The NLBSE'24 Tool Competition

Rafael Kallis
Rafael Kallis Consulting
Zurich, Switzerland
rk@rafaelkallis.com

Giuseppe Colavito
University of Bari
Bari, Italy
giuseppe.colavito@uniba.it

Ali Al-Kaswan
Delft University of Technology
Delft, The Netherlands
a.al-kaswan@tudelft.nl

Luca Pascarella
ETH Zurich
Zurich, Switzerland
lpascarella@ethz.ch

Oscar Chaparro
College of William & Mary
Williamsburg, USA
oscarch@wm.edu

Pooja Rani
University of Zurich
Zurich, Switzerland
rani@ifi.uzh.ch

ABSTRACT

We report on the organization and results of the tool competition of the third International Workshop on Natural Language-based Software Engineering (NLBSE'24). As in prior editions, we organized the competition on automated issue report classification, with focus on small repositories, and on automated code comment classification, with a larger dataset. In this tool competition edition, six teams submitted multiple classification models to automatically classify issue reports and code comments. The submitted models were fine-tuned and evaluated on a benchmark dataset of 3 thousand issue reports or 82 thousand code comments, respectively. This paper reports details of the competition, including the rules, the teams and contestant models, and the ranking of models based on their average classification performance across issue report and code comment types.

KEYWORDS

Tool-Competition, Labeling, Benchmark, Issue Reports, Code Comments

ACM Reference Format:
Rafael Kallis, Giuseppe Colavito, Ali Al-Kaswan, Luca Pascarella, Oscar Chaparro, and Pooja Rani. 2024. The NLBSE'24 Tool Competition. In *Proceedings of 46th International Conference on Software Engineering (ICSE 2024).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The first competition was held in 2022 [13, 28], the second the year after [32, 38] and this one continues the series with the third edition of the Natural Language-based Software Engineering (NLBSE'24) tool competition on automated issue report and code comment classification. Both competitions aimed to bring practitioners and researchers together into developing more accurate classification models for automatically identifying the type of given issue report or code comment.

We focused on issue report classification for two reasons: (i) it is an important task for developers in the context of the issue management and prioritization process [37], and (ii) extensive research has been dedicated to addressing this problem using natural language processing (NLP) and machine learning (ML) techniques [22, 26]. Similarly, several works have shown the importance of source code comments in software development and maintenance [1, 11, 40]. For example, well-written code comments actively enhance code readability by documenting code changes. Nonetheless, not all code comments fit this role. Indeed, code comments are used to accomplish different tasks, such as code documentation, license declaration, report work in progress, *etc.* In other words, code comments contain various kinds of information that can support developers in different program comprehension and maintenance tasks [44]. To satisfy different needs, the information is written using a mix of code and natural language sentences; consequently, researchers have leveraged various NLP and ML-based techniques to identify the types of information in these sentences.

Six teams [3, 4, 17, 20, 21, 47] participated in the two competitions. Each team proposed classification models trained and evaluated on one of the two datasets we provided [29]. The first dataset contains 3 thousand issue reports extracted from 5 repositories of open-source projects and each issue is labeled with one type. This dataset has no overlap with previous competitions. The second dataset is a subset of dataset from Rani et al. [45] and Pascarella and Bacchelli [40] that contains the ground truth categories, *i.e.,* information types, of 14,875 comment sentences from 1,733 class comments of 20 projects written in three programming languages: Java, Pharo, and Python [45].

The baseline models provided for issue classification were based on three approaches: SetFit [46], RoBERTa [35] and FastText [27]. FastText is used by Kallis et al.'s Ticket Tagger [30, 31], RoBERTa is used by Izadi's CatIss [24, 26], and SetFit is used by Colavito et al. [10]. The baseline solution provided for code comment classification, named STACC [2], was based on a set of Sentence Transformers.

Given these datasets and the classification results of the baseline models, the participants of this year's competitions were expected to design their classifiers to outperform the baselines in identifying the correct type(s) of issue reports or code comments.

