MQDD: Pre-training of Multimodal Question Duplicity Detection for Software Engineering Domain

Jan Pašek, Jakub Sido, Miloslav Konopík, Ondřej Pražák
{pasekj,sidoj,konopik,ondfa}@kiv.zcu.cz
NTIS – New Technologies for the Information Society,
Department of Computer Science and Engineering,
Faculty of Applied Sciences, University of West Bohemia, Plzeň, Czech Republic

Abstract

This work proposes a new pipeline for leveraging data collected on the Stack Overflow website for pre-training a multimodal model for searching duplicates on question answering websites. Our multimodal model is trained on question descriptions and source codes in multiple programming languages. We design two new learning objectives to improve duplicate detection capabilities. The result of this work is a mature, fine-tuned Multimodal Question Duplicity Detection (MQDD) model, ready to be integrated into a Stack Overflow search system, where it can help users find answers for already answered questions. Alongside the MQDD model, we release two datasets related to the software engineering domain. The first Stack Overflow Dataset (SOD) represents a massive corpus of paired questions and answers. The second Stack Overflow Duplicity Dataset (SODD) contains data for training duplicate detection models.

1 Introduction

The benefits of Question-Answer (QA) networks for software developers such as the Stack Overflow website are widely exploited by professionals and beginners alike during the software creation process. Many solutions to various problems, short tutorials, and other helpful tips can be found on these networks. However, access to this valuable source of information highly depends on users' ability to search for the answers. In our paper, we introduce a multimodal method for detecting duplicate questions. Apart from the primary use to prevent posting duplicate questions, this technique can be directly used for better search. When users are posting already answered questions, they can get the answer immediately without the necessity to wait until someone else links the duplicate post or answers their question.

The duplicate question detection task aims to classify whether two questions share the same intent. In other words, if two questions are duplicates, they relate to the same answer. The duplicate detection task is quite challenging since the classifier needs to distinguish tiny semantic nuances that can significantly change the desired answer.

The posts in the QA networks for software development often intermix natural language with source code snippets. The great success of neural networks for Natural Language Processing (NLP) encourages us to build a bi-modal natural language (NL) and programming language (PL) encoder for duplicate detection (Wang et al., 2020) on question-answering platforms such as Stack Overflow.

Current state-of-the-art NLP methods build on large pre-trained models, leveraging Transformer architecture (Vaswani et al., 2017). The Transformer-based models such as BERT (Devlin et al., 2018), GPT (Brown et al., 2020), RoBERTa (Liu et al., 2019), or T5 (Raffel et al., 2019) are usually pre-trained on massive unlabeled corpora and applied to a task with much less training data afterward. We follow this idea and introduce the pre-training phase into our solution. To achieve the best possible results, we design duplicate-detection-specific pre-training objectives (see Section 3.3).

Since the source code snippets present in the Stack Overflow questions may be relatively long, we choose to base our model on the Longformer architecture (Beltagy et al., 2020); whose modified attention scheme scales linearly with the sequence length. The resulting model with ≈146M parameters is firstly pre-trained on a large semi-supervised corpus of Stack Overflow questions and answers. For detailed information about the dataset and pre-training, see Section 3.

Afterward, in Section 4, we fine-tune the obtained model on the duplicate detection task and compare our model with CodeBERT (Feng et al.,

Proceedings of Recent Advances in Natural Language Processing, pages 824–835
Varna, Sep 4–6, 2023
https://doi.org/10.26615/978-954-452-092-2_089

2020), which represents another NL-PL multimodal encoder. We also compare our model to a randomly initialized Longformer (Beltagy et al., 2020) and pre-trained RoBERTa (Liu et al., 2019) to see whether the pre-training of both models brings a significant improvement of the achieved results. The previously described experiments are visualized in Figure 1. At the end of this paper, we explore how well our model generalizes to other tasks by applying our model to the CodeSearchNet dataset (Husain et al., 2019) in Section 5.

Our main contributions are: 1) We release a fine-tuned Multimodal Question Duplicity Detection (MQDD) model for duplicate question detection. The model is mature enough to be deployed to Stack Overflow, where it can automatically link duplicate questions and, therefore, improve users' ability to search for desired answers. Furthermore, we release the pre-trained version of the encoder, so other researchers may reuse the most computationally intensive phase of our model training. 2) We present and explore the effect of entirely new pre-training objectives specially designed for duplicate detection. 3) We release a *Stack Overflow Dataset* (SOD) that can be used for pre-training models in a software engineering domain. Furthermore, we release a novel *Stack Overflow Duplicity Dataset* (SODD) for duplicate question detection, enabling other researchers to follow up on our work seamlessly.

2 Related Work

The naturally collected massive amounts of data in software management systems, issue tracker tools, and versioning systems makes the software development an ideal domain to apply deep models to increase work effectiveness.

Codex (Chen et al., 2021) represents a large pre-trained neural network model that can generate source code for the software engineering domain. It is designated for source code generation. Its slightly modified form is also integrated with the *GitHub Copilot*[1] system, a digital pair programmer. CodeT5 (Wang et al., 2021) is another model that also works with source code. It demonstrates the capability of solving multiple tasks thanks to converting all problems into a unified sequence-to-sequence form. Different approach is introduced in the paper by Sun et al. (2022), which translates source codes into a natural language to retrieve similar code snippets.

