Can Machines Read Coding Manuals Yet? – A Benchmark for Building Better Language Models for Code Understanding

Ibrahim Abdelaziz[1], **Julian Dolby**[1], **Jamie McCusker**[2], **Kavitha Srinivas**[1]
[1]IBM Research, T.J. Watson Research Center
[2] Rensselaer Polytechnic Institute (RPI)
ibrahim.abdelaziz1@ibm.com, dolby@us.ibm.com, mccusj2@rpi.edu, kavitha.srinivas@ibm.com

Abstract

Code understanding is an increasingly important application of Artificial Intelligence. A fundamental aspect of understanding code is understanding text about code, e.g., documentation and forum discussions. Pre-trained language models (e.g., BERT) are a popular approach for various NLP tasks, and there are now a variety of benchmarks, such as GLUE, to help improve the development of such models for natural language understanding. However, little is known about how well such models work on textual artifacts about code, and we are unaware of any systematic set of downstream tasks for such an evaluation. In this paper, we derive a set of benchmarks (BLANCA - Benchmarks for LANguage models on Coding Artifacts) that assess code understanding based on tasks such as predicting the best answer to a question in a forum post, finding related forum posts, or predicting classes related in a hierarchy from class documentation. We evaluate performance of current state-of-the-art language models on these tasks and show that there is significant improvement on each task from fine tuning. We also show that multi-task training over BLANCA tasks help build better language models for code understanding.

1 Introduction

Code understanding is an increasingly important application of AI, with over 100 papers targeting the area in the last year alone[1]. Much research in this area has focused on understanding code from abstract representations of the program such as Abstract Syntax Trees (ASTs) and program flow. However, there has been little emphasis in utilizing important semantics about code buried in textual artifacts, such as documentation or forum discussions. Extracting such information can significantly enrich code representations. As an example, Figure 1 shows a program where the classes *GLM* and *SGDClassifier* are being used. If one could enrich the representation of the two classes in the code with their key features from text, we would understand that both represent linear models, and hence both code snippets perform similar functions.

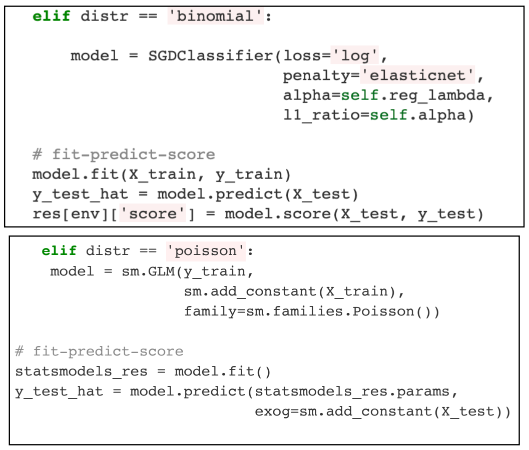
To enrich code with textual information, we need to be able to summarize textual information about classes and functions into vector representations. Pre-trained language models are an obvious choice, but we currently do not know how applicable they are to text about code, given the specialized language of the programming domain. We need a set of code-related downstream tasks to evaluate these models, just as GLUE (Wang et al. 2018) and SuperGLUE (Wang et al. 2019) have been used extensively to further language model development in the natural language understanding domain. CodeXGLUE (Lu et al. 2021) provides a suite of tasks but only a single task in it is related to textual code artifacts; it is translation of documentation about code from one natural language to another. To our knowledge, we know of no other tasks that focus on relations between textual artifacts about code. This paper attempts to fill this gap.

We have three goals in this paper: (a) design a suite of tasks we refer to as BLANCA (Benchmarks for LANguage models on Coding Artifacts) that can be used to train language models about the *semantics of code*, (b) evaluate whether existing models, fine-tuned for different aspects of natural language processing or different code oriented corpora, can perform well on these tasks, and (c) establish whether these tasks can be used to build better models for code understanding.

To construct these tasks, we relied on existing annotations in large public repositories such as GitHub (for code), StackOverflow, StackExchange and code documentation (for text about code). We exploited an integration of these sources in an open source dataset (Abdelaziz et al. 2021) to define the following five tasks focused on text about code:

- **Forum Answer Ranking (*R*)**. Some answers on forums have many votes or are selected as the best relative to others. Can language models predict the best answers?
- **Forum Link Prediction (*L*)**. Users of forum posts often point to other similar posts, which reflect semantically related posts compared to random pairs. Can language models predict links?
- **Forum to Class Prediction (*F*)**. Key features of classes or functions often get discussed in forum posts. Do language models discriminate related posts and class documentation from unrelated ones?
- **Class Hierarchy Distance Prediction (*H*)**. Code is often organized into class hierarchies. Do embedding distances from language models reflect distances in the hierarchy?

