# CodeNet: A Large-Scale AI for Code Dataset for Learning a Diversity of Coding Tasks

**Ruchir Puri[1], David S. Kung[1], Geert Janssen[1], Wei Zhang[1],**
**Giacomo Domeniconi[1], Vladimir Zolotov[1], Julian Dolby[1], Jie Chen[2,1],**
**Mihir Choudhury[1], Lindsey Decker[1], Veronika Thost[2,1], Luca Buratti[1],**
**Saurabh Pujar[1], Shyam Ramji[1], Ulrich Finkler[1], Susan Malaika[3], Frederick Reiss[1]**

[1]IBM Research [2]MIT-IBM Watson AI Lab [3]IBM Worldwide Ecosystems

## Abstract

Over the last several decades, software has been woven into the fabric of every aspect of our society. As software development surges and code infrastructure of enterprise applications ages, it is now more critical than ever to increase software development productivity and modernize legacy applications. Advances in deep learning and machine learning algorithms have enabled breakthroughs in computer vision, speech recognition, natural language processing and beyond, motivating researchers to leverage AI techniques to improve software development efficiency. Thus, the fast-emerging research area of "AI for Code" has garnered new interest and gathered momentum. In this paper, we present a large-scale dataset *CodeNet*, consisting of over 14 million code samples and about 500 million lines of code in 55 different programming languages, which is aimed at teaching AI to code. In addition to its large scale, CodeNet has a rich set of high-quality annotations to benchmark and help accelerate research in AI techniques for a variety of critical coding tasks, including code similarity and classification, code translation between a large variety of programming languages, and code performance (runtime and memory) improvement techniques. Additionally, CodeNet provides sample input and output test sets for 98.5% of the code samples, which can be used as an oracle for determining code correctness and potentially guide reinforcement learning for code quality improvements. As a usability feature, we provide several pre-processing tools in CodeNet to transform source code into representations that can be readily used as inputs into machine learning models. Results of code classification and code similarity experiments using the CodeNet dataset are provided as a reference. We hope that the scale, diversity and rich, high-quality annotations of CodeNet will offer unprecedented research opportunities at the intersection of AI and Software Engineering.

## 1 Introduction

There is a growing trend towards leveraging AI for building tools that support software engineering and development [1, 2]. AI can manipulate and generate computer code, but can it do so with high quality? Many researchers are fascinated by this possibility, encouraged by AI successes in other domains and tantalized by the vision of computers programming computers. Some recent deep-learning models [3, 4] for code have received a lot of publicity: trained on vast amounts of data and using novel architectures with billions of parameters, they sometimes generate surprisingly plausible code.

Given the success of non-AI tools for code, why should we consider AI to augment or possibly replace them? Firstly, AI can help refine and re-tune the heuristics used by traditional coding tools. Secondly, based on the training data from past experience, AI can help prioritize when there is more than one sound answer [5]. Thirdly, an AI-based tool may handle incomplete or invalid code more

35th Conference on Neural Information Processing Systems (NeurIPS 2021) Track on Datasets and Benchmarks.

robustly, thus expanding its scope. Finally, AI can incorporate signals usually ignored by traditional tools for code, such as the natural language in identifiers or comments.

In the enterprise environment, developers often face code written by large teams over many years and geographies. Developers must manipulate such code to modernize it, fix bugs, improve its performance, evolve it when requirements change, make it more secure, and/or comply with regulations. These tasks are challenging, and it is crucial to provide tool support for developers to be more productive at performing them. It is well known that the latest advancements in deep learning algorithms rely on best-of-breed datasets, such as ImageNet, to create increasingly complex and powerful models. In this paper, we present "CodeNet", a first-of-its-kind dataset in scale, diversity, and quality, to accelerate the algorithmic advances in AI for Code.

To promote widespread adoption of CodeNet, we will be launching contests involving use cases based on the dataset. The first contest [6] will focus on diversity, inclusion and spurring interest among aspiring data scientists. We are partnering with the Global Women in Data Science organization (with presence in over 50 countries) founded by Stanford University [7] and targeting teams with at least fifty percent women. We are planning follow-up contests that target experienced AI practitioners.

The rest of the paper is organized as follows. Section 2 introduces the CodeNet dataset. Related datasets are discussed in Section 3, and the differentiation of CodeNet with respect to these related datasets is elaborated in Section 4. Section 5 describes how CodeNet was curated and Section 6 enumerates the usability features of CodeNet with several pre-processing tools to transform source codes into representations that can be readily used as inputs into machine learning models. Section 7 discusses the upcoming CodeNet contest and Section 8 describes important baseline experiments with the CodeNet dataset. Section 9 presents further uses of the CodeNet dataset and Section 10 concludes the paper.

## 2 The CodeNet Dataset

The CodeNet dataset consists of a large collection of code samples with extensive metadata. It also contains documented tools to transform code samples into intermediate representations and to access the dataset and make tailored selections. Our goal is to provide the community with a large, high-quality curated dataset that can be used to advance AI techniques for source code.

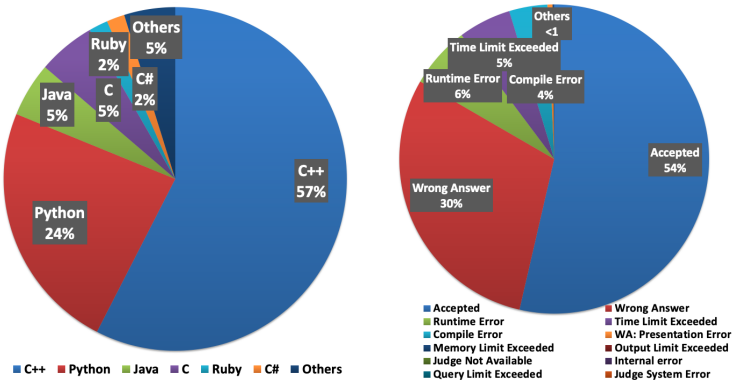
CodeNet is derived from the data available on two online judge websites: AIZU [8] and AtCoder [9]. Online judge websites pose programming problems in the form of courses and contests. The dataset consists of submissions to these problems, which are judged by an automated review process for correctness. Problem descriptions, submission outcomes, and associated metadata are available via various REST APIs.

**Scale and Statistics.** CodeNet contains a total of 13,916,868 submissions, divided into 4053 problems. Among the submissions, 53.6% (7,460,588) are accepted (compilable and pass the prescribed tests), 29.5% are marked with wrong answer, and the remaining rejected due to their failure to meet run time or memory requirements. To our knowledge, this is the largest dataset so far among similar kinds. Submissions are in 55 different languages; 95% of them are coded in C++, Python, Java, C, Ruby, and C#. C++ is the most common language, with 8,008,527 submissions (57% of the total), of which 4,353,049 are accepted. With the abundance of code samples, users can extract large benchmark datasets that are customized to their downstream use. See Figure 1 for a summary.

**Diversity.** The problems in CodeNet are mainly pedagogical and range from elementary exercises to sophisticated problems that require advanced algorithms. The submitters range from beginners to experienced coders. Some submissions are correct while others contain different types of errors, accordingly labeled. The submissions are in many different languages.

**Code Samples.** Each code sample is a single file and includes inputting the test cases and printing out the computed results. The file name uses standard extensions that denote the programming language, e.g., `.py` for Python. The majority of code samples contain only one function, although submissions to more complex problems might have several functions.

**Metadata.** The metadata enables data queries and selections among the large collection of problems, languages, and source files. The metadata is organized in a two level hierarchy. The first is the dataset level, which describes all problems. The second is the problem level, which details all the submissions to a single problem. Metadata and data are separated in the dataset structure.




(a) Languages (b) Status

Figure 1: Percentage of submissions per language (left) and per status (right).

At the dataset level, a single CSV file lists all problems and their origins, along with the CPU time and memory limits set for them. Additionally, every problem has an HTML file with a detailed description of the problem, the requirements and constraints, and the IO examples.

At the problem level, every problem has a CSV file. The metadata for each submission is summarized in Table 8 in the supplement, which lists the fields contained in each CSV file as well as the corresponding descriptions.

**Limitations.** All code samples in CodeNet may not be extensively commented, and these comments may be in multitude of languages. Therefore, AI techniques that rely on learning from preponderance of comments in the code may face challenges. The code samples are solutions to high-school and beginning college level programming problems. This dataset is not suitable for users looking for code with enterprise API's and advanced design patterns.

## 3 Related Datasets

A wide variety of datasets for source code exist, with many targeting one or a small number of tasks. Such tasks include clone detection, vulnerability detection [10, 11], cloze test [12], code completion [13, 14], code repair [15], code-to-code translation, natural language code search [16], text-to-code generation [17], and code summarization [16]. A detailed discussion of several of these tasks and their respective datasets is available in CodeXGLUE [18], which is a collection of existing datasets. CodeNet, on the other hand, is a new dataset curated from scratch, that aims to support a broad set of use cases. Popular datasets of a similar kind are POJ-104 [19] (which is incorporated as part of CodeXGLUE as well) and GCJ [20] (derived from Google Code Jam). We compare CodeNet to these datasets in the following.

### 3.1 POJ-104

POJ-104 was collected from a pedagogical online judge system. The code samples are submissions to 104 programming problems. With 500 submissions to each problem, there is a total of 52,000 code samples in the dataset. This dataset has been used by many authors for code classification [19] and code similarity [21].

POJ-104 is faced with several limitations.

1. The code samples are in C and C++, but the two languages are not distinguished. Although they are closely related, mixing them leads to parsing errors and a reduction of useful code samples [21].
2. Useful metadata such as the results of the judging system (acceptance, error types etc.) are missing. Therefore, for certain applications where compilabilty or code correctness is important, additional pre-processing efforts are needed and useful code samples are reduced [21]. The dataset does not contain the problem statement, although some example problems are described in [22], and information on how to execute the code samples is absent.
3. Some problems are identical (e.g., problems 26 and 62), and some submissions are near duplicates of each other, although the percentage of such cases is low compared to other datasets.

### 3.2 GCJ

GCJ [20] was collected from the submissions to the Google Code Jam competitions from 2008 to 2020. Similar to CodeNet, the submissions cover a wide variety of programming languages, with C++, Java, Python, and C being the predominant ones. The C++ subset has been extracted into a POJ-104-like benchmark and used in some publications. This benchmark dataset, GCJ-297 [23], has 297 problems and approximately 280K submissions. The number of submissions is imbalanced among problems.

GCJ is advantageous over POJ-104 in size and language diversity, but we believe that an even larger dataset such as CodeNet can better serve the community. GCJ contains neither metadata nor information on identical problems and near duplicates.

## 4 CodeNet Differentiation

Table 1: Related datasets comparison

| | CodeNet | GCJ | POJ |
|---|---|---|---|
| Total number of problems | 4053 | 332 | 104 |
| Number of programming languages | 55 | 20 | 2 |
| Total number of code samples | 13,916,828 | 2,430,000 | 52,000 |
| C++/C subset data size (code samples) | 8,008,527 | 280,000 | 52,000 |
| Percentage of problems with test data | 51% | 0% | 0% |
| Task: Memory Consumption Prediction | Yes | No | No |
| Task: Runtime Performance Comparison | Yes | No | No |
| Task: Error Prediction | Yes | No | No |
| Task: Near duplicate prediction | Yes | No | No |

A high quality code dataset has certain desired properties. We constructed CodeNet according to these requirements. In the following, we discuss how CodeNet differentiates itself from the existing datasets along these lines. Table 1 is a comparison with related datasets.