ICSE 2024, April 2024, Lisbon, Portugal

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

2 ISSUE REPORT CLASSIFICATION

In this section, we report the structure and measures for the tool competition on issue report classification. The competition followed a similar structure to the previous editions [28, 32]. We received feedback from last edition's participants [10, 33] concerning the dataset that was used.

The main criticism was the presence of noisy labels caused by the variability in labeling rationale among different projects, which may lead to inaccurate results [53]. We also received concerns regarding the cost of fine-tuning state-of-the-art models, as it is not inclusive to those without access to the necessary infrastructure. An additional concern is the limited applicability of repository-specific fine-tuning methods to smaller repositories, as most repositories only have few issue reports.

We also introduced an additional SetFit [46] baseline model based on last edition's winner by Colavito et al. [10].

The remainder of this section is organized as follows. We first describe the dataset, then list the competition rules, then summarize this year's submissions, and finally, we present the evaluation and results of the submissions. We published a GitHub repository[1] to guide and inform potential participants about the competition.

2.1 Benchmark Dataset

We provided a dataset with three thousand issue reports extracted from five popular open-source projects hosted on GitHub: "React", "Tensorflow", "Visual Studio Code", "Bitcoin", and "OpenCV" [2]. Sampling issue reports across only five reference repositories should make the dataset's labeling rationale more consistent compared to last year when we sampled across all public repositories. The reduced dataset size aims to be more inclusive to those without access to expensive infrastructure.

The issues were extracted using the GitHub API[3]. We extracted the most recently closed issues at the time, *i.e.,* October 5th 2023, that contained any of the labels *bug*, *feature*, or *question*. These are the most frequently used labels on GitHub [6, 26]. We chose a balanced label distribution in order to have a comparable dataset across the five repositories.

We extracted the following data attributes for each issue: the issue *title* or summary, and the issue *body* or description. Additionally, each issue is labeled with one class that indicates its type, namely, *bug*, *feature*, or *question*. To further reduce possible inconsistencies in the labeling rationale, we exclude issue reports with multiple labels. The dataset was given in CSV format without applying any further pre-processing on the issues.

We partitioned the dataset into a training set and a test set using a 50/50 split. We therefore have one hundred issue reports per label and repository, for each of the training and test set ($100 \times 3 \times 5 \times 2 =$ 3,000 issue reports). We published a Jupyter notebook that performs the above steps in our tool competition's repository on GitHub.

[1] Issue report classificatio repository: github.com/nlbse2024/issue-report-classification
[2] Reference repositories of competition dataset: github.com/facebook/react, tensorflow/tensorflow, microsoft/vscode, bitcoin/bitcoin, opencv/opencv
[3] GitHub API: docs.github.com/en/rest

2.2 Baselines and Competition Rules

We published three classification models as competition baselines. The first baseline uses SetFit, a framework for few-shot fine-tuning of Sentence Transformers [46], used by Colavito et al.'s tool [10] that was submitted in the previous edition of the competition. Our second baseline uses RoBERTa [35], the backbone Transformer in the CatIss tool by Izadi [24, 26], submitted in the first edition of the competition. The third baseline uses FastText [27], a static word embedding used in the tool Ticket Tagger by Kallis et al. [30–32].

The participants had to train and tune their five classification models using the training set and evaluate the models using the test set. The test set was used to determine the official classification results and the ranking of the contestant models.

The participants were free to select and transform any variables from the training set. Pre-trained models were permitted, but can only be fine-tuned on the training set. Any inputs or features used to create or fine-tune the classifier, had to be derived from the provided training set. Participants were allowed to pre-process, sample, apply over/under-sampling, select a subset of the attributes, perform feature engineering, filter records, split the training set into a model-tuning validation set. The participants were free to apply any pre-processing or feature engineering on the test set except sampling, rebalancing, undersampling or oversampling techniques.

The proposed models were evaluated based on their classification performance on the test set. The classifiers had to assign a single label to an issue: *bug*, *feature*, or *question*. The classification performance of a model is measured by the weighted-average F_1-score over all three classes. While the F_1-score was used for ranking the models and determining the winner of the competition, we also asked the participants to report the following metrics: Precision and Recall for each class [25].