[1]https://ml4code.github.io/papers.html



```
  elif distr == 'binomial':

      model = SGDClassifier(loss='log',
                            penalty='elasticnet',
                            alpha=self.reg_lambda,
                            l1_ratio=self.alpha)

  # fit-predict-score
  model.fit(X_train, y_train)
  y_test_hat = model.predict(X_test)
  res[env]['score'] = model.score(X_test, y_test)
```

```
  elif distr == 'poisson':
    model = sm.GLM(y_train,
                   sm.add_constant(X_train),
                   family=sm.families.Poisson())

# fit-predict-score
statsmodels_res = model.fit()
y_test_hat = model.predict(statsmodels_res.params,
                           exog=sm.add_constant(X_test))
```

Figure 1: Usage in code

- **Class Usage Prediction (*U*)**. Similar code is often used in similar ways. Are embedding distances smaller for documentation about classes that are used similarly, and larger for dissimilar ones?

We compare performance on these tasks for seven language models, chosen for differences in architecture, training tasks, and corpora, as outlined in Section 3. Our main findings are as follows:

- Out of the box, language models trained on general corpora perform reasonably well on most BLANCA tasks, compared to models trained on code specific corpora such as CodeBERT (Feng et al. 2020) or BERTOverflow (Tabassum et al. 2020), attesting to the generality of these models.
- However, on every task, fine tuning on code specific models resulted in significant boost in performance, highlighting the usefulness of BLANCA tasks for building better language models.
- Multi-task training produced better performance on many BLANCA tasks, suggesting the tasks do help models learn code semantics that transfers across tasks.

To aid further research in code understanding, the code, datasets, the fine-tuned models and the leaderboard are publicly available[2] under an open source license (Eclipse for the code and Creative Common with Attribution for the data), and we hope they prove useful to the code understanding community to enrich representations of programs with textual information about classes and functions.

2 Related Work

There have been numerous benchmarks for code summarization or generation of code from natural language, and hence they have focused on collecting code and textual documentation that characterize the code. For these tasks, most have used the approach of generating code and its associated documentation strings, e.g., (LeClair and McMillan 2019), (Movshovitz-Attias and Cohen 2013). Similarly, code and corresponding textual documentation have been used for numerous tasks involving searching for code, e.g., (Li, Kim, and Chandra 2019), (Husain et al. 2019), or searching for posts given code, e.g., (Ponzanelli et al. 2014).

While such benchmarks are useful for joint embeddings of code and their associated text, they are restricted to tasks around code summarization, code generation, comment generation or code search; i.e., they do not directly help with the evaluation of language models for textual artifacts about code. Furthermore, most of the datasets in the literature do not correlate textual artifacts around code with code usage, with the exception of (Hu et al. 2018), which does link the generation of API sequence information from their usage in code to the problem of code summarization. The work in (Yang et al. 2017) connects code on GitHub to StackOverflow posts, but the latter dataset is not available. Again, their datasets are targeted to the task of code summarization, and code search respectively. Similarly StackOverflow posts have been used for tasks such as answer summarization (Cai et al. 2019), program repair (Liu and Zhong 2018) or generating code, e.g., (Liu and Zhong 2018). Finding directly related or duplicate posts is a recent task and dataset proposed in (Shirani et al. 2019), but there is no evaluation of any language model in that work.

Recently, (Tabassum et al. 2020) provided a BERTOverflow model for an in-domain representation of text about code. BERTOverflow is trained on 152 million StackOverflow questions over a BERT architecture, and has been fined tuned for software named entity recognition (e.g., finding mentions of operating systems in text). We use BERTOverflow as one of the models for the BLANCA tasks. There has also been work building structural language models from the abstract syntax trees, e.g., (Alon et al. 2019), which is clearly a related task, but the focus is once again on code. CodeXGLUE (Lu et al. 2021) provides a novel text-text benchmark which involves translation of documentation about code from one language to another, but that is arguably closer to natural language processing than code.

Thus, to the best of our knowledge, no work so far has examined how language models perform on a set of code related tasks for textual program artifacts, nor has there been much emphasis on building benchmarks to build better text representations for code understanding. BLANCA is built to address this gap.

3 Models

In choosing models for our experimentation, we needed language models to encode paragraphs in either class documentation or posts. We relied largely on the sentence transformers library (Reimers and Gurevych 2019), which provides a wide range of transformer models that have been fine-tuned for tasks such as information retrieval, paraphrase detection, and sentence similarity detection. These models have been shown to be effective in sentence and paragraph encoding style tasks. We also chose models with a different base, such as BERT (Devlin et al. 2019), XLM-RoBERTa (Conneau et al. 2020) and DistilBERT(Sanh et al. 2020). We

[2]https://github.com/wala/blanca/

Model name	Fine-tuning task
Universal Sentence Encoder‡	N/A
BERT-NLI†	Sentence similarity
DistilBERT-paraphrasing†	Paraphrase detection
xlm-r-paraphrase-v1†	Paraphrase detection
mmsmarco-DistilRoBERTa†	Information Retrieval
BERTOverflow†	StackOverflow/NER
CodeBERT-mlm†	NL-PL pairs in 6 languages

Table 1: Models used as baselines. Sources were tensorflow-hub, and SBERT. bert-base-nli-stsb-mean-tokens, distilroberta-base-paraphrase-v1, xlm-r-distilroberta-base-paraphrase-v1, msmarco-distilroberta-base-v2 and microsoft/codebert-base-mlm are their corresponding names in SBERT.