**Large scale.** A useful dataset should contain a large number and variety of data samples to expose the realistic and complex landscape of data distributions one meets in practice. CodeNet is the largest dataset in its class - it has approximately 10 times more code samples than GCJ and its C++ benchmark is approximately 10 times larger than POJ-104.

**Rich annotation.** For the dataset class in question, it is important to include information beyond which problem a code sample solves to enable a wide range of applications and use cases. It is useful to know whether a code sample solves the problem correctly, and if not, the error category (e.g., compilation error, runtime error, and out-of-memory error). Since the source code is supposed to solve a programming problem, it is advantageous to know the problem statement and have a sample input for execution and a sample output for validation. All such extra information is part of CodeNet but absent in GCJ and POJ-104.

**Clean samples.** For effective machine learning, the data samples are expected to be independent and identically distributed (iid); otherwise, the resulting performance metric could be significantly inflated [24]. The existence of duplicate and/or near duplicate code samples makes the iid assumption dubious. Hence, it is crucial to identify the near duplicates. The presence of identical problems in the dataset poses an even bigger issue. In CodeNet, we analyzed the code samples for (near) duplication and used clustering to find identical problems. While this process does not make our dataset satisfy the iid property, providing this information as part of the dataset release allows more flexibility for the users to customize benchmarks for their specific use cases. The near-duplicate information is not available in GCJ and POJ-104.

## 5 Construction of CodeNet

### 5.1 Collection of Code Samples

The CodeNet dataset contains problems, submissions, and metadata, scraped from the AIZU and AtCoder online judging systems. For AIZU, we used the provided REST APIs to download all the

metadata. For AtCoder, due to the absence of a REST API, we scraped the problems, submissions, and metadata directly from the web pages. We considered only public and non-empty submissions that did not contain errors or inconsistencies in the metadata. We manually merged the information from the two sources and adopted a unified format to create a single dataset.

### 5.2 Cleansing

Because data are collected from different sources, we apply a consistent character encoding (UTF-8) on all raw data files. Additionally, we remove byte-order marks and use Unix-style line-feeds as the line ending.

As indicated in section 4, we identify near-duplicates. We follow Allamanis [24] and use Jaccard similarity [25] as a metric to score code pairs. Each code sample is tokenized and stored as a bag-of-tokens multiset. In our case, we keep all tokens except comments and preprocessor directives. We compute the set and multiset Jaccard indices and respectively use 0.9 and 0.8 as the near-duplicate thresholds.

Besides similar code samples, identical problems are also likely because they have been gathered over many decades. We go through the problem description files (in HTML format) and apply `fdupes` to extract identical problem pairs. Additionally, using the near-duplicate information calculated for code samples, we consider a problem pair to be a potential duplicate when the number of near-duplicate code pairs exceeds a threshold. Clustering of duplicate problems is illustrated by the graphs in Figure 2, where each node denotes a problem and an edge between two nodes is labeled by the number of near-duplicate code pairs. Each connected graph is then a cluster of potential duplicate problems and we manually inspect the problem descriptions to verify the correctness of this duplicate detection.

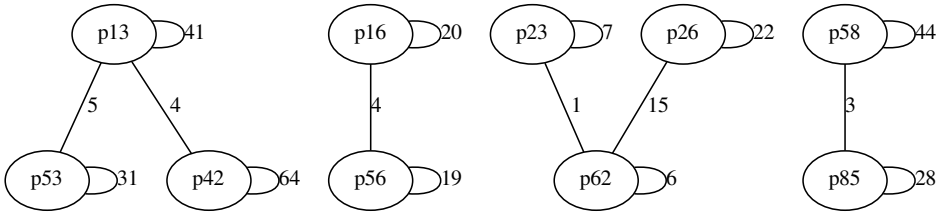



Figure 2: An example of a near-duplicate problem graph.

### 5.3 Benchmark Datasets

CodeNet has a rich set of code samples, and the user can assemble a customized benchmark according to his/her need. Following POJ-104, we extracted benchmark datasets from CodeNet in C++, Python, and Java. The benchmark characteristics are shown in Table 2. For the C++ benchmarks, the number of problems and their solutions are chosen to make the benchmark challenging. The benchmarks are filtered in the following ways. Each code sample is "unique" in the sense that it is not a near-duplicate of another code sample. The same is true of each problem. Samples with a large fraction of dead code are excluded. Each code sample has successfully passed through the tokenizer, the SPT generator, and the graph generator, all described in the next section. This step is to ensure that proper processing can be done to convert a code sample to a machine learning model input.

## 6 Code Representation and Tools

Machine learning with source code requires proper abstractions of the code. The abstractions are instantiated as representations in specific formats. As a usability feature, we provide several pre-processing tools to transform source codes into representations that can readily be used as inputs into machine learning models. They are described as follows.

**Tokenizer.** We offer fast C implementations of tokenizers for C, C++, Java, Python, and JavaScript. Additionally, the parse-tree generator described next can also produce token streams for C, C++, Java, and Python and can easily be extended to more languages.

**Simplified Parse Tree (SPT)** Simplified parse trees are derived from parse trees generated using ANTLR4 [26]. We traverse the ANTLR4 parse tree and remove internal nodes that only have one child. By doing so, we maintain the essential structure of the parse tree while pruning out unnecessary parser production rules. Finally, we adopt Aroma's [27] naming convention: leaf nodes are named by their literal strings and internal nodes are named by a concatenation of their children's names (only reserved words are kept while others are replaced by a hash mark #). We produce features for each node: (1) node type (token or parsing rule); (2) token type (e.g., an identifier), when applicable; (3) parsing rule type (e.g., an expression), when applicable; and (4) whether it is a reserved word. We adopt an extensible JSON graph schema so that edges can be augmented with types when needed. Currently, we support generating SPTs for four languages: C, C++, Java, and Python. Table 2 summarizes the SPT statistics for the four benchmarks.

Table 2: Benchmark statistics.

| | C++1000 | C++1400 | Python800 | Java250 |
|---|---|---|---|---|
| #problems | 1,000 | 1,400 | 800 | 250 |
| #samples | 500,000 | 420,000 | 240,000 | 75,000 |
| #SPT-nodes | 188,449,294 | 198,258,050 | 55,744,550 | 25,449,640 |
| #SPT-edges | 187,949,294 | 197,838,050 | 55,504,550 | 25,374,640 |

**Code graphs.** We augment the tool chain with a code graph generator using WALA [28], a general framework for program analysis. The backbone of a code graph is a system dependence graph, which is an inter-procedural graph of program instructions (e.g. call, read) expressing control flow and data flow information as edges. We also generate inter-procedural control flow graphs, which are control flow graphs of all the methods in the program, stitched together to connect call sites with target methods. Our code graph tool currently supports only Java and Python, but we plan to support more languages such as Javascript.

## 7 CodeNet Challenge

The launch of CodeNet was well received by the AI community and the media, with coverage from Forbes[29], VentureBeat[30], ZDNet[31] and others. Within a short span of 3 months, our github received 1000 stars and has been forked over 119 times. Our vision is to use CodeNet as an umbrella to curate AI for code datasets for widespread adoption and to drive innovation in AI for code. To leverage the momentum of CodeNet, we will be launching CodeNet challenges to create excitement in the AI community. The first contest [6] is mainly pedagogical and targets aspiring data scientists. In addition, we are partnering with the Global Women in Data Science organization (with presence in over 50 countries) founded by Stanford University [7] to emphasize diversity and inclusion (teams must have at least fifty percent women). We will organize workshops to introduce the topic, code similarity, and provide educational materials. This contest will be kicked off in late September and the winner will be announced in early December, around the NeurIPS2021 time frame. The conclusion of the first contest will be followed by a contest that will target experienced AI practitioners. Potential contest topics will revolve around practical and compelling use cases such as code language translation, code repair, code performance improvement, and code memory reduction.

## 8 Experiments with the CodeNet Dataset

In this section, we report the results of a code classification task, a similarity task, a generalization task, and a token inference task, using the four benchmark datasets (see Table 2) extracted from CodeNet. For this paper, these experiments are not meant to achieve the best-of-breed results using the state of the art. Our intention is to provide a set of baseline results as a reference. The experiments are typically performed on a Xeon machine using P100 or V100 GPUs. Details of the experiments are in appendices D, E, and F and their code and scripts are in the model-experiments folder of the CodeNet repository [32], when third party licenses allow.

### 8.1 Code Classification

In the classification task, each problem corresponds to a class: a code sample belongs to a class if it is a submission to the corresponding problem. For each experiment, 20% of the code samples are used for testing, while the rest are split in 4:1 for training and validation, respectively. We experiment with a diverse set of machine learning methods: bag of tokens, sequence of tokens, BERT model, and graph neural networks (GNNs).

1. **MLP with bag of tokens.** A code sample is represented by a vector of relative frequencies of token occurrences. Only operator and keyword tokens are used. The model is a 3-layer multilayer perceptron (MLP).
2. **CNN with token sequence.** We use the same set of tokens as above but retain their order to form a sequence. All sequences have the same length under zero padding. The classification model is a convolutional neural network (CNN) with an initial token embedding layer.
3. **C-BERT with token sequence.** Treating a code sample as a piece of natural language text, we build a C-BERT model [33] through pretraining on 10K top starred Github projects written in C. We use the Clang C tokenizer and Sentencepiece to tokenize each code sample. The pretrained model is fine-tuned on each benchmark.
4. **GNN with SPT.** Based on the parse tree representation, we use graph convolutional networks (GCN) [34] and graph isomorphism networks (GIN) [35] as well as their variants as the prediction model. The variant adds a virtual node to the graph to enhance graph message passing [36].
5. **GNN with Code Graph.** We also apply GCN on the code graph representation of the code.

Table 3: Classification accuracy (in %).

| | Java250 | Python800 | C++1000 | C++1400 |
|---|---|---|---|---|
| MLP w/ bag of tokens | 71.00±0.29 | 67.80±0.15 | 68.26±0.21 | 64.50±0.13 |
| CNN w/ token sequence | 89.52±0.59 | 87.46±0.25 | 93.96±0.18 | 93.71±0.18 |
| C-BERT | 97.40±0.19 | 97.09±0.18 | 93.79±0.01 | 91.83±0.06 |
| GNN (GCN) | 92.70±0.25 | 93.82±0.16 | 95.76±0.12 | 95.26±0.13 |
| GNN (GCN-V) | 93.02±0.81 | 94.30±0.15 | 96.09±0.17 | 95.73±0.07 |
| GNN (GIN) | 93.26±0.23 | 94.17±0.19 | 96.34±0.15 | 95.95±0.13 |
| GNN (GIN-V) | 92.77±0.66 | 94.54±0.12 | 96.64±0.10 | 96.36±0.10 |
| Code Graph+GCN | 94.10±.001 | 87.80±.007 | N/A | N/A |

Table 3 summarizes the classification accuracy for all models on all benchmarks. Despite the simplicity of bag of tokens, it achieves well over 60% accuracy. Maintaining token ordering, CNN with token sequence offers significant improvement, reaching approximately 90% across all benchmarks.