The previous papers build upon the architecture of the Transformer (Vaswani et al., 2017), which can be pre-trained on a massive corpus on unlabeled data, and applied on a downstream task only with much less demanding fine-tuning. This approach is used by BERT (Devlin et al., 2018), which employs the Transformer encoder to produce contextual representations of input tokens. These contextual embeddings (Peters et al., 2018; McCann et al., 2017) can then be utilized for various tasks, including the classification of entire sequences (Reimers and Gurevych, 2019) or individual tokens (Liu et al., 2021; Sun et al., 2019). Such success can probably be attributed to a well-designed attention mechanism (Bahdanau et al., 2014), which allows the model to capture contextual information from the entire sequence being processed.

The results obtained using large pre-trained model can be significantly influenced by the correct choice of training objective. Adapting the pre-training phase and finding a proper objective allows the model to exploit useful features from large source of data. For example, RoBERTa (Liu et al., 2019) slightly modifies the Masked Language Modeling (MLM) objective and abandons the Next Sentence Prediction (NSP) to improve the achieved results. Different way of improving results is represented by the changes in the architecture of the model. For example, Longformer (Beltagy et al., 2020) model significantly modifies the attention mechanism to mitigate the $\mathcal{O}(N^2)$ complexity of a vanilla attention enabling processing of longer sequences.

The whole concept of pre-trained encoders laid out by BERT (Devlin et al., 2018) is often applied to multimodal data as well. This enables, for example, a unified processing source codes and natural texts. The produced contextual embeddings of source code and text (Chen and Monperrus, 2019) is then directly applicable to downstream tasks such as code similarity, code search, or code fixing (Le et al., 2020).

The CuBERT (Kanade et al., 2020a) is an example of a multimodal encoder for Python source codes and texts. The model outperforms BiLSTM (Schuster and Paliwal, 1997; Kanade et al., 2020b) and randomly initialized Transformer (Vaswani et al., 2017) approach in five different tasks, including classification of variable misuse, wrong binary

[1] https://copilot.github.com

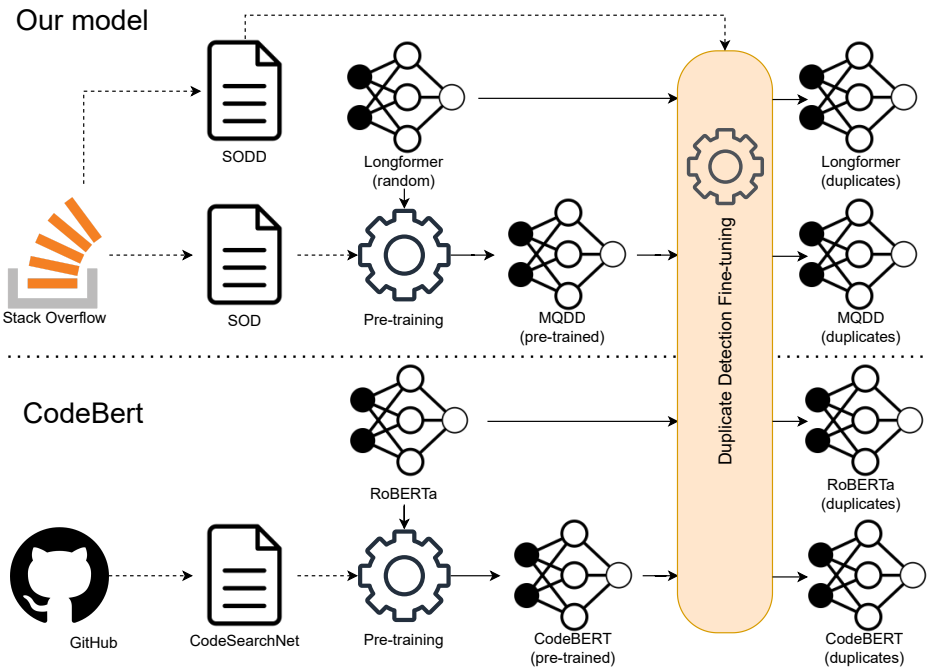

Figure 1: A visualization of the pipeline of our experiments. The upper part of the figure shows the construction of our SOD and SODD datasets and their usage for pre-training and fine-tuning our MQDD model. The lower part of the figure visualizes the pre-training of the CodeBERT done by Feng et al. (2020).

operator usage, swapped operands, and function-docstring match. Another representative of multimodal source code encoders is the CodeBERT model (Feng et al., 2020) pre-trained on a multilingual corpus of source codes from six different programming languages. The CodeBERT builds upon the RoBERTa (Liu et al., 2019) and follows the generator-discriminator approach laid out in ELECTRA (Clark et al., 2020). The resulting model shows superior results in code search, natural language-programming language (NL-PL) probing, and documentation generation.

