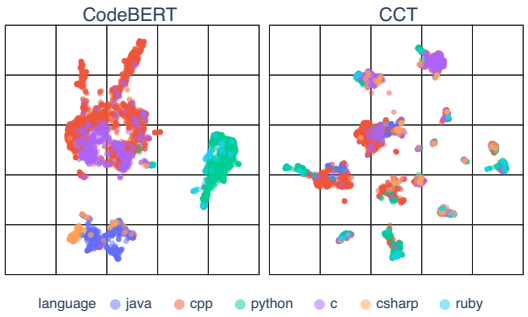
GraphCodeBERT	Clone Detection (MAP)	Code Search (MRR)	Defect detection (Acc)
Base	85.16	45.80	62.51
Base + $\mathcal{L}_{\text{XCD}}$	95.92	29.93	61.05
Base + $\mathcal{L}_{\text{XCD}}$ + $\mathcal{L}_{\text{LM}}$	95.67	47.18	63.68
Base + $\mathcal{L}_{\text{XCD}}$ + $\mathcal{L}_{\text{LM}}$	96.03	45.22	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.

Table 5: A comparison of DeepSeek-Coder 1.3b variations: clone detection on POJ-104, code search on *AdvTest*



(a) Projected embeddings of 12 coding problems.



(b) The same embeddings by programming language.

Figure 2: Sample multilingual embeddings.

A Analysis

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).

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.

Ablation Study. In this section, we study the effects of various parts of CCT. Table 5 shows the results of several DeepSeek-Coder-based models on clone detection, code search tasks. We compare the DeepSeek-Coder base model with different pretraining poolings and attention types.