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)

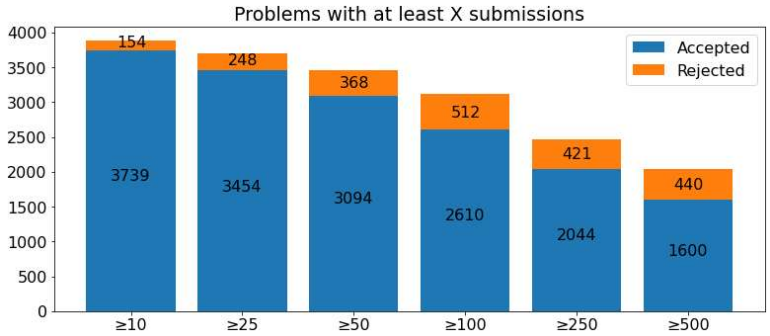

Figure 4: Number of problems providing at least X submissions. The bars show both the numbers of accepted submissions (blue) and rejected submissions (orange).

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.

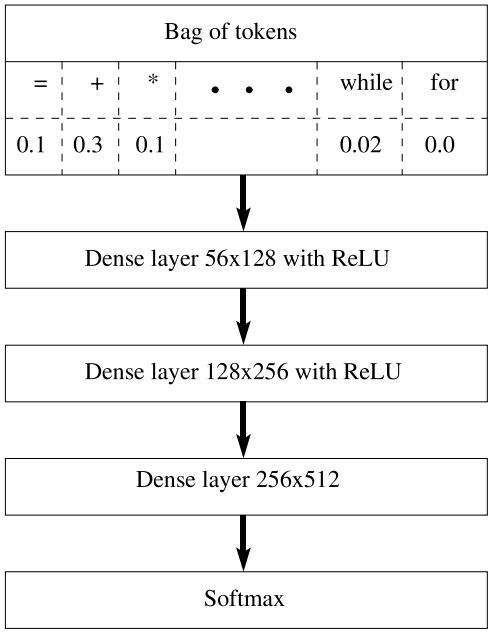

Figure 5: MLP architecture for code classification.

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.

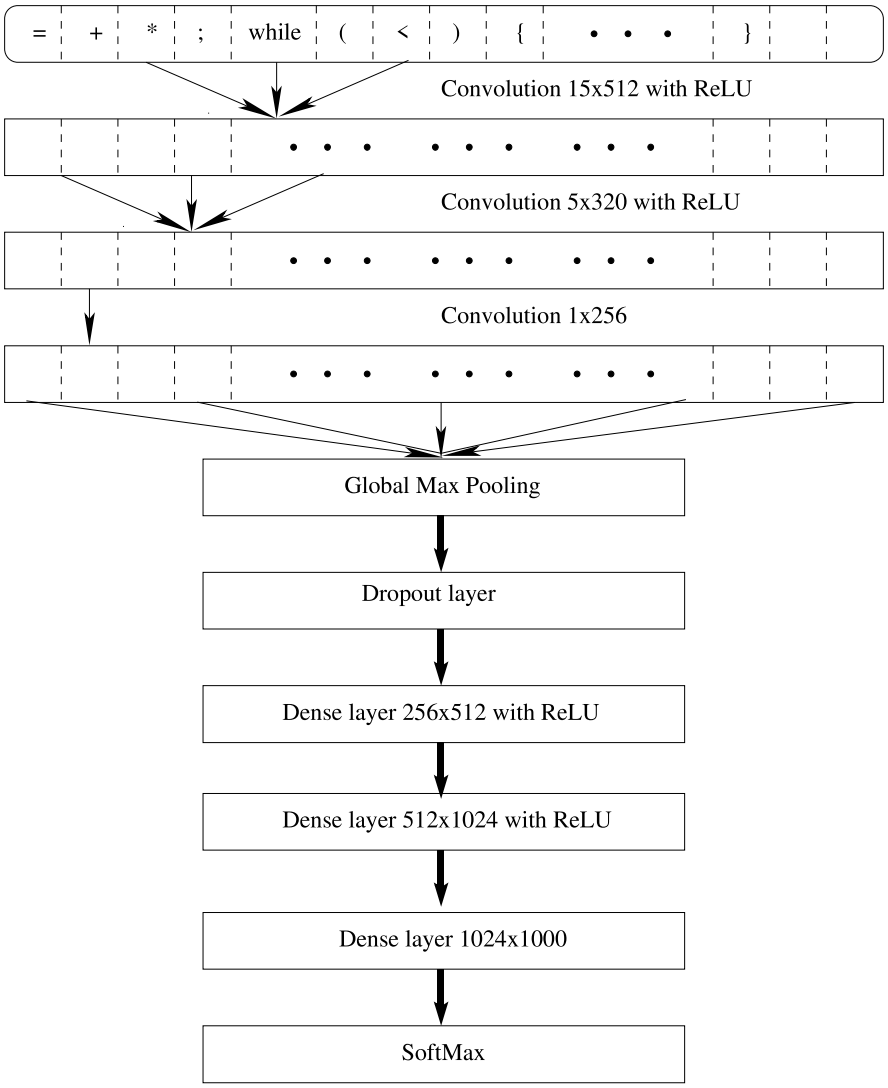

Figure 6: CNN architecture for code classification.