Our work differs from the previous multimodal source code encoders in the following points: 1) Our model is trained using novel pre-training objectives targeting specifically the duplicate detection task. 2) Unlike the CuBERT, explicitly designated for Python and CodeBERT, pre-trained on six different programming languages, our model is capable of processing inputs from an arbitrary programming language enabling it to be deployed to real-world question-answering platforms. 3) Our MQDD model employs a Transformer-based architecture with an attention scheme scaling linearly with sequence length allowing it to process long sequences in a reasonable time.

3 Model Pre-training

This section describes the pre-training procedure, including the construction of the new dataset from the Stack Overflow, the definition of the learning objectives, and the model itself.

3.1 Stack Overflow Dataset

For the pre-training, we construct our Stack Overflow Dataset (SOD), created from the Stack Overflow data dump[2]. The original data source[3] contain around 17,7M question. To construct the dataset, we take all question-answer pairs, extract the textual and source code parts and apply different preprocessing on both (for pre-processing details, see appendix A). A result of the pre-processing procedure are *tuples* (Q_t, Q_c, A_t, A_c) containing pre-processed texts (t) and codes (c) from both the questions (Q) and answers (A).

Afterwards, we construct the training set by taking *2-combinations* of the pre-processed *tuples*, resulting in 6 different *input pair* types described in Section 3.3. The acquired *input pairs* (x_1, x_2) are further processed in batches of 100 examples. For each pair in the batch, we sample one negative ex-

[2]Available at: https://archive.org/download/stackexchange.

[3]Data dump was downloaded in June 2020. Therefore, all the stated information is valid to this date.

Order	Tag	Percentage
1	javascript	10,95
2	java	9,88
3	c#	8,04
4	php	7,95
5	python	6,32
6	html	6,18
7	css	4,28
8	c++	4,15
9	sql	3,42
10	c	2,29
-	*total*	63,98

Table 1: The table presents a tag-based analysis of the percentage of individual programming languages in the SOD dataset. The table shows the 10 most frequent programming languages included in the dataset. Together they form ≈64% of all the examples. The remaining 36% are then made up of less popular programming languages or specific technologies.

ample by choosing a random text or code x_r from the batch buffer and use it as a replacement for the second element in the pair. This results in adding pair (x_1, x_r) to the training set.

Subsequently, we tokenize the input pairs. The resulting dataset contains 218.5M examples and can be downloaded from our GitHub repository `https://github.com/kiv-air/StackOverflowDataset`. A detailed description of the dataset's structure and dataset size is provided in appendix D and Table 4. Furthermore, Table 1 presents a detailed analysis of the programming languages included in the corpus.

3.2 Tokenization

Before extracting the input pairs, we employ the (Q_t, Q_c, A_t, A_c) tuples to train a joint tokenizer for both the source codes and English texts. We use the *Word Piece* tokenizer (Schuster and Nakajima, 2012), whose vocabulary size is typically set to a value between 10K-100K subword tokens. In our work, we set the vocabulary size to 50K subword tokens, which is large enough to encompass both the textual and code tokens while preserving a reasonable size of the embedding layer. When constructing the dataset, we ignore all tokens that occur less than five times in the dataset.

3.3 Pre-training Objectives

Similarly to BERT (Devlin et al., 2018), we employ a *Masked Language Modeling (MLM)* task during the pre-training phase. The *MLM* objective aims to reconstruct original tokens from intentionally modified input sequences. The modification replaces randomly selected tokens with a special `[MASK]` token or any other token from the dictionary.

Besides the *MLM*, we introduce two Stack Overflow dataset-specific tasks dealing with multimodal data. The first task is called *Question-Answer (QA)*, and it aims to classify whether the *input pair* originates from a question-answer relationship. The individual elements of the *input pair* can be either a natural language text or a programming language snippet. Therefore, we work with the following *input pair* types:

- Question text - Answer code *(Qt-Ac)*
- Question code - Answer code *(Qc-Ac)*
- Question text - Answer text *(Qt-At)*
- Question code - Answer text *(Qc-At)*

The second Stack Overflow-related task is called *Same Post (SP)*. Similarly to the *QA* task, the *SP* works with *input pairs* of natural language and source code snippets. However, unlike the *QA* task, *SP* classifies whether the elements of the *input pair* come from the same post (a post represents either a question or an answer). The resulting possible *input pair* types are the following:

- Answer Text - Answer Code *(At-Ac)*
- Question Text - Question Code *(Qt-Qc)*

We designed these learning objectives specifically to achieve the best possible result on our target task - *duplicate detection* (Section 4). We presume that employing these tasks requiring a deep understanding of the multimodal input helps us outperform similar models such as CodeBERT (Feng et al., 2020). Furthermore, our learning objectives require comparing and matching the semantics of both the textual input and the source code, which can be leveraged on downstream tasks such as *code search* (Heyman and Cutsem, 2020; Sachdev et al., 2018; Arwan et al., 2015).

3.4 Model Description

We choose to employ the architecture of the *Longformer* model (Beltagy et al., 2020) for its attention mechanism that scales linearly with the input sequence length. This addresses the fact that the processed input sequences (mainly the source code) may contain several hundreds of tokens. Processing such long sequences with the vanilla attention

mechanism used in the Transformer (Vaswani et al., 2017) can be computationally exhausting.