More complex neural models sometimes further improve the prediction performance, as witnessed by C-BERT, which reaches approximately 97% for both Java and Python. It is interesting to note that even though C-BERT is pre-trained with C programs, its performance on the two C++ benchmarks is less impressive. We speculate that such a lower performance is related to programming practices. For C++, it is common to have identical program construction, such as declaration of constants (e.g., pi and epsilon) and data structures, appear across C++ submissions to different problems, but such a practice is rare in Java and Python.

Overall, the GNN models exhibit competitive performance. They are consistently the top performers, if not the best. The code graph representation slightly improves over the SPT representation on Java, but performs less well on Python.

### 8.2 Code Similarity

In the similarity task, two pieces of code samples are considered similar if they solve the same problem (type-4 similarity in [37]). Note that textual similarity does not guarantee similarity in functionality. For example, programs that differ by only one token might behave very differently; hence, they are not considered similar. For the token-based experiments, we treat the problem as binary classification. We use the same training, validation and testing split as in classification. Code pairs are randomly sampled within each subset. The number of similar pairs is the same as dissimilar ones. For the SPT representation, we experiment with several popular techniques, including AROMA [27], MISIM [21], and GMN [38]. The following contains more details about the models and methods.

1. **MLP with bag of tokens.** This model is the same as the one for code classification, except that the input is a concatenation of the two bag-of-tokens vectors from each program.
2. **Siamese network with token sequence.** The token sequence is the same as the one for code classification. The model is a Siamese network with two CNNs with shared weights.

3. **SPT with handcrafted feature extraction:** The method AROMA [27] uses normalized SPT node names and handcrafted rules to extract feature vectors for each SPT. Then, similarity is computed as a dot product of the extracted feature vectors.
4. **GNN with SPT:** With the same SPT, on the other hand, MISIM [21] uses a graph neural network to extract high-level features, and uses the cosine similarity of the extracted features to compute similarity. Additionally, we apply graph matching network (GMN) [38], which uses a cross-graph attention mechanism to learn pair-wise structural similarity of graphs, on the SPT pairs to predict similarity. The implementation is adapted from [39].

Table 4: Similarity accuracy (in %).

| | Java250 | Python800 | C++1000 | C++1400 |
|---|---|---|---|---|
| MLP w/ bag of tokens | 81.80±0.06 | 86.61±0.08 | 85.82±0.05 | 86.54±0.07 |
| Siamese w/ token sequence | 89.70±0.18 | 94.67±0.12 | 96.19±0.08 | 96.56±0.07 |

Table 4 summarizes the classification accuracy for the first two models. The performance of bag of tokens is modest, considering that the problem is a binary classification with perfectly balanced classes. On the other hand, the Siamese model significantly outperforms bag of tokens, as expected.

Table 5: Similarity MAP@R score.

| | Java250 | Python800 | C++1000 | C++1400 |
|---|---|---|---|---|
| Rule-based w/ SPT (AROMA) | 0.19 | 0.19 | 0.17 | 0.15 |
| GNN w/ SPT (MISIM) | 0.64±0.007 | 0.65±0.003 | 0.78±0.005 | 0.77±0.002 |

Table 5 summarizes the MAP@R [40] score for two SPT-based approaches with solutions for 50% problems used for training, 25% for validation, and 25% for test. MISIM GNN model is trained for 1000 epochs. AROMA results in a relatively low score because the feature extraction is rule-based and no model is learned, whereas MISIM uses a neural network to extract features through supervised training.

Table 6: Similarity MAP@R score on Java250.

| | (p4, s5) | (p3, s300) | (p10, s300) |
|---|---|---|---|
| GNN w/ SPT (MISIM, structure only) | 0.472±0.023 | 0.194±0.010 | 0.096±0.009 |
| GNN w/ SPT (GMN, structure only) | 0.679±0.056 | 0.432±0.035 | 0.256±0.015 |
| GNN w/ SPT (GMN + MISIM node attributes) | 0.985±0.015 | 0.794±0.036 | 0.780±0.026 |

Exploring further into the Java250 benchmark, Table 6 summarizes the MAP@R score with a variety of test sets: (p4, s5), (p3, s300), and (p10, s300), indicating 4, 3, and 10 problems with 5, 300 and 300 solutions each respectively. Across all test sets, GMN outperforms MISIM if both are trained with only the SPT structure; when combined with MISIM node attributes, GMN further improves the score significantly.

### 8.3 Generalization Across Datasets

Models trained on the CodeNet benchmark datasets can benefit greatly from their high quality. To demonstrate this, we compare C++1000 to one of the largest publicly available datasets of its kind, GCJ-297 [23]. For the purpose of this comparison, we train the same MISIM model on C++1000 and GCJ-297 and test the two trained models on a third, independent dataset - POJ-104. The result of this comparison is plotted in Figure 3.

The $x$-axis of this plot is the number of training epochs used and the $y$-axis is the MAP@R score. The MISIM model for both datasets is trained for $500$ epochs and the MAP@R score for validation and test is computed after every ten epochs. There are a total of four curves - a validation and a test curve for GCJ-297 and a validation and a test curve for C++1000.

The training curves show that a $10\%$ higher validation score can be achieved with GCJ-297 compared to C++1000. However, when tested on POJ-104, the model trained on GCJ-297 achieves a $12\%$ lower score compared to the model trained on C++1000. We believe C++1000 has better generalization than GCJ-297 mainly for two reasons: i) high data bias in GCJ-297 because the top $20$ problems with the most number of submissions account for $50\%$ of all submissions and ii) cleaning and de-duplication of submissions in CodeNet dataset (as described in Section 5.2).

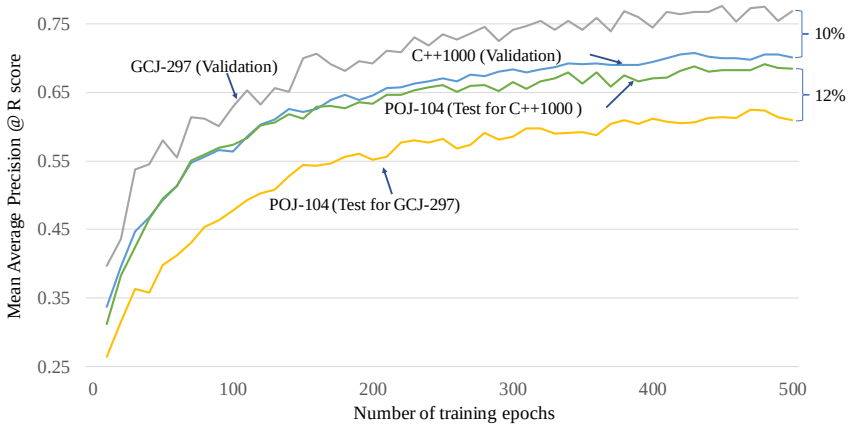



Figure 3: Test score on POJ-104 is 12% higher when a model is trained on C++1000 as compared to a model trained on GCJ-297, even though the validation score for GCJ-297 model is 10% higher than the validation score for C++1000 model.

### 8.4 Masked Language Modelling for Token Inference

A task such as code completion relies on the ability to predict a token at a certain position in a sequence. To accomplish this we can build a masked language model (MLM) using a technique that randomly masks out tokens in an input sequence and aims to correctly predict them in an as-yet-unseen test set. We train a popular BERT-like attention model on the C++1000 CodeNet benchmark after tokenization to a vocabulary of over 400 tokens and obtain a top-1 prediction accuracy of 0.9104 (stddev: 0.002) and a top-5 accuracy of 0.9935 (stddev: 0.0005).

## 9 Further Uses of CodeNet

The rich metadata and language diversity open CodeNet to a plethora of use cases. The problem-submission relationship in CodeNet corresponds to type-4 similarity [37] and can be used for code search and clone detection. The code samples in CodeNet are labeled with their acceptance status so we can readily extract pairs of buggy and fixed code for code repair [41, 42]. A large number of code samples come with inputs so that we can execute the code to extract the CPU run time and memory footprint, which can be used for regression studies and prediction.

CodeNet may also be used for program translation, given its wealth of programs written in a multitude of languages. Translation between two programming languages is born out of a practical need to port legacy codebases to modern languages in order to increase accessibility and lower maintenance costs. With the help of neural networks, machine translation models developed for natural languages [43] were adapted to programming languages, producing pivotal success [4]. One considerable challenge of neural machine translation is that model training depends on large, parallel corpora that are expensive to curate [44], especially for low-resource languages (e.g., legacy code). Recently, monolingual approaches [45, 4] were developed to mitigate the reliance on parallel data, paving ways to build models for languages with little translation. Compared with current popular data sets (e.g., [4, 46]), CodeNet covers a much richer set of languages with ample training instances.

## 10 Conclusion

Artificial intelligence has made great strides in understanding human language. Computer scientists have been fascinated by the possibility and tantalized by the vision of computers (AI) programming computers. In this paper, we presented "CodeNet", a first-of-its-kind very large-scale, diverse and high-quality dataset to accelerate the algorithmic advances in AI for Code. This dataset is not only unique in its scale, but also in the diversity of coding tasks it can help benchmark: from code similarity and classification for advances in code recommendation algorithms, and code translation between a large variety of programming languages, to advances in code performance improvement techniques. We hope that the scale, diversity and rich, high-quality annotations of CodeNet will offer unprecedented research opportunities at the intersection of AI and Software Engineering.

## 11 Acknowledgements

We would like to acknowledge AIZU and AtCoder for making the code submissions publicly available. We would like to thank the IBM Data Asset eXchange team for providing a platform to host the CodeNet dataset. We would like to thank the Women in Data Science team at Stanford University and the IBM Call for Code team for their collaboration in launching the CodeNet challenge.

## 12 Bibliography

[1] Miltiadis Allamanis, Earl T Barr, Premkumar Devanbu, and Charles Sutton. A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR)*, 51(4):1–37, 2018.

[2] Yanming Yang, Xin Xia, David Lo, and John Grundy. A survey on deep learning for software engineering. *arXiv preprint arXiv:2011.14597*, 2020.

[3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

[4] Marie-Anne Lachaux, Baptiste Roziere, Lowik Chanussot, and Guillaume Lample. Unsupervised translation of programming languages. In *NeurIPS*, 2020.

[5] Zheng Wang and Michael O'Boyle. Machine learning in compiler optimization. *Proceedings of the IEEE*, 106(11):1879–1901, 2018.

[6] `http://ibm.biz/cfcsc-codenet`.

[7] Women in data science. `https://widsconference.org/`.

[8] Yutaka Watanobe. Aizu online judge. `https://onlinejudge.u-aizu.ac.jp`.

[9] Atcoder. `https://atcoder.jp/`.

[10] Yunhui Zheng, Saurabh Pujar, Burn Lewis, Luca Buratti, Edward Epstein, Bo Yang, Jim Laredo, Alessandro Morari, and Zhong Su. D2a: A dataset built for ai-based vulnerability detection methods using differential analysis. In *Proceedings of the ACM/IEEE 43rd International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '21, New York, NY, USA, 2021. Association for Computing Machinery.