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, 49]. 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) [50].

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 [51] code-base with PyTorch Geometric [52] back-end over PyTorch 1.6.0 [53] 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 [54].

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

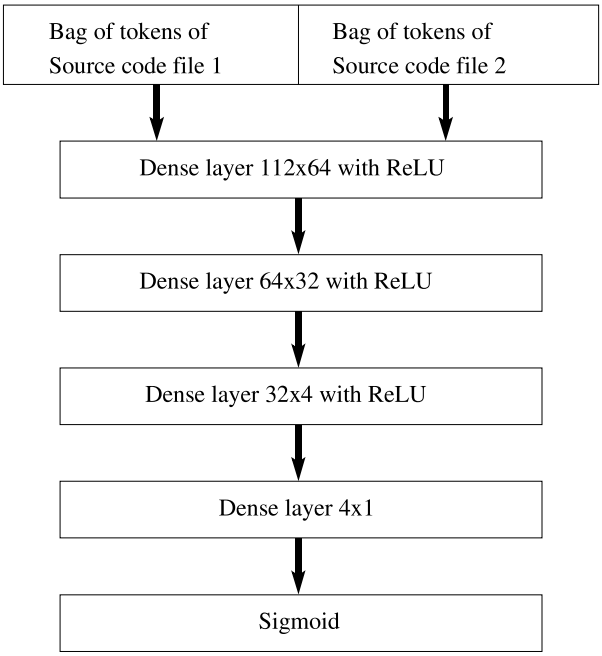

Figure 7: MLP architecture for similarity analysis.

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.

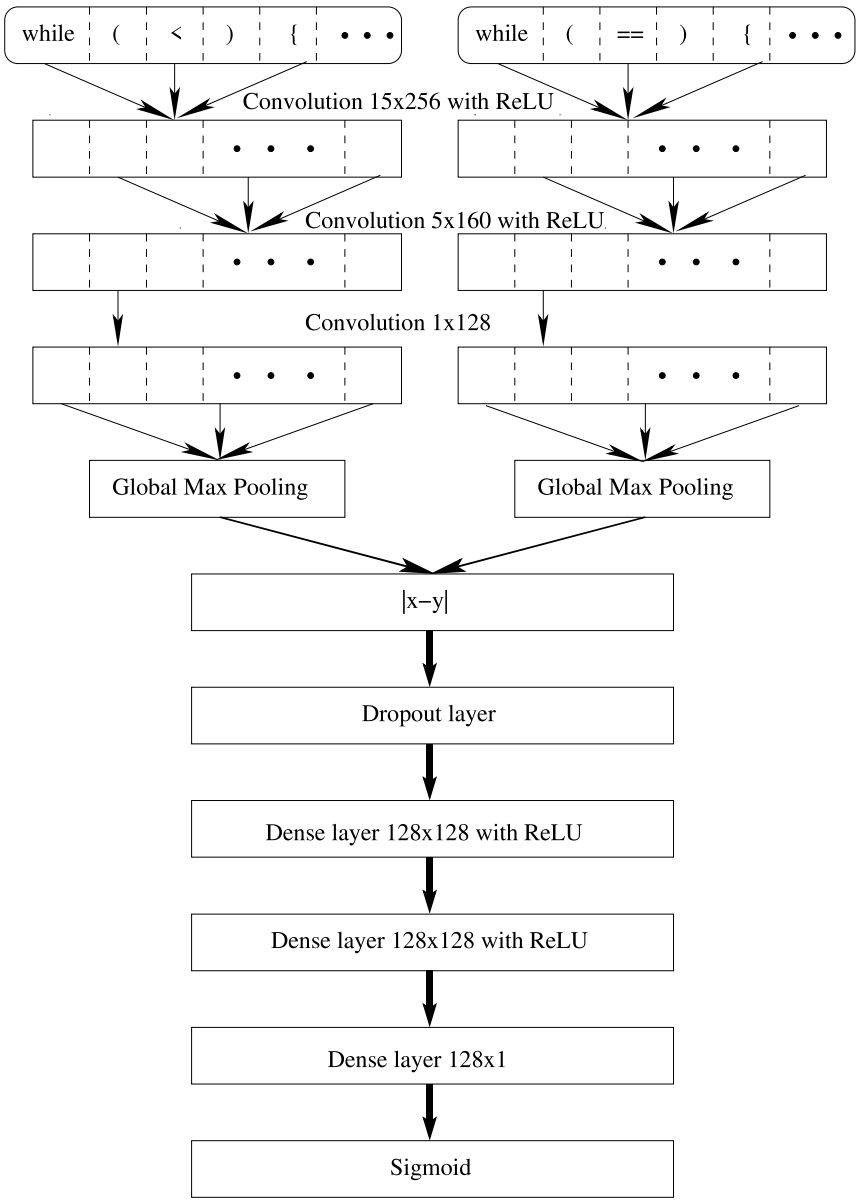

Figure 8: Siamese architecture for similarity analysis.

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 [55]. 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 [55]. 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 [55] 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 [55] 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 [55].

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 [56]. 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).