We use the *Hugging Face's Transformers* (Wolf et al., 2020) model with approximately 146M parameters (for more details on the model, see appendix B).

On top of the base model, we build two different classification heads. The first head, dealing with the *MLM* task, takes the input tokens' contextual embeddings as its input. It means the *MLM head* works with the matrix $\mathbf{E} \in \mathbb{R}^{N \times H}$, where H is the hidden size and N is the length of the input sequence. *MLM* prediction is obtained by passing the matrix through a *linear layer* so that $\mathbf{MLM_{output}} = E \times W_{mlm}$, where $W_{mlm} \in \mathbb{R}^{H \times |V|}$, and $|V|$ represents the size of the vocabulary. In other words, the model produces a probability distribution over the vocabulary for each of the input tokens, including the masked ones. To optimize the weights, we further calculate a cross-entropy loss over the network's prediction.

The second head classifies whether an input pair represents a *question-answer pair* and whether both inputs originate from the *same post*. To achieve this, the head takes the contextual embedding of the special `[CLS]` token ($\texttt{[CLS]} \in \mathbb{R}^H$)[4]. The vector is then transformed using a *linear layer* with *ReLu* (Nair and Hinton, 2010) used as an activation function - $\mathbf{QA\_SP_{intermediate}} = relu(\texttt{[CLS]} \times W_{qa\_sp_1})$, where $W_{qa\_sp_1} \in \mathbb{R}^{H \times D}$ and D represents a dimensionality of the intermediate layer. In the end, the *Question-Answer/Same Paragraph* (QA/SP) head output is obtained using another *linear layer* - $\mathbf{QA\_SP_{output}} = QA\_SP_{intermediate} \times W_{qa\_sp_2}$, where $W_{qa\_sp_2} \in \mathbb{R}^{D \times 2}$. Put differently, the *QA/SP* head is a multi-label classifier with two output neurons. The first one represents a probability of the input pair originating from the same post. The second one represents the probability of the input pair originating from the *question-answer* relationship. To optimize the weights with respect to our *QA/SP* objectives, we compute a binary cross-entropy loss over the two output neurons.

3.5 Pre-training Procedure

We optimize our model using Adam optimizer (Kingma and Ba, 2014) with a *learning rate* of $1e{-}5$ while employing both *linear warmup* and *linear decay* to zero. The *linear warmup* is configured to reach the target *learning rate* in 45K batches. The pre-training is carried out on two Nvidia A100 GPUs and two AMD EPYC 7662 CPU cores with a batch size of 64 examples.

We perform a single iteration over the whole dataset ($\approx 220M$ examples) with such a configuration while trimming the sequences to a *sequence length* of 256 tokens. Afterward, we set the *sequence length* to 1024 tokens and train the model on additional 10M examples, enabling us to train positional embeddings for longer sequences.

4 Duplicate Question Detection

Following the pre-training phase, this section focuses on applying the obtained model to the task of duplicate detection. In the first part, we describe the construction of a new dataset for duplicate detection. The next part presents how we integrate the pre-trained model into a two-tower neural network. At the end of this section, we describe the concluded experiments and present the results.

4.1 Stack Overflow Duplicity Dataset

Similarly to the pre-training phase, we employ the Stack Overflow data dump to assemble the Stack Overflow Duplicity Dataset (SODD). The data contain approximately 491K pairs of questions marked to be duplicated by the page's users. To replenish the dataset with negative samples, we employ randomly chosen questions and similar questions retrieved using ElasticSearch[5]. More specifically, we sample three random questions and retrieve six similar questions for each duplicate pair. The similarities are retrieved using the ElasticSearch either based on a full-text similarity of the question's body or associated tags. However, each question can be included in the dataset at most once. The resulting dataset consists of approximately 1.4M examples represented by triplets (x_1, x_2, y), where x_1 and x_2 represent the questions and $y \in \{$*duplicate, text_similar, tag_similar, different*$\}$ represents the label. Although the dataset differentiates between different and similar questions, all of our experiments treat the similar question pairs as different (non-duplicate). In other words, our experiments perform a binary classification into *duplicate, not duplicate* classes. For more information about the dataset size, see Table 5.

[4]The [CLS] token is an artificial token added at the begging for sequence classification tasks.

[5]https://www.elastic.co

The question pairs acquired from the Stack Overflow are stored in the *HTML* format. Therefore, we employ a `BeautifulSoup`[6] library to remove unwanted *HTML* markup and extract normal text and source code snippets. Besides, we pre-process the source code stripping all inline comments and newline characters. Similarly to the source codes, we replace numbers and date/-time information with placeholder tokens and remove newlines and punctuation in the textual part of the dataset. The resulting dataset can be obtained from our repository https://github.com/kiv-air/StackOverflowDataset. For a detailed description of the dataset structure, see appendix E.