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

[12] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gonga, Linjun Shou, Bing Qin, Ting Liu, and Daxin Jiang. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155v4*, 2020.

[13] Miltiadis Allamanis and Charles Sutton. Mining source code repositories at massive scale using language modeling. In *10th Working Conference on Mining Software Repositories (MSR)*, page 207–216. IEEE, 2013.

[14] Veselin Raychev, Pavol Bielik, and Martin Vechev. Probabilistic model for code with decision trees. *ACM SIGPLAN Notices*, 2016.

[15] Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. An empirical study on learning bug-fixing patches in the wild via neural machine translation. In *ACM Transactions on Software Engineering and Methodology (TOSEM)*, pages 1–29, 2019.

[16] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436v3*, 2019.

[17] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Mapping language to code in programmatic context. *arXiv preprint arXiv:1808.09588*, 2018.

[18] 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. Codexglue: A machine learning benchmark dataset for code understanding and generation, 2021.

[19] Tal Ben-Nun, Alice Shoshana Jakobovits, and Torsten Hoefler. Neural code comprehension: A learnable representation of code semantics. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 3588–3600. Curran Associates, Inc., 2018.

[20] Farhan Ullah, Hamad Naeem, Sohail Jabbar, Shehzad Khalid, Muhammad Ahsan Latif, Fadi Al-turjman, and Leonardo Mostarda. Cyber security threats detection in internet of things using deep learning approach. *IEEE Access*, 7:124379–124389, 2019.

[21] Fangke Ye, Shengtian Zhou, Anand Venkat, Ryan Marcus, Nesime Tatbul, Jesmin Jahan Tithi, Niranjan Hasabnis, Paul Petersen, Mattson. Timothy, Tim Kraska, Pradeep Dubey, Vivek Sarkar, and Justin Gottschlich. Misim: A novel code similarity system, 2021.

[22] `https://sites.google.com/site/treebasedcnn/home/problemdescription`.

[23] gcj-dataset. `https://openreview.net/attachment?id=AZ4vmLoJft&name=supplementary_material`.

[24] Miltiadis Allamanis. The adverse effects of code duplication in machine learning models of code. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2019, page 143–153, New York, NY, USA, 2019. Association for Computing Machinery.

[25] Wikipedia. Jaccard index — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Jaccard_index`, 2020.

[26] Terence Parr. *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf, 2nd edition, 2013.

[27] Sifei Luan, Di Yang, Celeste Barnaby, Koushik Sen, and Satish Chandra. Aroma: code recommendation via structural code search. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):1–28, Oct 2019.

[28] IBM T.J. Watson Research Center. Wala. `https://github.com/wala/WALA`, 2021.

[29] Forbes on codenet. `https://www.forbes.com/sites/moorinsights/2021/06/04/ibm-codenet-artificial-intelligence-that-can-program-computers-and-solve-a-100-billion-legacy-code-problem/?sh=343813636cdc`.

[30] Venturebeat on codenet. `https://venturebeat.com/2021/05/10/ibms-codenet-dataset-aims-to-train-ai-to-tackle-programming-challenges/`.

[31] Zdnet on codenet. `https://www.zdnet.com/article/ibm-launches-autosql-watson-orchestrate-codenet-enterprise-ai-tools-at-think/`.

[32] Project codenet repository. `https://github.com/IBM/Project_CodeNet`.

[33] Luca Buratti, Saurabh Pujar, Mihaela Bornea, Scott McCarley, Yunhui Zheng, Gaetano Rossiello, Alessandro Morari, Jim Laredo, Veronika Thost, Yufan Zhuang, and Giacomo Domeniconi. Exploring software naturalness through neural language models, 2020.

[34] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[35] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[36] Veronika Thost and Jie Chen. Directed acyclic graph neural networks. In *ICLR*, 2021.

[37] Hitesh Sajnani. *Large-Scale Code Clone Detection*. PhD thesis, University of California, Irvine, 2016.

[38] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching network for learning the similarity of graph structured objects. In *International Conference on Machine Learning (ICML)*, 2019.

[39] Graph-matching-networks. `https://github.com/Lin-Yijie/Graph-Matching-Networks`.

[40] Kevin Musgrave, Serge J. Belongie, and Ser-Nam Lim. A metric learning reality check. *CoRR*, abs/2003.08505, 2020.

[41] Zimin Chen, Steve Kommrusch, Michele Tufano, Louis-Noël Pouchet, Denys Poshyvanyk, and Martin Monperrus. Sequencer: Sequence-to-sequence learning for end-to-end program repair. *IEEE Transaction on Software Engineering*, 2019.

[42] Michihiro Yasunaga and Percy Liang. Break-it-fix-it: Unsupervised learning for program repair. In *International Conference on Machine Learning (ICML)*, 2021.

[43] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. Preprint arXiv:1609.08144, 2016.

[44] Xinyun Chen, Chang Liu, and Dawn Song. Tree-to-tree neural networks for program translation. In *NeurIPS*, 2018.

[45] Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. Unsupervised machine translation using monolingual corpora only. In *ICLR*, 2018.

[46] 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. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. Preprint arXiv:2102.04664, 2021.

[47] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. 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. Codexglue: A machine learning benchmark dataset for code understanding and generation. *CoRR*, abs/2102.04664, 2021.

[48] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning, 2018.

[49] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

[50] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[51] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[52] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019.

[53] Codenet dataset. `https://developer.ibm.com/exchanges/data/all/project-codenet`.

[54] Ankur Singh. "end-to-end masked language modeling with bert". `https://keras.io/examples/nlp/masked_language_modeling`.

## Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] Please see Sections 2, 3, and 4.
   (b) Did you describe the limitations of your work? [Yes] Please see the paragraph on **Limitations** in Section 2.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] The CodeNet dataset is about code written for pedagogical purposes. We believe that it does not have any negative societal impact. On the contrary, we are launching a challenge/contest based on the CodeNet dataset with the Global Women in Data Science organization with presence in over 50 countries to promote diversity, inclusion and data science education in the field of AI for Code.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] The paper presents a dataset with code samples submitted by students to simple programming problems. The dataset neither does harm to living beings, nor raise any security and economic concerns, human rights and surveillance issues, nor damage the environment, nor deceive people and damage their livelihood. We have anonymized each submitter's user id, and tried filtering offensive words. We have also followed the term of service of the website from which we download the dataset.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments (e.g. for benchmarks)...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The source code and instructions of the experiments are available in the model-experiments folder at `https://github.com/IBM/Project_CodeNet`, when third-party licenses allow. The datasets used in the experiments are available in `https://developer.ibm.com/technologies/artificial-intelligence/data/project-codenet`.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Please see Section 8 and appendix D, appendix E, and appendix F in the supplementary materials.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Please see Section 8.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Please see Section 8 and appendix D, appendix E, and appendix F in the supplementary materials.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A] Our work is creating/releasing new assets
   (b) Did you mention the license of the assets? [Yes] The license of the dataset is CDLA Permissive v2.0. It is mentioned in `https://developer.ibm.com/technologies/artificial-intelligence/data/project-codenet` and `https://www.linuxfoundation.org/press-release/enabling-easier-collaboration-on-open-data-for-ai-and-ml-with-cdla-permissive-20/`.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] The new assets are available at `https://developer.ibm.com/technologies/artificial-intelligence/data/project-codenet`.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] We have looked into the terms of service, and ensured that the code samples can be used for research purposes and we also contacted the respective communities.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] Please see Section 2. We have anonymized the user ids in the submissions. The data samples are computer code for solving context problems and in principle should not have any offensive content. While we cannot guarantee that there is no personal information (e.g. name of a person) and potential offensive content, but we have made every possible effort to minimize any such possibility.

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] We did not use crowdsourcing or conduct research with human subjects

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] We did not use crowdsourcing or conduct research with human subjects

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] We did not use crowdsourcing or conduct research with human subjects

# A Additional details about CodeNet

## A.1 URL

`https://developer.ibm.com/technologies/artificial-intelligence/data/project-codenet/` is the landing page of the dataset. It contains links to download the full dataset and the benchmarks datasets (similar to POJ-104) in C++, Python and Java, which users can use to perform similarity and classification experiments.

`https://github.com/IBM/Project_CodeNet` is the link to the Project CodeNet repository, which contains software that supports and complements the CodeNet dataset. There are productivity tools to aggregate codes samples based on user criteria and pre-processing tools to transform code samples into sequence of tokens, simplified parse trees and code graphs. The repository also contains notebooks that illustrate the usage of some of the tools and source code and scripts we used to perform the experiments in the paper.

The URLs are all accessible to the general public.

## A.2 Author statement

IBM represents and warrants it is the original author of the dataset and has the right to re-publish associated third-party code under open source license terms. IBM further represents and warrants it has the authority to grant the rights and licenses (CDLA Permissive v2.0) associated with the dataset to third parties.

## A.3 Hosting and maintenance plan

The CodeNet dataset is hosted under the IBM Data Asset eXchange (DAX) platform, which is an online hub open to IBM and external developers and data scientists to find free and open data sets under open data licenses. For developers, DAX offers a trusted source for open data sets for artificial intelligence (AI). These data sets are ready to use in enterprise AI applications and are supplemented with relevant notebooks and tutorials. DAX was launched in 2019 and maintained by the Center for Open-Source Data & AI Technologies (CODAIT) team, who has been working on steadily adding new data sets to the exchange, as well as resources that help explore these data sets.

The Project CodeNet repository is hosted under `github.com/IBM` and is maintained by the Project CodeNet team in IBM Research.

## A.4 How to read the CodeNet dataset

The data and metadata are organized in a rigorous directory structure. The top level Project_CodeNet directory contains several sub-directories: `data`, `metadata`, `problem_descriptions`, and `derived`. The code samples or submissions reside under the data directory. The data directory is organized as `(problem_id)/(language)/(submission)`, so the file path `data/p00023/C++/s006384060.cpp` denotes a submission to problem p00023 in C++ with id s006384060. Detailed statement of the problems can be found in `problem_descriptions/(problem_id).html`. The meta data for the dataset is contained in the `metadata` directory. `metadata/problem_list.csv` contains metadata for all the problems in the dataset, which is summarized in Table 7. `metadata/(problem_id).csv` contains the metadata for all the submissions to problem `problem_id`, which is described in Table 8. Each submission comes with cpu time, memory usage and status with possible values described in Table 9. The `derived` directory contains information derived from the dataset, such as near-duplicate information for submissions to specific languages, token sequences for code samples, and information on identical problems.

## A.5 Long term preservation

The dataset is hosted in the IBM Data Asset eXchange and is stored on IBM Cloud. Project CodeNet is IBM's long term research effort to encourage open innovation at the intersection of AI and Software Engineering. IBM has demonstrated a sustained commitment to open source innovation and the CodeNet dataset and repository will be maintained and enhanced as long as is needed.