	Data type	Train	Test
Forum Answer Ranking	Question-answer pairs	450,000	50,000
Forum Link Prediction	Question-Question pairs	23,516	5,854
Forum to Class Prediction	Question-Class pairs	11,488	1,275
Class Hierarchy Prediction	Class-Class pairs	16,215,400	1,801,716
Class Usage Prediction	Class-Class pairs	75,862	8,439

Table 2: BLANCA's tasks and dataset statistics

also added a non-transformer style model (Google's Universal Sentence Encoder (Cer et al. 2018)), and models fine-tuned on StackOverflow posts (BERTOverflow (Tabassum et al. 2020)) and code documentation (CodeBERT (Feng et al. 2020)) to see if domain-specific training is helpful. We did not consider models, such as CuBERT (Kanade et al. 2020), designed only for code, and not text about code. The reason is that cuBERT's vocabulary is based on programming language tokens for Java or Python, which is only partially useful for text about code. Table 1 shows the types of base models used in our evaluation, using the names from the sentence-transformers (SBERT[3]) library.

We also tested if fine-tuning on each task would enhance performance, to establish whether the tasks can be used to build a better language model. For fine-tuning, we started either with BERTOverflow or CodeBERT, with the assumption that an in-domain representation would provide some advantage. We also examined whether multi-task training would improve performance, to see if better models could be built from using a combination of BLANCA tasks.

4 Tasks

All our datasets describe code artifacts in Python, and are derived from GraphGen4Code[4], which links 1.3 million programs of Python code to associated posts and class-documentation (Abdelaziz et al. 2021). For multi-task fine tuning, we report, for each task, the model with the best performance, and we outline its characteristics. In Section 4.6, we discuss more general findings for multi-task training. Performance on tasks is encoded as follows in tables: (1) Forum Answer Ranking (R), (2) Forum Link Prediction (L), (3) Forum Class Prediction (F), (4) Class Hierarchy Prediction (H), and (5) Class Usage Prediction (U). Table 2 lists each BLANCA task and the corresponding train/test data sizes.

Dataset Annotation Quality Two of BLANCA tasks are based on manually curated datasets by millions of users such as ranking answers in StackOverflow forums (Forum Answer Ranking) and manually linking similar posts (Forum Link Prediction). These data are high quality, in the sense that they are crowd annotated by humans, which is how most gold standards get constructed. Class Hierarchy and Class Usage Prediction tasks are both based on objective properties of code artifacts (class hierarchy and similarities among classes in terms of their methods, respectively), so once again, the issues of data quality do not arise. The only task where we did not have explicit human labeling for every example is Forum to Class Prediction. In this task, we relied on heuristics to automatically label the data. Furthermore, to assure quality, we performed a manual evaluation of a sample with three human annotators (see Section 4.3).

Hyperparameter Search for Finetuning We started with the default parameters of our base models; CodeBERT and BERTOverflow. We also tried to use Population Based Training from RayTune[5] to perform hyper-parameter search for the Forum Answer Ranking (R) and Forum Link Prediction (L) tasks. However, we did not get better performance compared to using the default parameters from the corresponding base models.

We describe below how we formulated each task, the dataset definition process and the performance of various language models on it.

4.1 Forum Answer Ranking (R)

Task Description StackOverflow and StackExchange contain questions and answers. Accepted answers are manually annotated and most answers have a vote count. The core task here is to predict the best answer to each question, and order the answers by their popularity.

Dataset We generated a dataset of 500K questions such that each question comes with at least three answers. The average number of answers per question in this dataset is 4.9 answers, and the average number of votes per question is 23.5 and per answer is 12.74. The train and test tasks were split 90-10, so the train set had 450,000 questions and test had 50,000 questions. To build the fine tuning model, we modeled this as a task similar to training on the Semantic Textual Similarity Benchmark (STSB) adopted by SBERT. Each answer was ranked according to popularity, and ties were broken by adding only one of the answers that were tied. The ranks were then converted to a score between 0 (worst rank) and 1 (best rank), with a cosine similarity loss, and an embedding similarity evaluator from the SBERT library. Fine tuning was performed on BERTOverflow and CodeBERT models, with the 90% of training data for training, 10% of the training data for validation, for 10 epochs.

[3]https://github.com/UKPLab/sentence-transformers

[4]https://wala.github.io/graph4code/

[5]https://docs.ray.io/en/latest/tune/index.html

	MRR	NDCG
DistilBERT-paraphrasing	0.5937 (.001)	0.8393 (.001)
BERT-NLI	0.5972 (.001)	0.8407 (.001)
msmarco-DistilRoBERTa	0.5992 (.001)	0.8427 (.001)
xlm-r-paraphrase-v1	0.5977 (.001)	0.8411 (.001)
USE	0.6114 (.001)	0.8483 (.001)
BERTOverflow	0.5910 (.001)	0.8375 (.001)
CodeBERT	0.5926(.001)	0.8375 (.001)
FT-BERTOverflow	0.6743 (.001)	0.8823(.001)
FT-CodeBERT	0.6671(.001)	0.8790(.001)
RFLHU-BERTOverflow	**0.6879 (.001)**	**0.8893 (.001)**

Table 3: Performance of language models on forum answer ranking (R). The numbers in parentheses are the standard errors of the sample mean. FT represents fine tuning on R alone, RFLHU-BERTOverflow is the best multi-task training model.