4.2 Model

We employ a variant of a two-tower neural network to adapt our pre-trained model to the duplicate detection task. Our setup (Figure 2) encodes both questions separately using the same pre-trained encoder, obtaining representations of the questions ($x_{e1}, x_{e2} \in \mathbb{R}^d$). The representations are then concatenated ($x_e = [x_{e1}; x_{e2}]$) and transformed using a linear layer with ReLu activation (Nair and Hinton, 2010), as stated in equation 1.

$$x_L = max(0, x_e W_L + b_L) \quad (1)$$

$$x_H = softmax(x_L W_H + b_H) \quad (2)$$

At the top of our duplicate detection model, there is a classification head consisting of a linear layer with two neurons, whose activation is further transformed using a softmax function, (Bridle, 1990) as shown in equation 2.

An alternative approach would be to jointly pass both questions into the encoder and build a classification head at the top. However, our architecture of the two-tower model allows the representations of the whole corpus to be pre-computed and indexed in a fast vector space search library such as Faiss[7] (Johnson et al., 2019) (see the future work in Section 7). Thanks to that, it is possible to compute only the representation of the newly posted question and run a quick search inside the vector space. This is much faster than running the model for each pair of questions composed of a new question and the others in the corpus.

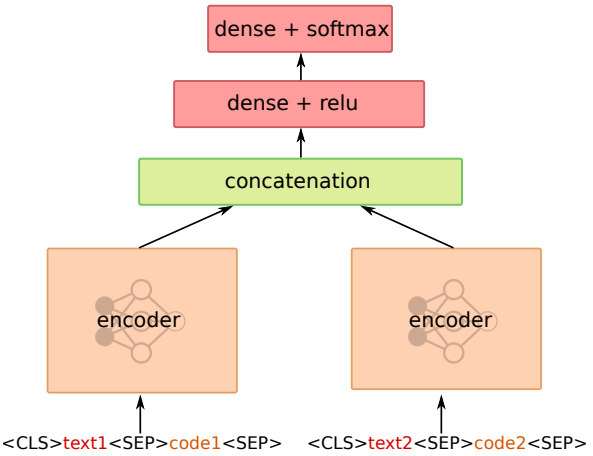

Figure 2: The neural network model architecture used for duplicate question detection. The encoder blocks in the figure share the same weights and represent either an MQDD, CodeBERT (Feng et al., 2020), or RoBERTa (Liu et al., 2019).

4.3 Experimental Setup - Duplicate Detection

Similarly to the pre-training phase, we use the Adam optimizer (Kingma and Ba, 2014) with a learning rate set to 6.35e−6 to train the model on a computation node with two cores of AMD EPYC 7662 CPU and two Nvidia A100 GPUs. In each experiment, we train the model for 24 hours with a batch size of 96 examples and observe the progress of cross-entropy loss, accuracy, and F1 score. The hyperparameters were set based on 30 hyperparameter-search experiments conducted using the *Weights & Biases* (Biewald, 2020) *sweeps* service[8]. For detailed information about the hyperparameter setting, refer to appendix C.

To evaluate the effectiveness of our pre-training objectives, we compare our model with the *CodeBERT* (Feng et al., 2020), RoBERTa (Liu et al., 2019), and randomly initialized Longformer (Beltagy et al., 2020). The comparison experiments also utilize the architecture depicted in Figure 2, where we only replace the encoder with the model being compared. The training setup for the comparison experiment is identical to the setup described above. It means that we fine-tune the models for 24 hours on the same hardware.

[6] https://beautiful-soup-4.readthedocs.io/en/latest/

[7] https://github.com/facebookresearch/faiss

[8] https://docs.wandb.ai/guides/sweeps

4.4 Results

As evaluation metrics, we use an *F1 score* and *accuracy*. We summarize the results of our experiments in Table 2, where the achieved results are stated with 95% confidence intervals computed over 10 runs. From the results, we can see that our model significantly outperformed all alternative approaches. For further discussion on the results, see Section 6.

Model	Accuracy	F1 Score
MQDD	**74.83 ± 0.10**	**75.10 ± 0.10**
CodeBERT	70.44 ± 0.12	70.70 ± 0.13
RoBERTa	70.16 ± 0.19	70.51 ± 0.22
Longformer†	67.31 ± 0.12	67.71 ± 0.19

Table 2: Summary of duplicate detection experiment results stated with 95% confidence intervals computed over 10 runs. The † sign marks randomly initialized models. For a discussion of the results, see Section 6.

5 Generalization to Other Tasks

To explore how well our model generalizes to other tasks, we choose the **code search** task. The information retrieval seems to be close to our pre-training tasks. For all the experiments, we use the *CodeSearchNet* dataset (Husain et al., 2019) containing approximately 2.3M examples from six different programming languages extracted from *GitHub* repositories.

5.1 Domain-Specific Pre-Training

Since our model is pre-trained on Stack Overflow data significantly different from the *CodeSearchNet* extracted from *GitHub*, we employ a domain-specific pre-training to adapt our model to the target domain.

We employ the *masked language modeling* (MLM) learning objective for the domain-specific pre-training. We perform 20 iterations over the *CodeSearchNet* dataset following the same experimental setup as described in Section 3.5.

5.2 Experimental Setup – Code Search

To fine-tune our model on the *CodeSearchNet* dataset (Husain et al., 2019), we utilize its pre-processed version from the authors of CodeBERT (Feng et al., 2020) since it comes with negative examples, unlike the original dataset distribution. In our experiments, we train a separate model for each of the six available programming languages and compare our results with the results obtained using the CodeBERT (Feng et al., 2020), RoBERTa (Liu et al., 2019), and randomly initialized Longformer (Beltagy et al., 2020).