Table 7: Metadata at the dataset level

| name of column | data type | unit | description |
|---|---|---|---|
| id | string | none | unique anonymized id of the problem |
| name | string | none | short name of the problem |
| dataset | string | none | original dataset, AIZU or AtCoder |
| time_limit | int | millisecond | maximum time allowed for a submission |
| memory_limit | int | KB | maximum memory allowed for a submission |
| rating | int | none | rating, i.e., difficulty of the problem |
| tags | string | none | list of tags separated by "\|"; not used |
| complexity | string | none | degree of difficulty of the problem; not used |

Table 8: Metadata at the problem level

| name of column | data type | unit | description |
|---|---|---|---|
| submission_id | string | none | unique anonymized id of the submission |
| problem_id | string | none | anonymized id of the problem |
| user_id | string | none | anonymized user id of the submission |
| date | int | seconds | date and time of submission in the Unix timestamp format (seconds since the epoch) |
| language | string | none | mapped language of the submission (ex: C++14 ->C++) |
| original_language | string | none | original language specification |
| filename_ext | string | none | extension of the filename that indicates the programminglanguage used |
| status | string | none | acceptance status, or error type |
| cpu_time | int | millisecond | execution time |
| memory | int | KB | memory used |
| code_size | int | bytes | size of the submission source code in bytes |
| accuracy | string | none | number of tests passed; *Only for AIZU |

### A.6 License

The dataset is distributed under the CDLA Permissive v2.0 license (`https://github.com/Community-Data-License-Agreements/Working-Drafts/blob/main/CDLA-Permissive-2.0.md` and `https://www.linuxfoundation.org/category/press-release/linux-foundation-press-release/page/2/`) The repository is under the Apache License 2.0 (`https://www.apache.org/licenses/LICENSE-2.0`).

### A.7 Persistent dereferenceable identifier

The DOI for the Project CodeNet code repository is 10.5281/zenodo.4814770.

Table 9: All the possible status values

| status | abbreviation | numeric code |
|---|---|---|
| Compile Error | CE | 0 |
| Wrong Answer | WA | 1 |
| Time Limit Exceeded | TLE | 2 |
| Memory Limit Exceeded | MLE | 3 |
| Accepted | AC | 4 |
| Judge Not Available | JNA | 5 |
| Output Limit Exceeded | OLE | 6 |
| Runtime Error | RE | 7 |
| WA: Presentation Error | PE | 8 |
| Waiting for Judging | WJ | |
| Waiting for Re-judging | WR | |
| Internal Error | IE | |
| Judge System Error | | |

# B Datasheet

## B.1 Motivation

1. **For what purpose was the dataset created?**
   The CodeNet dataset provides a very large dataset of software source code written in a diversity of programming languages to drive algorithmic innovations in AI for code tasks like: code translation, code similarity, code classification, code search etc.
2. **Who created this dataset (e.g. which team, research group) and on behalf of which entity (e.g. company, institution, organization)?**
   The CodeNet dataset is created by a team of scientists at IBM Research and MIT-IBM Watson AI Lab comprising Ruchir Puri, David S. Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, Veronika Thost, Luca Buratti, Saurabh Pujar, Shyam Ramji, Ulrich Finkler, Susan Malaika, and Frederick Reiss.
3. **What support was needed to make this dataset?**
   Project CodeNet is a research project within the IBM Research Division, so it is funded by the IBM Corporation.

## B.2 Composition

1. **What are the instances?(that is, examples; e.g., documents, images, people, countries) Are there multiple types of instances? (e.g., movies, users, ratings; people, interactions between them; nodes, edges)**
   The dataset consists of computer programs that are submissions to online judging sites and their accompanying metadata. CodeNet does not have multiple types of instances.
2. **How many instances are there in total (of each type, if appropriate)?**
   The dataset comprises 13,916,868 submissions, divided into 4053 problems (of which 5 are empty). Of the submissions 53.6% (7,460,588) are accepted, 29.5% are marked as wrong answer and the remaining suffer from one of the possible rejection causes. The data contains submissions in 55 different languages, although 95% of them are coded in the six most common languages (C++, Python, Java, C, Ruby, C#). C++ is the most common language with 8,008,527 submissions (57% of the total) of which 4,353,049 are accepted.
3. **What data does each instance consist of ? "Raw" data (e.g., unprocessed text or images)? Features/attributes?**
   The data are files of software programs as is. The character encoding of each file is UTF-8.
4. **Is there a label or target associated with each instance? If so, please provide a description.**
   Yes, each instance of data (file) has associated metadata that may be interpreted as labels. The problem that a certain instance (code sample) intends to solve may be used as a label in classification and similarity. The acceptance status of each code sample, the CPU time, and memory footprint can also be used as labels.

5. **Is any information missing from individual instances? If so, please provide a description, explaining why this information is missing (e.g., because it was unavailable). This does not include intentionally removed information, but might include, e.g., redacted text.**

   Some metadata values might not be available for all instances. This can be attributed to the source not (or incorrectly) providing the metadata.

6. **Are relationships between individual instances made explicit (e.g., users' movie ratings, social network links)? If so, please describe how these relationships are made explicit.**

   Relationships between instances are explicitly available in the provided metadata. As an example, multiple instances (code submissions) by the same person can be found by scanning the metadata for that person's (anonymized) id number. All submissions to a particular problem id are to be found in a single metadata CSV file.

7. **Are there recommended data splits (e.g., training, development/validation, testing)? If so, please provide a description of these splits, explaining the rationale behind them.**

   Data splits are left to the discretion of the user, since CodeNet can be used for a wide variety of use cases. No such division is made in the dataset.

8. **Are there any errors, sources of noise, or redundancies in the dataset? If so, please provide a description.**

   It all depends on how errors, noise and redundancies are defined. There are probably minor errors in the metadata directly attributable to the source: some non-Accepted programs are identified with the wrong language, probably caused by a programmer making a wrong selection while submitting his or her work. Some run-time data for incorrect programs are listed as a negative number. It happens that some user or users submit the same program (data instance) multiple times to the same or different problem tasks.

9. **Is the dataset self-contained, or does it link to or otherwise rely on external resources (e.g., websites, tweets, other datasets)? If it links to or relies on external resources, a) are there guarantees that they will exist, and remain constant, over time; b) are there official archival versions of the complete dataset (i.e., including the external resources as they existed at the time the dataset was created); c) are there any restrictions (e.g., licenses, fees) associated with any of the external resources that might apply to a future user? Please provide descriptions of all external resources and any restrictions associated with them, as well as links or other access points, as appropriate.**

   The dataset is self-contained.

10. **Does the dataset contain data that might be considered confidential (e.g. data that is protected by legal privilege or by doctor-patient confidentiality, data that includes the content of individuals' non-public communications)? If so, please provide a description.**

    No. Any personal information that is available in the metadata at the source websites is anonymized in the dataset. However, it is possible that names or handles of persons still being present in the source code instances themselves as variable, class or function names.

11. **Does the dataset contain data that, if viewed directly, might be offensive, insulting, threatening, or might otherwise cause anxiety? If so, please describe why.**

    We have done some filtering. In one case, the programming language name is offensive, so we renamed it. It might be possible that people used offensive language in naming a variable in the program, but we have made every possible effort to minimize any such possibility.

12. **Does the dataset relate to people?**

    No.

### B.3 Collection Process

1. **How was the data associated with each instance acquired? Was the data directly observable (e.g., raw text, movie ratings), reported by subjects (e.g., survey responses), or indirectly inferred/derived from other data (e.g., part-of-speech tags, model-based guesses for age or language)? If data was reported by subjects or indirectly inferred/derived from other data, was the data validated/verified? If so, please describe how.**

   The data are acquired from publicly accessible on-line judging websites. We used the AIZU (https://judge.u-aizu.ac.jp/) and AtCoder (https://atcoder.jp/) online judging sites. The data are accessible and observable by clicking specific url links.

2. **What mechanisms or procedures were used to collect the data (e.g., hardware apparatus or sensor, manual human curation, software program, software API)? How were these mechanisms or procedures validated?**

   Some of the data was available as archived zip files or a REST API for download, otherwise a webpage scraper tool was used to retrieve the data (while observing any throttles on bandwidth). No verification beyond mere manual inspection was applied to the downloaded data.

3. **If the dataset is a sample from a larger set, what was the sampling strategy (e.g., deterministic, probabilistic with specific sampling probabilities)?**

   No specific strategy: as much data as was available at the time (2020) was downloaded.

4. **Who was involved in the data collection process (e.g., students, crowdworkers, contractors) and how were they compensated (e.g., how much were crowdworkers paid)?**

   There were no third-party participants in the data collection.

5. **Over what timeframe was the data collected? Does this timeframe match the creation timeframe of the data associated with the instances (e.g., recent crawl of old news articles)? If not, please describe the timeframe in which the data associated with the instances was created.**

   The data was collected in 2020 and the code samples might go back to a decade ago. The dataset was first published on May 5, 2021.

6. **Were any ethical review processes conducted (e.g. by an institutional review board)? If so, please provide a description of these review processes, including the outcomes, as well as a link or other access point to any supporting documentation.**

   No. The dataset was examined by IBM Corporation lawyers for suitability of public disclosure.

7. **Does the dataset relate to people? If not, you may skip the remainder of the questions in this section.**

   Only as far as the fact that the data instances are created/written by people.

8. **Did you collect the data from the individuals in question directly, or obtain it via third parties or other sources (e.g. websites)?**

   The data was collected indirectly from submitters via online judging websites.

9. **Did the individuals in question consent to the collection and use of their data? If so, please describe (or show with screenshots or other information) how consent was requested and provided, and provide a link or other access point to, or otherwise reproduce, the exact language to which the individuals consented.**

   They did consent directly to the respective online judging sites that we used as source. See e.g. `https://onlinejudge.u-aizu.ac.jp/term_of_use`.

### B.4 Data Preprocessing/Cleaning

1. **Was any preprocessing/cleaning/labeling of the data done (e.g., discretization or bucketing, tokenization, part-of-speech tagging, SIFT feature extraction, removal of instances, processing of missing values)? If so, please provide a description. If not, you may skip the remainder of the questions in this section.**

   Minor processing of the data instances was performed mainly to make all file name extensions uniform and make sure the character encoding is UTF-8, all line endings adhere to the UNIX standard (a single linefeed character), and any byte-order marks (BOM) are removed.

   All metadata was carefully examined, anonymized where necessary and corrected or updated when possible (e.g. the file size in bytes is part of the metadata and needed updating).

2. **Was the "raw" data saved in addition to the preprocessed/cleaned/labeled data (e.g., to support unanticipated future uses)? If so, please provide a link or other access point to the "raw" data.**

   The raw data is saved but considered not to be part of the published Project CodeNet dataset.

3. **Is the software used to preprocess/clean/label the instances available? If so, please provide a link or other access point.**

   Some of the software (mostly bash scripts) are available in our github `https://github.com/IBM/Project_CodeNet`.

4. **Does this dataset collection/processing procedure achieve the motivation for creating the dataset stated in the first section of this datasheet? If not, what are the limitations?**
   Yes. This dataset and its derived benchmark datasets offer the scale, diversity, and quality to drive research in applying AI techniques to code.