Evaluation To capture how well the embeddings of different language models identified the ranking of answers, we computed the cosine distances between the question embedding and the embedding of each of the answers, and ranked answers by nearest in cosine distance to furthest. We report standard information retrieval metrics of average Mean Reciprocal Rank (MRR) and average Normalized Discounted Cumulative Gain (NDCG) on this predicted ranking.

Table 3 shows that most language models do reasonably well on this task, which is not surprising because text in forum posts is mostly natural language. Surprisingly though, there is no benefit for the base BERTOverflow model that has been tuned on StackOverflow posts compared to the rest of non-finetuned models. However, fine-tuned BERTOverflow does much better, which is consistent with our hypothesis that it is possible to use these tasks for building better language models. Across many tasks, fine-tuning on BERTOverflow produced better performance than finetuning on CodeBERT, which suggests that forum discussions contain in most cases, the right mixture of explanations in natural language along with code. Moreover, the best performance was achieved with multi-task finetuning (RFLHU-BERTOverflow), which suggests that use of multiple BLANCA tasks builds better language models for textual code artifacts.

4.2 Forum Link Prediction (L)

Task Description Forum posts with links to one another are usually related compared to unlinked posts; we investigate if language models place such related post pairs closer in vector space. We focus on embedding distance because it is a more direct metric for assessing the quality of the embedding rather than classification accuracy.

Dataset For this task, we generated 23,516 pairs of posts for training (11,758 positive and 11,758 negative), 5,854 pairs (2,727 positive and 2,727 negative) for testing. Finetuning was set up as a classification task in SBERT, with the use of contrastive loss along with a binary classification evaluator from the SBERT library. All other training details were similar to the forum answer ranking task.

Model	Linked	Unlinked	T
DistilBERT-paraphrasing	0.38	0.71	112.49
BERT-NLI	0.31	0.53	74.92
msmarco-DistilRoBERTa	0.34	0.74	110.42
xlm-r-paraphrase-v1	0.37	0.70	105.02
USE	0.34	0.74	142.04
BERTOverflow	0.20	0.31	59.52
CodeBERT	0.03	0.04	19.39
FT-BERTOverflow	0.09	0.52	180.42
FT-CodeBERT	0.08	0.50	147.21
RFLHU-BERTOverflow	**0.08**	**0.58**	**198.10**

Table 4: Cosine distance between linked and unlinked posts (L). FT represents fine-tuning on L alone.

Relevant to this task, (Shirani et al. 2019) recently introduced a similar benchmark for predicting relatedness in StackOverflow posts focused on Java code, as opposed to our dataset which is language agnostic. Their dataset contains 300K of linked pairs categorized into 1) duplicates: questions in StackOverflow marked by moderators as duplicates, 2) direct: explicitly linked posts, 3) indirectly or transitively connected posts through a direct or a duplicate link and 4) isolated or unlinked posts. Direct and isolated links are similar to our positive and negative examples. We evaluate all our models' ability to differentiate these link types. Note that we did not use this data for fine-tuning a model which discriminates the different categories; but one might expect direct links and duplicates to be closer in embedding distance, and isolated links to be the furthest, with indirect links in the middle. Shirani et al. (2019) did not evaluate this with any of the language models, so we examine whether these categories of relatedness of posts is reflected in embeddings of pre-trained models.

Evaluation As shown in Table 4, all language models showed a statistically significant difference ($p \leq .01$) on independent sample t-tests between linked and unlinked posts. BERTOverflow with fine-tuning (both versions tuned on L only and RFLHU) performing the best in terms of pulling apart linked and unlinked posts. We note that the size of T value normalizes the distance between linked and unlinked posts by their variance; that is, the T value captures not only the average distance but also the separation between the two distributions. Our focus then is on the absolute value of that separation as provided by the T value. Figure 2 shows this visually. We note that BERTOverflow and CodeBERT as base models discriminated least between linked and unlinked posts, but fine-tuning clearly helped greatly. This is evident in the solid and dashed lines for RFLHU-BERTOverflow where it shows little overlap between linked and unlinked posts.

Figure 3 shows the results of a variety of language models for question relatedness variant of this task (Shirani et al. 2019). We ensured that none of Shirani et al. (2019)'s test set examples were used in our training set. Across all models, directly related questions are closest in embedding space

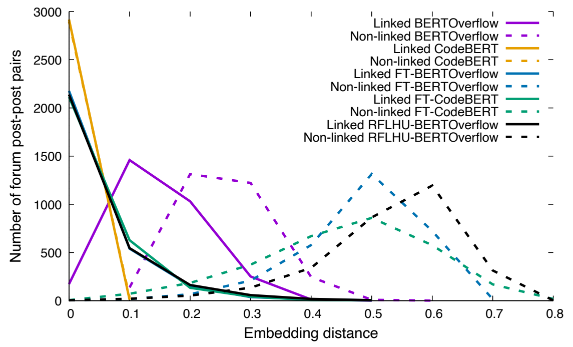


Figure 2: Linked versus unlinked pair distances for all models (L).