For all of the experiments, we employ the `AutoModelForSequenceClassification` class from the *Hugging Face's Transformers* (Wolf et al., 2020) library as it comes with an in-build classification head that operates over the pooled output of the base model.

Similarly to the duplicate detection experiments, we perform the fine-tuning on two NVidia A100 GPUs for 24 hours with a batch size of 64 examples. For optimization, we also employ the Adam (Kingma and Ba, 2014) optimizer with a *learning rate* of $1e{-}5$. Furthermore, we utilize *learning rate warmup* during the first 256 batches and apply *linear learning rate decay* to zero.

5.3 Results

In the case of the code search task, we use the F1 score metric. The complete summary of the results with 95% confidence intervals computed over 10 runs can be found in Table 3. The results show that both the *CodeBERT* (Feng et al., 2020) and *RoBERTa* (Liu et al., 2019) significantly outperform our model in the code search task.

6 Discussion

As the results stated in Sections 4.4 and 5.3 suggest, our model excels in detecting duplicates but lags in source code retrieval. We expected the dominance of our model in the duplication detection task. However, an interesting observation is that the pre-training of the CodeBERT, whose author's (Feng et al., 2020) initialized it using the RoBERTa's (Liu et al., 2019) weights, does not bring any improvement when applied to the duplicate detection. On the other hand, it is surprising that our MQDD model does not perform comparably well as the CodeBERT on the code search as our pre-training objectives require the model to build a deep understanding of the processed source code.

This can be explained by the fact that the datasets used for pre-training of both models have very different characteristics. The SOD does not contain source code from a constrained set of six programming languages (see Table 1), as in the case of the CodeBERT. Therefore, our model may produce representations of all programming languages in average quality. In contrast, the CodeBERT

Model	Go	Java	JavaScript	PHP	Python	Ruby
MQDD	95.33 ± 0.04	80.11 ± 0.15	70.09 ± 0.48	85.58 ± 0.16	84.14 ± 0.48	82.77 ± 0.31
CodeBERT	**96.68 ± 0.06**	**83.75 ± 0.06**	**83.42 ± 0.06**	**88.50 ± 0.03**	**88.25 ± 0.12**	**87.22 ± 0.31**
RoBERTa	95.94 ± 0.06	81.58 ± 0.23	80.35 ± 0.25	86.78 ± 0.09	86.02 ± 0.11	84.06 ± 0.20
Longformer†	66.62 ± 0.14	66.51 ± 0.24	66.71 ± 0.15	66.68 ± 0.06	66.71 ± 0.10	66.74 ± 0.15

Table 3: Results summary of *code search* experiments in six different programming languages. The F1 score is stated in percents with 95% confidence intervals computed over 10 runs. The best results in each language are highlighted in bold. The † sign marks randomly initialized models. For an analysis of the results see Section 6.

may produce high-quality representations in the six programming languages it was pre-trained on, but lower than average representations of the other programming languages. This would also explain why CodeBERT does not perform so well on duplicates; it excels in processing the six programming languages but fails to generalize to other abundantly contained languages in the Stack Overflow dataset.

However, the offered explanation does not cover that RoBERTa, whose pre-training dataset did not contain any source code, outperforms our model in the code search task. We speculate that this can be caused by the MQDD model being trapped in its local optimum due to its pre-training designed especially for the duplicate detection. This can make it difficult to get out of this local optimum when fine-tuned on a slightly different dataset and task. This phenomenon is often referred to as a *negative transfer* (Rosenstein et al., 2005; Zhang et al., 2020) and can be caused, among other things, by the discrepancy between the pre-training and fine-tuning domains.

Given that our research aimed to build a model designed directly for the detection of duplicates on platforms such as Stack Overflow, it can be stated that the results we achieve are satisfactory. Our model far exceeds the results achieved by competitive work on a task that can be perceived as more demanding due to the need to process a general source language and distinguish seemingly insignificant semantic nuances. For example, questions *"How to implement a producer-consumer in Java"* and *"How to implement a producer-consumer in C++"* must be identified as different since the answers would significantly differ.

7 Future Work

Our work opens up further opportunities to build on our current research. First of all, it would be interesting to explore methods that would eliminate the effect of negative transfer and thus allow the use of our pre-trained model in other tasks.

Furthermore, the follow-up work can integrate our model into a production-ready duplicate detection system employing a fast vector space search library such as *Faiss*.

The proposed system can be further extended by a duplicate detection model that jointly processes both questions allowing the attention mechanism to attend across both inputs. Such a model can potentially achieve better results and be deployed along with our two-tower-based model. Our two-tower model would then be used to filter out candidate duplicate questions. Afterward, the cross-attention model could verify that the candidate questions are indeed duplicates more accurately.