### B.5 Uses

1. **Has the dataset been used for any tasks already? If so, please provide a description.**
   Yes. Several baseline experiments on code classification and similarity have been performed and documented in the paper.
2. **Is there a repository that links to any or all papers or systems that use the dataset? If so, please provide a link or other access point.**
   Yes. `https://github.com/IBM/Project_CodeNet`.
3. **What (other) tasks could the dataset be used for?**
   The rich metadata and diversity open Project CodeNet to a plethora of uses cases. The problem-submission relationship in Project CodeNet corresponds to type-4 similarity and can be used for code search and clone detection. The code samples in Project CodeNet are labeled with their acceptance status and we can explore AI techniques to distinguish correct codes from problematic ones. Project CodeNet's metadata also enables the tracking of how a submission evolves from problematic to accepted, which could be used for exploring automatic code correction. A large number of code samples come with inputs so that we can execute the codes to extract the CPU run time and memory footprint, which can be used for regression studies and predictions. Given its wealth of programs written in a multitude of languages, Project CodeNet may serve as a valuable benchmark dataset for source-to-source translation.
4. **Is there anything about the composition of the dataset or the way it was collected and pre-processed/cleaned/labeled that might impact future uses? For example, is there anything that a future user might need to know to avoid uses that could result in unfair treatment of individuals or groups (e.g. stereotyping, quality of service issues) or other undesirable harms (e.g. financial harms, legal risks) If so, please provide a description. Is there anything a future user could do to mitigate these undesirable harms?**
   No.
5. **Are there tasks for which the dataset should not be used? If so, please provide a description.**
   No.

### B.6 Dataset Distribution

1. **Will the dataset be distributed to third parties outside of the entity (e.g. company, institution, organization) on behalf of which the dataset was created? If so, please provide a description.**
   Yes, the dataset will be distributed to the general public.
2. **When will the dataset be released/first distributed? What license (if any) is it distributed under?** The dataset was released in May 2021 under the the CDLA Permissive v2.0 licence `https://github.com/Community-Data-License-Agreements/Working-Drafts/blob/main/CDLA-Permissive-2.0.md`.
3. **How will the dataset be distributed (e.g. tarball on website, API, GitHub)? Does the dataset have a digital object identifier (DOI)?**
   The dataset is made available as a downloadable gzipped tar file here: `https://developer.ibm.com/technologies/artificial-intelligence/data/project-codenet/`. There is no DOI yet.
4. **Will the dataset be distributed under a copyright or other intellectual property (IP) license, and/or under applicable terms of use (ToU)? If so, please describe this license and/or ToU, and provide a link or other access point to, or otherwise reproduce, any relevant licensing terms or ToU, as well as any fees associated with these restrictions.**
   The dataset is made available under the CDLA Permissive v2.0 licence `https://github.com/Community-Data-License-Agreements/Working-Drafts/blob/main/CDLA-Permissive-2.0.md`.

5. **Have any third parties imposed IP-based or other restrictions on the data associated with the instances? If so, please describe these restrictions, and provide a link or other access point to, or otherwise reproduce, any relevant licensing terms, as well as any fees associated with these restrictions.**
   No, not as far as we know.
6. **Do any export controls or other regulatory restrictions apply to the dataset or to individual instances? If so, please describe these restrictions, and provide a link or other access point to, or otherwise reproduce, any supporting documentation.**
   No. These code samples are solutions to pedagogical programming problems at the high school and beginning college level and should not be subject to export control.

### B.7 Dataset Maintenance

1. **Who is supporting/hosting/maintaining the dataset?**
   International Business Machines corporation.
2. **How can the owner/curator/manager of the dataset be contacted (e.g. email address)?**
   The users can create an issue on our github or contact any of the listed authors.
3. **Is there an erratum? If so, please provide a link or other access point.**
   No.
4. **Will the dataset be updated (e.g. to correct labeling errors, add new instances, delete instances)? If so, please describe how often, by whom, and how updates will be communicated to users (e.g. mailing list, GitHub)?**
   Yes, there are plans to add new instances to the dataset, in the next six months to a year's time frame. The update will be performed by IBM and communicated through the github.
5. **If others want to extend/augment/build on this dataset, is there a mechanism for them to do so? If so, please provide a description. Will these contributions be validated/verified? If so, please describe how. If not, why not? Is there a process for communicating/distributing these contributions to other users? If so, please provide a description.**
   There is no such mechanism yet, but it is under consideration. Interested parties are invited to consider contacting the authors or creating an issue to that effect in our github.

## C Further information of CodeNet

Table 10 summarizes the metadata available for each code submission to a problem. Figure 4 gives the distributions of problems based on number of submissions received.

Table 10: Submission metadata.

| column | unit/example | description |
|---|---|---|
| submission_id | s[0-9]{9} | anonymized id of submission |
| problem_id | p[0-9]{5} | anonymized id of problem |
| user_id | u[0-9]{9} | anonymized user id |
| date | seconds | date and time of submission |
| language | C++ | consolidated programming language |
| original_language | C++14 | original language |
| filename_ext | .cpp | filename extension |
| status | Accepted | acceptance status, or error type |
| cpu_time | millisecond | execution time |
| memory | kilobytes | memory used |
| code_size | bytes | source file size |
| accuracy | 4/4 | passed tests (AIZU only) |

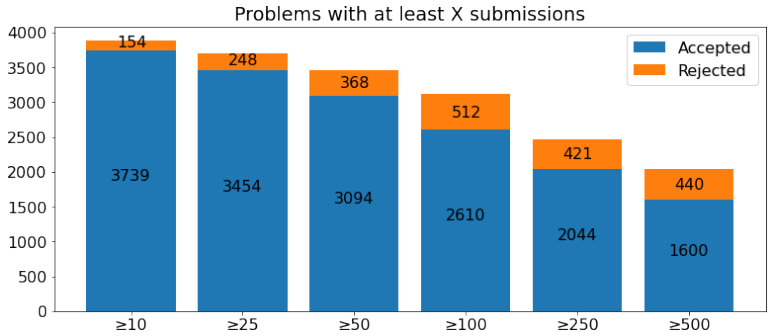



Figure 4: Number of problems providing at least X submissions. The bars show both the numbers of accepted submissions (blue) and rejected submissions (orange).

## D Details of Experiments on Code Classification

### D.1 MLP with Bag of Tokens

One of the simplest representations of a code sample is a bag of tokens. Here, the code sample is represented by a vector of relative frequencies of token occurrences in the source code. The vector is computed by the following steps:

1. Convert a given source code into a sequence of tokens using a tokenizer (i.e., lexical analyzer).
2. From this sequence, remove the tokens considered not useful for code classification.
3. Count the number of each token type in the reduced sequence and form a vector of counts.
4. Normalize the vector with respect to L2 norm.

We do not use all tokens available in the grammar of the programming language. Only some operators and keywords are used. All identifiers, comments and literals are ignored. We also ignore some operators and many keywords that in our opinion provide no significant information on the algorithm the source code implements.

The vector representing a bag of tokens has the same length for every code sample, which makes it convenient for processing with a neural network. The vector is usually short, which makes training of a neural network fast. However, in a bag-of-tokens representation, information about the number of occurrences and position of each token is lost. Hence, the accuracy of a classifier using a bag-of-tokens representation is rather limited.

Table 11 provides results of code classification of all four benchmarks. The columns give the benchmark name, the test accuracy, the number of training epochs, the run time of each epoch, and the number of token types considered. All networks are implemented using Keras API of TensorFlow machine learning tool. Training is performed on a single V100 GPU, using Adam optimizer with learning rate 1e-3, and batches of 32 samples. In each experiment, 20% of the samples are used for testing, while the rest are split in 4:1 for training and validation, respectively.

Table 11: Code classification by MLP with bag of tokens.

| Benchmark dataset | Accuracy %% | Number epochs | Run time sec/epoch | Number tokens |
|---|---|---|---|---|
| Java250 | 71.00±0.29 | 30 | 2 | 81 |
| Python800 | 67.80±0.15 | 22 | 7 | 71 |
| C++1000 | 68.26±0.21 | 20 | 14 | 56 |
| C++1400 | 64.50±0.13 | 17 | 12 | 56 |

Figure 5 shows the neural network used for solving the classification problem for the C++1400 benchmark. The neural networks used for classification of other benchmarks are similar to this one. As we see in Table 11 their performance is quite similar.

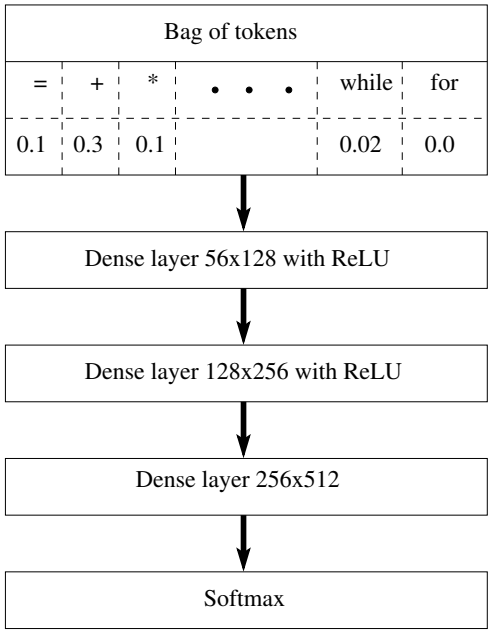



Figure 5: MLP architecture for code classification.

From Table 11 we see that training is rather fast, the reason being that the network is simple. In spite of simplicity, this neural network performs very well. The 64.50±0.13% test accuracy for C++1400 benchmark dataset is significantly better than the potential 0.071% accuracy of random guess. It indicates that the relative frequencies of source code tokens provide sufficient information for classifying code.

### D.2 CNN with Token Sequence

The sequence-of-tokens representation retains more information of a code sample than the bag-of-tokens representation. For our experiments on code classification, we use the same set of tokens that is used in the above bag-of-tokens approach. Similarly, we omit all comments and identifiers.

Table 12 shows results of code classification on all four benchmarks by using the sequence-of-tokens representation. The columns give the benchmark name, the test accuracy, the number of training epochs, the run time of each epoch, and the number of token types considered. All networks are implemented using Keras API of TensorFlow machine learning tool. The training is performed on

Table 12: Code classification by CNN with token sequence.

| Benchmark dataset | Accuracy %% | Number epochs | Run time sec/epoch | Number tokens |
|---|---|---|---|---|
| Java250 | 89.52±0.59 | 810 | 10 | 81 |
| Python800 | 87.46±0.25 | 504 | 26 | 71 |
| C++1000 | 93.96±0.18 | 235 | 59 | 56 |
| C++1400 | 93.71±0.18 | 334 | 60 | 56 |

four V100 GPUs, using Adam optimizer in data parallel mode with learning rate 1e-3, and batches of 512 samples. In each experiment, 20% of the samples are used for testing, while the rest are split in 4:1 for training and validation, respectively.

We have experimented with several types of neural networks. Figure 6 shows the neural network we choose for the C++1400 benchmark. It is a multi-layer convolutional neural network. It uses categorical encoding of source code tokens. For batching, the sequences of tokens are padded with zeros.