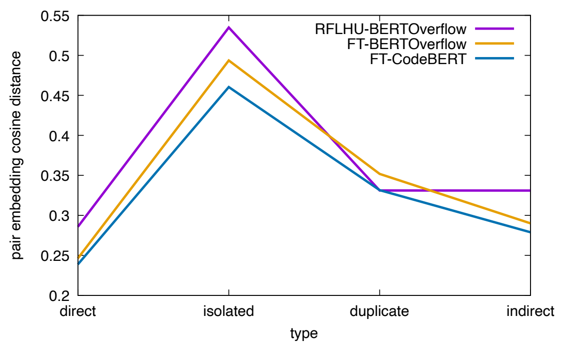


Figure 3: Direct, indirect, duplicate and isolated pair distances for all models.

followed by indirectly related questions. Questions marked duplicate posts were similar to the indirect questions only in the RFLHU model, which seemed to be picking up relatedness in both indirect and duplicate questions. We note that questions marked duplicates in forums are only duplicates at a level of coding abstraction. For example, the two questions "How to return multiple objects from a Java method?" and "Java how to return two variables?" are a duplicate pair. Although the two questions talk about the same problem, the discussions and even the solutions are different. Therefore, cosine similarity between them is not as close as one would expect. Finally, isolated question pairs are the most distant compared to all other pairs across all models (all differences from isolated pairs to direct, indirect and duplicate pairs were statistically significant at the .01 level). Multi-task fine-tuning (RFLHU-BERTOverflow) clearly helped the best in getting semantically related posts closer and pulling apart the unrelated ones.

4.3 Forum to Class Prediction (F)

Task Description Forum posts often describe specific code artifacts in text, where they discuss key features of a class or a function. A key question is whether a model can predict if a post about a class and documentation of the same class are related.

Model	Related	Unrelated	T
DistilBERT-paraphrasing	0.55	0.68	16.61
BERT-NLI	0.45	0.60	14.73
msmarco-DistilRoBERTa	0.45	0.66	20.37
xlm-r-paraphrase-v1	0.53	0.67	17.15
USE	0.53	0.74	20.67
BERTOverflow	0.33	0.47	18.31
CodeBERT	0.06	0.09	12.23
FT-BERTOverflow	0.07	0.77	46.88
FT-CodeBERT	0.08	0.82	50.07
RFLHU-CodeBERT	**0.11**	**0.66**	**53.98**

Table 5: Cosine distance between documentation-post pairs (F) that are related and unrelated. FT represents fine-tuning on F alone.

Dataset In order to find posts that were more focused on discussions of a specific class or function's features, we queried an ElasticSearch index of posts with a query per class as in GraphGen4Code (Abdelaziz et al. 2021), insisting that the class and its package be both mentioned in the question. These constituted our positive class-post examples. For negative examples, we chose hard negatives, requiring that both class name and its package not be mentioned anywhere within the question and its answers; but nevertheless the post matched either class or package names. To ensure the quality of this data, we asked 3 annotators to label a random sample of 100 examples; 50 positive and 50 negative. This manual inspection revealed that negatives were in fact negatives, in the sense that even if the class was mentioned, it was usually, from a different package, or very often from different programming languages (e.g. Java, Javascript, etc). The average hit and miss rates from the three annotators were 96.7% and 3.3%, respectively. In this task, we created 8,827 negative examples and 2,661 positives for training, and 980 negative examples and 295 positive examples for testing. Fine-tuning the model was analogous to the forum link prediction task.

Evaluation As shown in Table 5, all language models showed a statistically significant difference ($p \leq .01$) on independent sample t-tests between positive and negative class-post examples. Again, fine-tuning helps improving the performance on this task significantly; e.g. single task tuning of FT-CodeBERT vs. CodeBERT and FT-BERTOverflow compared to BERTOverflow. Fine-tuning on multiple tasks, e.g. RFLHU-CodeBERT, gave better performance compared to the single-task tuned models, FT-CodeBERT and FT-BERTOverflow. As shown in Figure 4, the distance was greatly enhanced by fine-tuning e.g. BERTOverflow vs. fine-tuned RFLHU-BERTOverflow.

4.4 Class Hierarchy Distance Prediction (H)

Task Description Semantically related classes tend to be linked by developers in a class hierarchy, so its reasonable to ask if neural embeddings of related classes cluster closer together in a class hierarchy. We structured this as a class distance prediction task, with class distances ranging from 1 to 10.