8 Conclusion

This work presents a new pre-trained BERT-like model that detects duplicate posts on programming-related discussion platforms. Based on the Longformer architecture, the presented model is pre-trained on our novel pre-training objectives (*QA* and *SP*) that aim to target the duplicate detection task. The comparison with the competitive CodeBERT model shows that our model outperforms other approaches, suggesting the effectiveness of our learning objectives. Furthermore, we investigated the generalization capabilities of our model by applying it to a code retrieval task. In this task, it turned out that our model does not exceed the results achieved with either CodeBERT or the more general RoBERTa model. We attribute these findings to the significant differences between our pre-training dataset and the evaluation dataset for the code search task. Therefore, we consider our model an excellent choice for solving duplicate detection. However, it seems to be too specialized to solve other tasks well.

Our models are publicly available for research purposes in our Hugging Face[9] and GitHub[10] repositories.

[9] https://huggingface.co/UWB-AIR
[10] https://github.com/kiv-air/MQDD

Acknowledgments

This work has been supported by Grant No. SGS-2022-016 Advanced methods of data processing and analysis. Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic.

References

Achmad Arwan, Siti Rochimah, and Rizky Januar Akbar. 2015. Source code retrieval on stackoverflow using lda. In *2015 3rd International Conference on Information and Communication Technology (ICoICT)*, pages 295–299.

Dzmitry Bahdanau, Kyunghyun Cho, and Y. Bengio. 2014. Neural machine translation by jointly learning to align and translate. *ArXiv*, 1409.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *CoRR*, abs/2004.05150.

Lukas Biewald. 2020. Experiment tracking with weights and biases. Software available from wandb.com.

John S. Bridle. 1990. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236, Berlin, Heidelberg. Springer Berlin Heidelberg.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *CoRR*, abs/2005.14165.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Zimin Chen and Martin Monperrus. 2019. A literature study of embeddings on source code. *CoRR*, abs/1904.03061.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: pre-training text encoders as discriminators rather than generators. *CoRR*, abs/2003.10555.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. Codebert: A pre-trained model for programming and natural languages. *CoRR*, abs/2002.08155.

Geert Heyman and Tom Van Cutsem. 2020. Neural code search revisited: Enhancing code snippet retrieval through natural language intent. *CoRR*, abs/2008.12193.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. *CoRR*, abs/1909.09436.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. 2020a. Pre-trained contextual embedding of source code. *CoRR*, abs/2001.00059.

Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. 2020b. Pre-trained contextual embedding of source code. *CoRR*, abs/2001.00059.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Triet Huynh Minh Le, Hao Chen, and Muhammad Ali Babar. 2020. Deep learning for source code modeling and generation: Models, applications and challenges. *CoRR*, abs/2002.05442.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Zihan Liu, Feijun Jiang, Yuxiang Hu, Chen Shi, and Pascale Fung. 2021. NER-BERT: A pre-trained model for low-resource entity tagging. *CoRR*, abs/2112.00405.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. *CoRR*, abs/1708.00107.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA. Omnipress.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084.

Michael Rosenstein, Zvika Marx, Leslie Kaelbling, and Thomas Dietterich. 2005. To transfer or not to transfer.

Saksham Sachdev, Hongyu Li, Sifei Luan, Seohyun Kim, Koushik Sen, and Satish Chandra. 2018. Retrieval on source code: A neural code search. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2018, page 31–41, New York, NY, USA. Association for Computing Machinery.

M. Schuster and K.K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152.

Chi Sun, Luyao Huang, and Xipeng Qiu. 2019. Utilizing BERT for aspect-based sentiment analysis via constructing auxiliary sentence. *CoRR*, abs/1903.09588.

Weisong Sun, Chunrong Fang, Yuchen Chen, Guanhong Tao, Tingxu Han, and Quanjun Zhang. 2022. Code search based on context-aware code translation. *arXiv preprint arXiv:2202.08029*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Liting Wang, Li Zhang, and Jing Jiang. 2020. Duplicate question detection with deep learning in stack overflow. *IEEE Access*, 8:25964–25975.

Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *CoRR*, abs/2109.00859.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Wen Zhang, Lingfei Deng, and Dongrui Wu. 2020. A survey on negative transfer. *CoRR*, abs/2009.00909.

A Dataset Pre-processing

The data retrieved from the Stack Overflow data dump contain an HTML markup that needs to be pre-processed before being used to train a neural network. Furthermore, the natural language and source code snippets are mixed in a single HTML document, so we need to separate those two parts.

We use the `BeautifulSoup`[11] library to extract the textual data from the HTML markup. To do so, we remove all content enclosed in `<code></code>` tags and strip all the remaining HTML tags. Afterward, we remove all newline characters and multiple subsequent space characters induced by stripping the HTML tags.

On the other hand, while pre-processing the code snippets, we first extract all content from `<pre><code></code></pre>` using the `BeautifulSoup` library and throw away the rest. Afterward, we remove the newlines and multiple spaces, as in the case of the textual part.

B Longformer Model Configuration

The implementation of the Longformer model that we employ in the pre-training is the `transformers.LongformerModel`[12] from *HuggingFace Transformers* library. Below, we provide a detailed listing of the model's parameters.