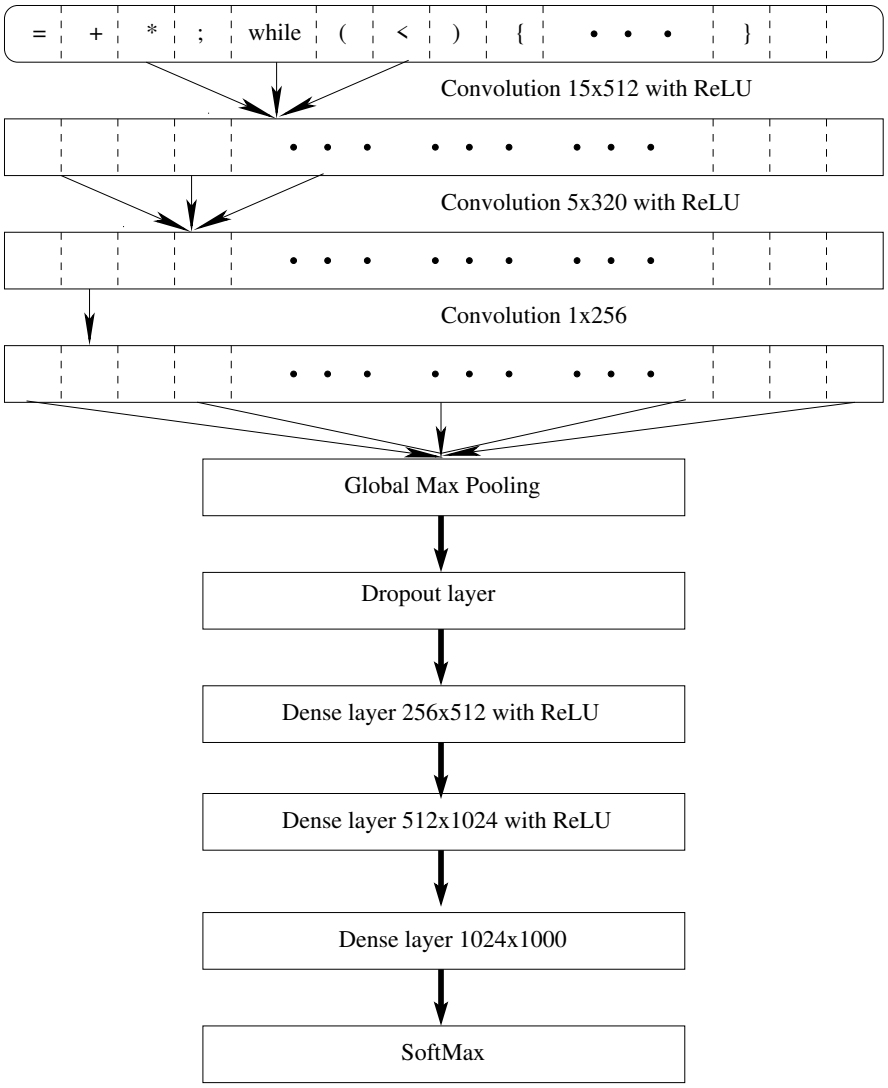



Figure 6: CNN architecture for code classification.

Using this network we get a test accuracy 93.71±0.18% for C++1400 benchmark dataset, which is significantly better than the accuracy shown by the bag-of-tokens approach. The neural networks

used for classification of other benchmarks are similar to the one shown in Figure 6. As we see in Table 12, their performance is similar.

### D.3 C-BERT with Token Sequence

The sequence-of-tokens representation can be used with other neural networks of increasing capacity. We build a C-BERT model (a transformer model introduced in [33]) by pre-training on 10,000 top starred GitHub open source projects written in C, where we use Clang C tokenizer and Sentencepiece to tokenize the pre-training data. The C-BERT model is then fine tuned on each classification benchmark. Additionally, we experiment with the POJ-104 dataset, which contains code examples in C and C++.

C-BERT achieves appealing results on binary classification and vulnerability detection with C source code [10, 47]. However, it has not been used on multiclass classification tasks or with other languages such as C++, Java, and Python. Because we use sub-word tokenization and different programming languages share common tokens, we could apply the C-BERT model directly on the benchmarks.

After pretraining, we fine tune the model for five epochs on each benchmark, with a batch size 32 and learning rate 2e-5. The fine-tuning was done on two V100 GPUs and it took 30 minutes to four hours, depending on the size of the dataset. The sub-word vocabulary size is 5,000. Contexts larger than 512 tokens were truncated.

Table 13 summarizes the accuracies C-BERT achives on the four CodeNet benchmarks as well as the POJ-104 dataset. C-BERT achieves high accuracy and performs the best on Java and Python.

Table 13: C-BERT results (accuracy, in %) for code classification.

| | POJ-104 | C++1000 | C++1400 | Java250 | Python800 |
|---|---|---|---|---|---|
| C-BERT | 98.41±0.01 | 93.79±0.01 | 91.83±0.06 | 97.40±0.19 | 97.09±0.18 |

The relatively low performance on C++ benchmarks is possibly related to the idiosyncrasies of the dataset and certain programming practices. Manual inspection suggests that lack of detailed variable names in C++ hurts the performance of the model, in problems appearing similar and having similar solutions. Removing one of the similar problems improves the model performance on the other problem. Moreover, one programming practice which could potentially confuse the models is that certain C++ users copied common constants (e.g., pi and epsilon) and data structures (e.g., enums) to all solutions they submitted. In many cases, these duplicate contents were not even used. We did not observe such practices in Python and Java.

### D.4 GNN with SPT

We experiment with four types of GNNs with SPT-based graph representations of the source code: the Graph Convolutional Network (GCN) [34], the Graph Isomorphism Network (GIN) [35], and a virtual-node-included variant for each (denoted by -V). The variant adds a virtual node to the graph to enhance graph message passing [36]. We use the Adam optimizer with learning rate 1e-3 for training. All GNN models have five layers. We have experimented with more than 5 layers (i.e., 8 and 10), however deeper GNNs do not improve performance, as deeper GNNs might suffer from the over-smoothing problem (i.e., node features become less distinguishable after many rounds of message passing) [48].

We conduct 6/2/2 random split for each of the 4 benchmarks: i.e., 60% training data, 20% testing data, and 20% validation data. We run five folds for each benchmark with early stop "patience" set 20 (i.e., stop only when validation loss has not decreased in the past 20 epochs). Our model training typically converges within 200 epochs in a 1-fold run. We modified OGB [49] code-base with PyTorch Geometric [50] back-end over PyTorch 1.6.0 [51] to run our experiments. The experiments are conducted on one NVIDIA V100 GPU. For large benchmarks such as C++1000 and C++1400, it takes about 1 week to finish a 5-fold run. We summarize model accuracy, training time over 5-folds, and training epochs over 5-folds in Table 14. As we can see, adding a virtual node improves GNN performance (both GCN and GIN). Overall, GIN and its variants work better than GCN and its variants, likely due to the fact that GIN theoretically generalizes the Weisfeiler-Lehman Isomorphism Test and achieves maximum expressive power among GNNs [52].

For the detailed model, hyper-parameter setup, data splits and etc, please refer to `https://github.com/IBM/Project_CodeNet/tree/main/model-experiments/gnn-based-experiments`.

Table 14: GNN (SPT) results for code classification. Each task trains over 5-folds with early stopping patience parameter set as 20. We record test accuracy (with standard deviation), total training time over 5 folds, and total training epochs over 5 folds.

| | Java250 | Python800 | C++1000 | C++1400 |
|---|---|---|---|---|
| GCN | 92.70±0.25<br>10.55 hrs<br>411 epochs | 93.82±0.16<br>14.50 hrs<br>219 epochs | 95.76±0.12<br>47.96 hrs<br>228 epochs | 95.26±0.13<br>67.34 hrs<br>310 epochs |
| GCN-V | 93.02±0.81<br>12.50 hrs<br>419 epochs | 94.30 ±0.15<br>23.02 hrs<br>325 epochs | 96.09±0.17<br>61.55 hrs<br>287 epochs | 95.73±0.07<br>71.85 hrs<br>358 epochs |
| GIN | 93.26±0.23<br>19.80 hrs<br>513 epochs | 94.17±0.19<br>41.67 hrs<br>496 epochs | 96.34±0.15<br>116.67 hrs<br>441 epochs | 95.95±0.13<br>133.50 hrs<br>502 epochs |
| GIN-V | 92.77±0.66<br>26.25 hrs<br>656 epochs | 94.54±0.12<br>51.67 hrs<br>570 epochs | 96.64±0.10<br>142.25 hrs<br>496 epochs | 96.36±0.10<br>208.47 hrs<br>678 epochs |

# E Details of Experiments on Code Similarity

## E.1 MLP with Bag of Tokens

For experiments on code similarity analysis, we use the same bag of tokens as for code classification. The input to the neural network is constructed by concatenating two bags of tokens, one for each source code file.

Table 15 provides results of code similarity analysis on all four benchmarks. The columns give the benchmark name, the test accuracy, the number of training epochs, the number of samples in each epoch, the run time of each epoch, the number of token types considered, and the number of test samples. All networks are implemented using Keras API of TensorFlow machine learning tool. The training is performed on a single V100 GPU, using Adam optimizer with learning rate 1e-3, and batches of 256 samples.

Table 15: Similarity analysis by MLP with bag of tokens.

| Benchmark dataset | Accuracy %% | Number epochs | Size of epoch | Run time sec/epoch | Number tokens | N test samples |
|---|---|---|---|---|---|---|
| Java250 | 81.80±0.06 | 20 | 4,096,000 | 21 | 81 | 512,000 |
| Python800 | 86.61±0.08 | 94 | 4,096,000 | 24 | 71 | 512,000 |
| C++1000 | 85.82±0.05 | 64 | 4,096,000 | 21 | 56 | 512,000 |
| C++1400 | 86.54±0.07 | 64 | 4,096,000 | 22 | 56 | 512,000 |

Figure 7 shows the neural network used for code similarity analysis on the C++1400 benchmark. The neural networks used for code similarity analysis on other benchmarks are similar to this one. As we see in Table 15, their accuracy is similar.

As we see in Table 15, the model accuracy is rather modest (<87%) for all benchmark datasets, which is not very high for a binary classification problem of a fully balanced dataset. Obviously, the bag of tokens is too primitive and misses many important details necessary for identifying similarity.

## E.2 Siamese Network with Token Sequence

For experiments on code similarity, we use the same sequence of tokens as for code classification. The neural network has two inputs, one for each source code file. After experimenting with various neural network architectures, we select the siamese network for its good performance.