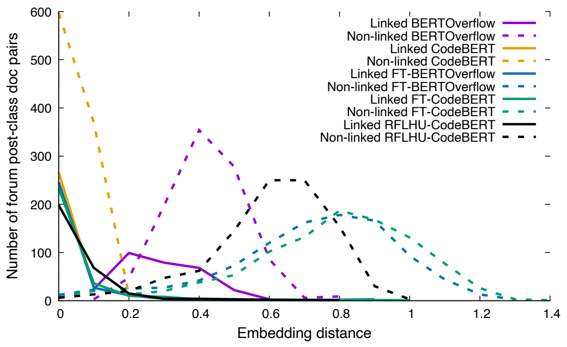


Figure 4: Related and unrelated documentation-post pair distances for some fine-tuned and non-fine-tuned models (F).

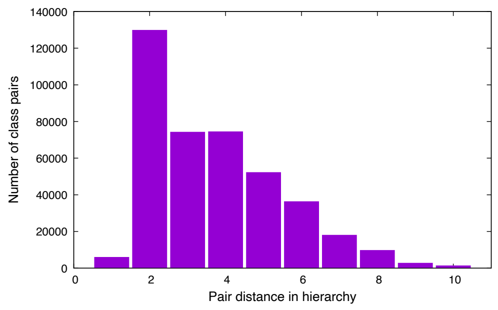


Figure 5: Number of class-pairs by class distance.

Dataset We collected the documentation associated with 257,655 classes in GraphGen4Code (Abdelaziz et al. 2021), but many of these represent different names that resolve to the same class. We aliased the classes to its canonical version by loading the class dynamically to obtain its runtime name, and added in classes that we could not load for some reason, which resulted in 90,464 classes.

To get classes related by distance, we created an undirected graph of class to superclass relations for every module, being careful not to add edges from any class to the class `object`. For each module graph, we computed distances between every pair of classes using an all pairs shortest paths algorithm. We eliminated pairs with distances greater than 10, and this resulted in a set of pairs that we split randomly such that 16,215,400 million pairs of classes were in train, and 1,801,716 million pairs were in test. For fine-tuning, we structured this similar to the forum ranking task, with distances translated to scores between 0 (least related) and 1 (most related), and we used cosine similarity loss, coupled with a embedding similarity evaluator from SBERT. Training on 16.2 million pairs was computationally expensive so we trained it on a random sample of 100,000 training examples, 10,000 of which was used for validation. Figure 5 shows the distribution of embedding distances for each class distance (1-10) to show the dataset characteristics.

Evaluation Since this is a regression task, we evaluated the Pearson r correlation, which as shown in Table 6 varied from 0.17 (for BERTOverflow) to 0.34 (for Fine-tuned-BERTOverflow); all are statistically significant at $p \leq 0.01$. Regression for each model is shown in Figure 6. The improvement from fine-tuning for BERTOverflow showed that the task is useful for building better embeddings. Some other models showed reasonable performance with no tuning (xlm-r-paraphrase at 0.27, and USE at 0.28 respectively), so there is clearly a room to improve these different base models as well, but we leave that issue for future work, since our goal is more on task development rather than building better models.

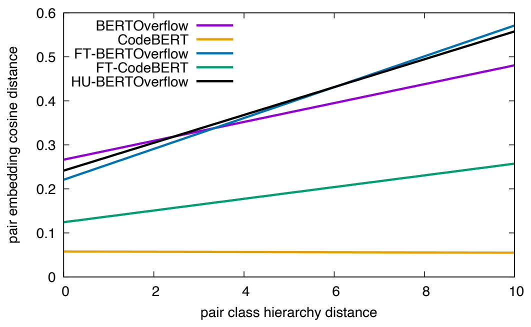


Figure 6: Prediction of embedding distance from class distance (H) for all models. Standard error of regression was less than 0.0002 for all models.

Model	Pearson r
DistilBERT-paraphrasing	0.26
BERT-NLI	0.20
msmarco-DistilRoBERTa	0.23
xlm-r-paraphrase-v1	0.27
USE	0.28
BERTOverflow	0.17
CodeBERT	-0.01
FT-BERTOverflow	**0.34**
FT-CodeBERT	0.24
HU-BERTOverflow	0.29

Table 6: Correlation of class hierarchy (H) distance to embedding distance by model. FT represents fine-tuning on H alone.

4.5 Class Usage Prediction (U)

Task Description GitHub contains millions of programs, where classes are used in code to achieve some purpose. Classes that are used in the same way; i.e., same set of methods get invoked on them, might be expected to be rated as more similar than classes that do not share any methods. We structured this as a similarity rating task.

Dataset To construct this dataset, we used the GraphGen4Code knowledge graph (Abdelaziz et al. 2021) which has data flow graphs for 1.3 million GitHub programs. Dataflow tracks the flow of data through return values and parameters within a program. As an example, for the program snippet shown in Figure 1, dataflow would show that `fit` and `predict` calls occur on objects returned by calls

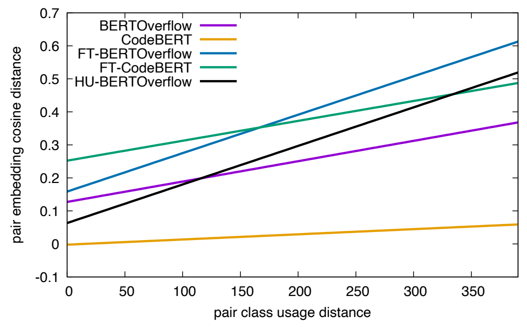


Figure 7: Prediction of embedding distance by class usage (U) similarity for all models. Standard error of regression was less than 1.0e-4 for all models.