- attention_probs_dropout_prob = 0.1
- attention_window = 256
- hidden_act = gelu
- hidden_dropout_prob = 0.1
- hidden_size = 768
- initializer_range = 0.02
- intermediate_size = 3072
- layer_norm_eps = 1e-12
- max_position_embeddings = 1026
- num_attention_heads = 12
- num_hidden_layers = 12
- position_embedding_type = absolute
- vocab_size = 50256
- intermediate_layer_dim (D) = 1000

[11] https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[12] https://huggingface.co/docs/transformers/model_doc/longformer#transformers.LongformerModel

C Duplicate Detection Hyperparameters

For fine-tuning our MQDD model on the duplicate detection task, we employ the Adam optimizer with an initial learning rate of $6.35\mathrm{e}{-6}$. We train the model on sequences of 256 subword tokens with a batch size of 100 examples. Additionally, we use an L2 normalization with a normalization factor set to 0.043. Another regularization method we employ is the dropout with the following configuration:

- attention dropout in the Longformer = 0.2
- hidden dropout in the Longformer = 0.5
- dropout at the first linear layer of the classification head = 0.26
- dropout at the second linear layer of the classification head = 0.2

D Stack Overflow Dataset Structure

The *Stack Overflow Dataset* (SOD) consists of a metadata file and several data files. Each line of the metadata file (`dataset_meta.csv`) contains a *JSON* array with the following information:

- **question_id** - identifier of the question in format `<id>-<page>` (in our case the `page = stackoverflow`)
- **answer_id** - identifier of the answer in format `<id>-<page>` (in our case the `page = stackoverflow`)
- **title** - title of the question
- **tags** - tags associated with the question
- **is_accepted** - boolean flag indicating whether the answer represents an accepted answer for the question

The dataset export is organized in such a way that i-th row in the metadata file corresponds to training examples located on the i-th row in the data files. There are six different data file types, each comprising training examples of different *input pair types* (described in Section 3.3). A complete list of the data file types follows:

- `dataset_AC_AT.csv` - code from an answer with text from the same answer
- `dataset_QC_AC.csv` - code from a question with code from a related answer
- `dataset_QC_AT.csv` - code from a question with text from a related answer

- `dataset_QC_QT.csv` - code from a question with text from the same question
- `dataset_QT_AC.csv` - text from a question with code from a related answer
- `dataset_QT_AT.csv` - text from a question with text from a related answer

Each row in the data file then represents a single example whose metadata can be obtained from a corresponding row in the metadata file. A training example is represented by a *JSON* array containing two strings. For example, in the `dataset_QC_AC.csv`, the first element in the array contains code from a question, whereas the second element contains code from the related answer. It shall be noted that the dataset export does not contain negative examples since they would significantly increase the disk space required for storing the dataset. The negative examples must be randomly sampled during pre-processing, as discussed in Section 3.1.

Since the resulting dataset takes up a lot of disk space, we split the individual data files and the metadata file into nine smaller ones. Therefore, files such as, for example, `dataset_meta_1.csv` and corresponding `dataset_QC_AT_1.csv` can then be found in the repository.

Statistic	QC	QT	AC	AT	Total
avg. # of characters	846	519	396	369	-
avg. # of tokens	298	130	140	92	-
avg. # of words	83	89	44	60	-
# of characters	16.1B	13.5B	6.6B	9.6B	45.8B
# of tokens	5.7B	3.4B	2.3B	2.4B	13.8B
# of words	1.6B	2.3B	0.7B	1.6B	6.2B

Table 4: Detailed statistics of the released Stack Overflow Dataset (SOD). The table shows the average number of characters, tokens, and words in different source codes present in questions (QC) or answers (AC) and texts present in questions (QT) or answers (AT). Besides the average statistics, the table provides a total count of tokens, words, or characters. To calculate the statistics related to token counts, we utilized the tokenizer presented in Section 3.2, whereas we employed a simple space tokenization for the word statistics.

E Stack Overflow Duplicity Dataset Structure

The published *SODD* dataset is split into train/dev/test splits and is stored in *parquet*[13] files compressed using gzip. The data can be loaded using the *pandas*[14] library using the following code snippet:

```
!pip3 install pandas pyarrow

import pandas as pd

d=pd.read_parquet('<file>.parquet.gzip')
```

The dataframe loaded using the snippet above contains the following columns:

- **first_post** - HTML formatted data of the first question (contains both text and code snippets)
- **second_post** - HTML formatted data of the second question (contains both text and code snippets)
- **first_author** - username of the first question's author
- **second_author** - username of the second question's author
- **label** - label determining the relationship of the two questions
  0. duplicates
  1. similar based on full-text search
  2. similar based on tags
  3. different
  4. accepted answer
- **page** - Stack Exchange page from which the questions originate (always set to `stackoverflow`)

As one can see, our dataset contains accepted answers as well. Although we are not using them in our work, we included them in the dataset to open up other possibilities of using our dataset.

For detailed information about the size of our SODD dataset, see table 5.

Type	Train	Dev	Test	Total
Different	550K	64K	32K	646K
Similar	526K	62K	30K	618K
Duplicates	191K	22K	11K	224K
Total	1.2M	148K	73K	1.4M

Table 5: Stack Overflow Duplicity Dataset (SODD) size summary.

[13] https://parquet.apache.org/documentation/latest/

[14] https://pandas.pydata.org