Table 16 provides results of code similarity analysis on all four benchmarks. The columns give the benchmark name, the test accuracy, the number of training epochs, the number of samples in each

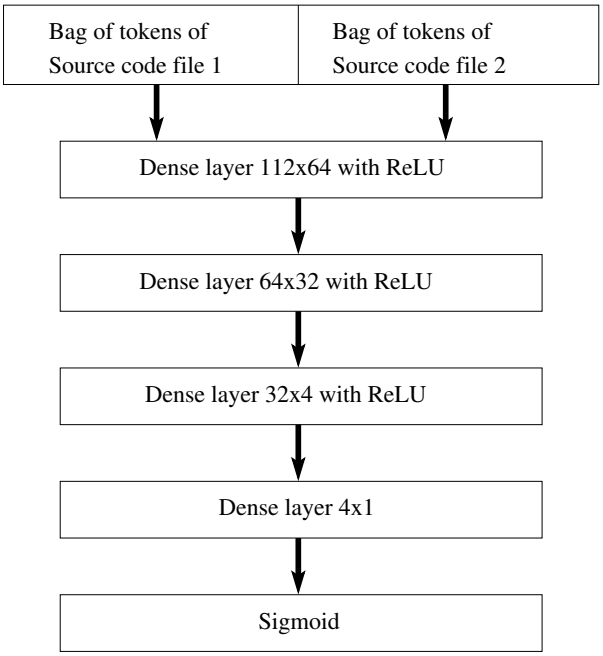



Figure 7: MLP architecture for similarity analysis.

epoch, the run time of each epoch, the number of token types considered, and the number of test samples. All networks are implemented using Keras API of TensorFlow machine learning tool. The training is performed on four V100 GPUs, using Adam optimizer in data parallel mode with learning rate 1e-3, and batches of 512 samples.

Table 16: Similarity analysis by Siamese network with token sequence.

| Benchmark dataset | Accuracy %% | Number epochs | Size of epoch | Run time sec/epoch | Number tokens | N test samples |
|---|---|---|---|---|---|---|
| Java250 | 89.70±0.18 | 29 | 51,200 | 114 | 75 | 512,000 |
| Python800 | 94.67±0.12 | 110 | 64,000 | 89 | 71 | 512,000 |
| C++1000 | 96.19±0.08 | 123 | 64,000 | 89 | 56 | 512,000 |
| C++1400 | 96.56±0.07 | 144 | 64,000 | 96 | 56 | 512,000 |

The neural network for the C++1400 benchmark is depicted in Figure 8. The siamese parts of the network have the same structure and share all their weights. If the inputs are identical, so are the outputs. Therefore, by construction, the network guarantees detecting similarity of identical source code samples. The outputs of the siamese parts are compared by computing the absolute difference.

The network shows 96.56±0.07% test accuracy for C++1400 benchmark dataset. We consider this a good result, especially considering that the token sequence ignores all identifiers, comments, and many keywords. The neural networks used for code similarity analysis of other benchmarks are similar to the one shown in Figure 8. As we see in Table 16, their accuracy is quite similar.

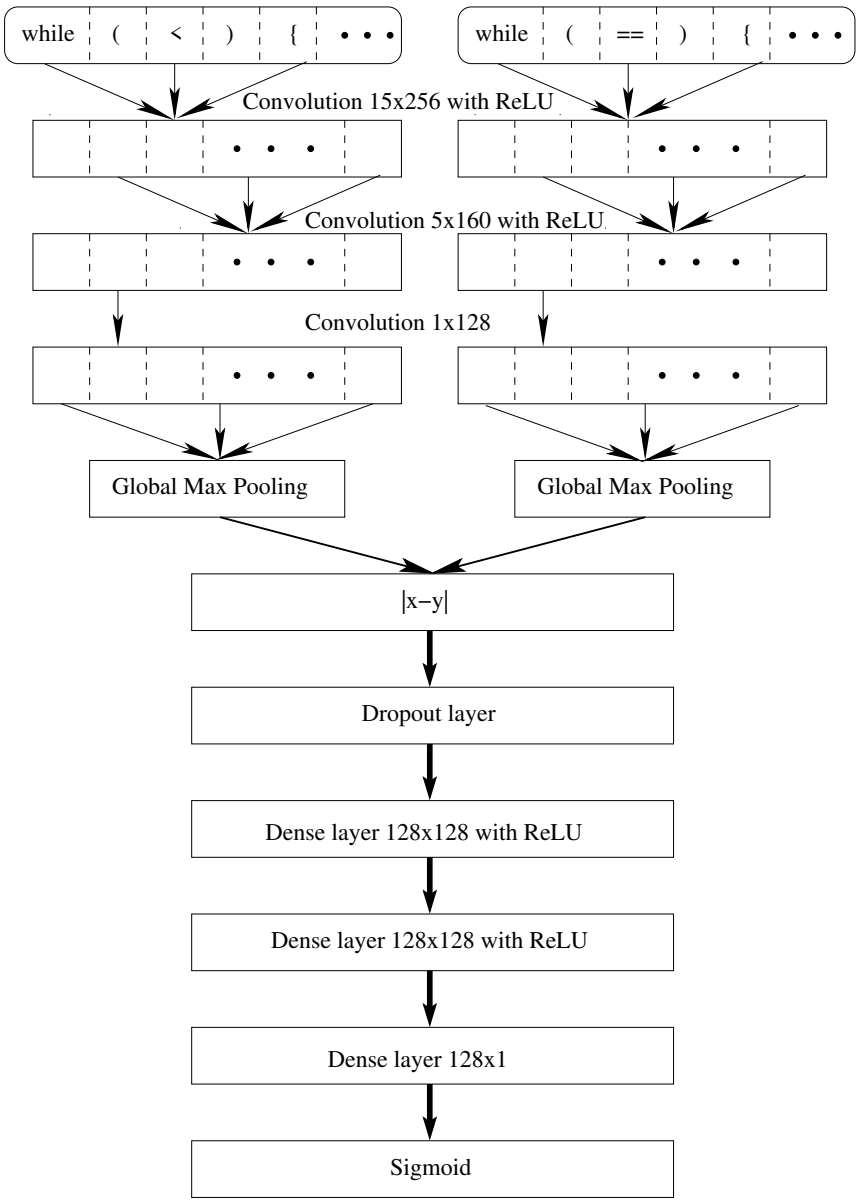



Figure 8: Siamese architecture for similarity analysis.

### E.3 SPT-based experiments

Following MISIM [21], the train, validation, and test datasets for the SPT-based experiments draw from entirely different problems. In our experiments, we use 50% problems for training, 25% for validation, and 25% for test. The train, validation, and test split used for the experiments can be found at [53]. Similarity scores in Table 5 and Table 6 report mean and standard deviation of MAP@R [40] values evaluated with models trained using five random seeds. The models are trained on a Xeon(R) CPU E5-2680 v4, 2.4GHz, 256 GiB memory using a NVIDIA V100 GPU. The SPTs used in these experiments have nodes annotated with attributes derived by combining SPT features (refer to Section 6), following the context-aware semantic structure (CASS) proposed in [21].

AROMA experiments are performed using the implementation in MISIM's supplementary material [23] and the input (SPTs) used for these experiments can be found at [53]. Due to the high memory requirement for computing MAP@R on the test set of CodeNet benchmarks, we had to reduce the feature set of AROMA. We estimate that AROMA results can improve by 10–25% when

all features are used. AROMA is rule-based and no training is involved, hence we don't report mean and standard deviation in Table 5. For each of the four datasets – Java250, Python800, C++1000, C++1400 – MISIM's GNN model is trained for a total of 1000 epochs at a learning rate of 0.001 with Adam optimizer. Each epoch consists of 1000 iterations, and in each iteration, 16 problems and 5 solutions per problem are randomly sampled, and all solution pairs are used for training as in [21]. MISIM results for the four languages can be reproduced by downloading the MISIM code and scripts [23] and using the provided CASS files [53] as input.

For the GMN experiments (row 2 and row 3 in Table 6), we adapt the implementation in [39] of the GMN model [38] using SPTs [53] as graphs. We follow the recommendations in [38] for the model configuration, as they produce the best and stable results in our experiments. Specifically, we use 5 layers of propagation with weight sharing across layers, dot-product similarity for the cross-graph attention mechanism, and GRU layer to update node embeddings from the propagation scheme. For GMN training, given the large set of SPT pairs, we adopt an approach similar to [21] of randomly sampling 16 problems with 5 solutions each. We use triplet loss with approximate hamming similarity [38] for each sample, which is formed using a similar pair combined with a dissimilar SPT. After every 100 iterations with a batch size of 64, another set of 16 problems and 5 solutions are sampled randomly for a total of 150,000 iterations (1500 sampled sets). GMN results could improve further with more training iterations. We use Adam optimizer with a learning rate of 1e-4 for training.

The first two rows of Table 6 compare similarity models trained on SPT graph structure only. The first row in the table adapts the MISIM GNN model by masking the node labels to allow the model to learn structural features only. The second row uses the GMN [38] model with cross-graph attention-based matching for structural similarity using a node vector dimension of 32 and graph representation dimension of 128.

For the GMN+MISIM node attributes experiment, row 3 in Table 6, we allow the GMN model to learn features based on both node attributes and the SPT structure. Accordingly, we replace the node encoder in the GMN, an MLP, with an embedding layer, for generating node feature vectors. We explore different node feature vector dimensions, such as 64, 100, 128, and found 100 to produce good results for the given number of training iterations. All other parameter settings remain the same as the structure only GMN experiments from row 2 of Table 6. The GMN results can be reproduced using the Java250 CASS files available at [53].

MAP@R score [40] is computationally expensive for GMN models because an embedding has to be computed for all SPT pairs in the test set, and hence Table 6 reports results on smaller sampled test sets.

## F Details of MLM Experiment

Here we show how a masked language model (MLM) can be trained with CodeNet. We closely follow the approach by Ankur Singh, documented in the blog [54]. The goal of the model is to infer the correct token for an arbitrary masked-out location in the source text. We assume that in every text, precisely one token is randomly masked. The original token at such position is then the golden label.

From each of the 1000 C++1000 problems, we randomly select 100 samples for training and another 100 for testing. Each C++ source file is tokenized into a vocabulary of 442 distinct tokens as categorized in Table 17. For example, `while` is a keyword and `strlen` is a function literal.

Table 17: Token categories used for MLM.

| Type | Count | Description |
|---|---|---|
| the keyword | 95 | all C++20 reserved words |
| the function | 280 | function names in common header files |
| the identifier | 42 | standard identifiers, like stderr, etc. |
| the punctuator | 16 | small set of punctuation symbols |
| # or ## | 2 | the C pre-processor symbols |
| 0, 1 | 2 | special case for these frequent constants |
| the token class | 5 | identifier, number, operator, character, string |

This code snippet:

```
for (i = 0; i < strlen(s); i++) {}
```

will be tokenized to:

```
for ( id = 0 ; id < strlen ( id ) ; id operator ) { }
```

The tokenized source files are read into a pandas dataframe and processed by the Keras Text Vectorization layer, to extract a vocabulary and encode all token lines into vocabulary indices, including the special "[mask]" token. Each sample has a fixed token length of 256. The average number of tokens per sample across the training set is 474. Short samples are padded with 0 and those that are too large are simply truncated.

The model is trained with 100,000 samples in batches of 32 over five epochs, with a learning rate of 0.001 using the Adam optimizer. We evaluate the trained model on a test set of 100,000 samples. Each sample is pre-processed in the same way as the training samples and one token (never a padding) is arbitrarily replaced by the "[mask]" symbol. Then, a prediction is generated and the top 1 and top 5 results are compared with the expected value. The achieved accuracies are top-1: 0.9104 (stddev: 0.002) and top-5: 0.9935 (stddev: 0.0005).