Model	Pearson r
DistilBERT-paraphrasing	0.35
BERT-NLI	0.17
msmarco-DistilRoBERTa	0.34
xlm-r-paraphrase-v1	0.36
USE	0.41
BERTOverflow	0.37
CodeBERT	0.33
FT-BERTOverflow	0.52
FT-CodeBERT	0.30
HU-BERTOverflow	**0.61**

Table 7: Correlation of class usage similarity (U) with embedding distance by model. FT represents fine-tuning on U alone.

Model	R	F	L	H	U
RFLHU-BERTOverflow	**0.69/0.89**	46.49	**198.10**	0.15	0.27
RFLHU-CodeBERT	0.68/0.88	**53.98**	148.72	0.12	0.38
RFLH-BERTOverflow	0.68/0.89	47.56	188.73	0.17	0.14
RFLH-CodeBERT	0.67/0.88	49.04	141.97	0.10	0.14
RFL-BERTOverflow	0.68/0.88	48.81	197.63	0.13	0.25
RFL-CodeBERT	0.68/0.89	53.29	144.17	0.09	0.26
HU-BERTOverflow	0.59/0.84	15.89	68.43	**0.29**	**0.61**
HU-CodeBERT	0.61/0.85	12.95	45.50	0.05	0.41

Table 8: Answer Ranking (R) numbers are MRR/NDCG, Hierarchy (H) and Usage (U) tasks are correlation where as Linked Posts (L) and Forum to Doscstrings (F) are T-statistic.

to the constructors of `SGDClassifier` and `GLM`. In this example, `SGDClassifier` shares 2 methods (denoted as M) with 1 class (denoted as C), which in this instance is `GLM`). The classes are similar, in the sense that they both share the same methods in usage, but the degree of similarity is dependent on the number of shared methods (M), and the number of classes that have the same methods (C). The smaller the C, the more likely it is that a pair is similar, and the larger the M the more likely it is that the pair is similar. To capture both dimensions of similarity into a single distance metric for learning, we defined an 'ideal' class pair in terms of our data - that is a vector with $[max(M), min(C)]$. We used the Euclidean distance of each class pair from this ideal vector as the dissimilarity metric. Given a pair, the task then is to predict if the classes were similar or distant based on their usage.

The train task contains 75,862 class pairs, and the test task contains 8,439 pairs, with an average distance of 312.21 for train pairs and an average distance of 312.12 for test pairs, suggesting the two had similar characteristics. The fine-tuning task was modeled the same as the class hierarchy prediction task.

Evaluation We frame this task as a regression task and evaluate the Pearson r correlation once again for all models, which varied from 0.17 (for BERT-NLI) to 0.61 (for HU-BERTOverflow) as shown in Table 7; all results are statistically significant at $p \leq 0.01$. The improvement from fine-tuning for BERTOverflow (0.37 to 0.52) also shows the task is useful for building better embeddings. Using hierarchy task as well with HU-BERTOverflow further improved performance to 0.61. This was not the case though for CodeBERT models with and without fine-tuning where its performance dropped to 0.30 from 0.33. We also show in Figure 7 the effectiveness of usage distance as a predictor of cosine embedding distance.

4.6 Multi-Task Training

We focus our multi-task training discussion on BERTOverflow, because combining it with training produced the best performance consistently. As shown in Table 8 tasks that derived from code properties (usage (U) and hierarchy (H)) did not benefit from training on ranking (R), forum to class (F) or linked posts (L) tasks on BERTOverflow, which suggests that tasks derived from code properties require different features than those emphasized by RFL tasks. Tasks derived from code properties (HU) however helped RFL tasks, suggesting the importance of having a diversity of tasks for tuning. We were expecting and found class hierarchy training to help the usage task, since code that is closely-related in the type hierarchy tends to have similar usage due to the nature of classes; this was confirmed by our findings. We also expected usage analysis to help class hierarchy training, because we expected parameter types of methods to relate to the class hierarchy; this did not happen, perhaps due to the dynamically-typed nature of Python, where distinct types can share method names. We expect this to be different in typed languages such as Java, and we plan to investigate it in our future work.

5 Conclusions

In this paper, we presented BLANCA, a set of benchmarks to help further research in code understanding from textual manuals and posts about code. We used BLANCA tasks to show that one can build better language models for understanding code artifacts. We also used multi-task training to demonstrate better representations of classes and functions from these models. We hope these will be useful in enriching code representations with their textual semantics embedding in natural language artifacts of code.

References

Abdelaziz, I.; Dolby, J.; McCusker, J.; and Srinivas, K. 2021. A Toolkit for Generating Code Knowledge Graphs. In *Proceedings of the 11th on Knowledge Capture Conference*, K-CAP '21, 137–144. New York, NY, USA: Association for Computing Machinery. ISBN 9781450384575.

Alon, U.; Sadaka, R.; Levy, O.; and Yahav, E. 2019. Structural Language Models for Any-Code Generation. *CoRR*, abs / 1910.00577.

Cai, L.; Wang, H.; Xu, B.; Huang, Q.; Xia, X.; Lo, D.; and Xing, Z. 2019. AnswerBot: An Answer Summary Generation Tool Based on Stack Overflow. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, 1134–1138. New York, NY, USA: Association for Computing Machinery.

Cer, D.; Yang, Y.; Kong, S.-y.; Hua, N.; Limtiaco, N.; St. John, R.; Constant, N.; Guajardo-Cespedes, M.; Yuan, S.; Tar, C.; Strope, B.; and Kurzweil, R. 2018. Universal Sentence Encoder for English. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 169–174. Brussels, Belgium: Association for Computational Linguistics.

Conneau, A.; Khandelwal, K.; Goyal, N.; Chaudhary, V.; Wenzek, G.; Guzmán, F.; Grave, E.; Ott, M.; Zettlemoyer, L.; and Stoyanov, V. 2020. Unsupervised Cross-lingual Representation Learning at Scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, 8440–8451. Association for Computational Linguistics.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171–4186. Minneapolis, Minnesota: Association for Computational Linguistics.

Feng, Z.; Guo, D.; Tang, D.; Duan, N.; Feng, X.; Gong, M.; Shou, L.; Qin, B.; Liu, T.; Jiang, D.; and Zhou, M. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. Cite arxiv:2002.08155Comment: Accepted to Findings of EMNLP 2020. 12 pages.

Hu, X.; Li, G.; Xia, X.; Lo, D.; Lu, S.; and Jin, Z. 2018. Summarizing Source Code with Transferred API Knowledge. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, 2269–2275. International Joint Conferences on Artificial Intelligence Organization.

Husain, H.; Wu, H.-H.; Gazit, T.; Allamanis, M.; and Brockschmidt, M. 2019. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. arXiv:1909.09436.

Kanade, A.; Maniatis, P.; Balakrishnan, G.; and Shi, K. 2020. Learning and evaluating contextual embedding of source code. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 12-18 July 2020*, Proceedings of Machine Learning Research. PMLR.

LeClair, A.; and McMillan, C. 2019. Recommendations for Datasets for Source Code Summarization. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 3931–3937. Minneapolis, Minnesota: Association for Computational Linguistics.

Li, H.; Kim, S.; and Chandra, S. 2019. Neural Code Search Evaluation Dataset. arXiv:1908.09804.

Liu, X.; and Zhong, H. 2018. Mining stackoverflow for program repair. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 118–129.

Lu, S.; Guo, D.; Ren, S.; Huang, J.; Svyatkovskiy, A.; Blanco, A.; Clement, C. B.; Drain, D.; Jiang, D.; Tang, D.; Li, G.; Zhou, L.; Shou, L.; Zhou, L.; Tufano, M.; Gong, M.; Zhou, M.; Duan, N.; Sundaresan, N.; Deng, S. K.; Fu, S.; and Liu, S. 2021. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. *CoRR*, abs/2102.04664.

Movshovitz-Attias, D.; and Cohen, W. W. 2013. Natural Language Models for Predicting Programming Comments. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 35–40. Sofia, Bulgaria: Association for Computational Linguistics.

Ponzanelli, L.; Bavota, G.; Di Penta, M.; Oliveto, R.; and Lanza, M. 2014. Mining StackOverflow to Turn the IDE into a Self-Confident Programming Prompter. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, 102–111. New York, NY, USA: Association for Computing Machinery. ISBN 9781450328630.

Reimers, N.; and Gurevych, I. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Sanh, V.; Debut, L.; Chaumond, J.; and Wolf, T. 2020. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv:1910.01108.

Shirani, A.; Xu, B.; Lo, D.; Solorio, T.; and Alipour, M. 2019. Question Relatedness on Stack Overflow: The Task, Dataset, and Corpus-inspired Models. *ArXiv*, abs/1905.01966.

Tabassum, J.; Maddela, M.; Xu, W.; and Ritter, A. 2020. Code and Named Entity Recognition in StackOverflow. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 4913–4926. Online: Association for Computational Linguistics.

Wang, A.; Pruksachatkun, Y.; Nangia, N.; Singh, A.; Michael, J.; Hill, F.; Levy, O.; and Bowman, S. 2019. SuperGLUE: A stickier benchmark for general-purpose language understanding systems. *Advances in Neural Information Processing Systems*, 32.

Wang, A.; Singh, A.; Michael, J.; Hill, F.; Levy, O.; and Bowman, S. 2018. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In

Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP, 353–355. Brussels, Belgium: Association for Computational Linguistics.

Yang, D.; Martins, P.; Saini, V.; and Lopes, C. 2017. Stack Overflow in Github: Any Snippets There? In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 280–290.