The competition's GitHub repository contained specific instructions and rules, including replication package and results of the baseline models based on SetFit, RoBERTa and FastText. More importantly, the repository contained notebooks aimed to facilitate participation in the competition as they were ready to be adapted, used, and executed.

2.3 Submitted Classification Models

Five teams submitted a classifier to participate in the competition. As listed in Table 1, all of the participants were able to outperform the SetFit baseline based on the competition's assessment metric, *i.e.,* the arithmetic mean of the F_1-score over all repos. We provide an overview of the accepted approaches.

Ebrahim and Joy [17] proposed a RoBERTa-based classifier with adapters [23, 41, 42]. Their approach consisted of fine-tuning some extra layers on top of the RoBERTa [35] model and then training a classification head. An adapter was trained for each repository. The authors used regular expressions to preprocess the issue text. They removed new lines, multiple tabs, and spaces, links, error traces, text between triple quotes, and special characters except for question marks. Their approach achieved an average F_1-score of 0.893.

Gómez-Barrera et al. [20] proposed a classifier based on CFFitST, a framework inspired by SetFit [52]. Their approach is based on Sentence Transformers embeddings and cosine similarity distance.

Table 1: Issue classification results over the three issue types, averaged across the five repositories, sorted by average F_1-score.

Classification Model	Metric	Bug	Feature	Question	Average
RoBERTa Adapters *Ebrahim and Joy* [17]	Precision	0.897	0.897	0.889	0.895
	Recall	0.890	0.902	0.888	0.893
	F_1-score	0.892	0.899	0.888	**0.893**
CFFitST *Gómez-Barrera et al.* [20]	Precision	0.849	0.853	0.831	0.845
	Recall	0.838	0.906	0.764	0.843
	F_1-score	0.853	0.877	0.789	**0.842**
RoBERTa *Alam et al.* [3]	Precision	0.814	0.853	0.830	0.832
	Recall	0.860	0.849	0.787	0.832
	F_1-score	0.836	0.851	0.808	**0.832** *
Flan-T5 *Rejithkumar et al.* [47]	Precision	0.809	0.858	0.825	0.831
	Recall	0.886	0.858	0.748	0.784
	F_1-score	0.846	0.858	0.784	**0.829** *
GPT-3.5 Turbo *Aracena et al.* [4]	Precision	0.810	0.868	0.818	0.832
	Recall	0.858	0.854	0.774	0.828
	F_1-score	0.832	0.859	0.732	**0.828**
SetFit *Kallis and Colavito* [10, 29]	Precision	0.846	0.844	0.800	0.830
	Recall	0.840	0.870	0.770	0.880
	F_1-score	0.842	0.855	0.782	**0.827**
RoBERTa *Kallis and Izadi* [24, 29]	Precision	0.808	0.812	0.773	0.798
	Recall	0.806	0.808	0.764	0.792
	F_1-score	0.805	0.805	0.765	**0.792**
FastText *Kallis et al.* [29–31]	Precision	0.765	0.715	0.685	0.722
	Recall	0.706	0.770	0.682	0.719
	F_1-score	0.732	0.739	0.683	**0.718**

Submissions marked with * have trained a single model over all repositories and will perform slightly worse than indicated.

They generate training pairs which can be divided in positive and negative pairs for each class. Positive pairs are composed by two samples belonging to the same class, while negative pairs consist of two samples belonging to different classes. The model is trained to maximize the similarity between positive pairs and minimize the similarity between negative pairs. The proposed approach consists in refining a model by training it multiple times. The number of examples for each class pair can have a fixed or variable size. The variable chunk size is used in order to add samples for the classes that performed better in the previous training. They experimented with removing code blocks from the text and representing the title and body with two different embeddings, but they did not observe improvements for the former, and they observed a decrease in performance for the latter. Their approach achieved an average F_1-score of 0.842.

Alam et al. [3] experimented with different BERT-like models. RoBERTa-large [35] emerged as the best performing model and was used for the tool competition submission. A single model was trained for all repositories. They preprocessed the issues by removing hyperlinks, special characters and numbers, and converting the text to lowercase. Their approach achieved an average F_1-score of 0.832.

Rejithkumar et al. [47] proposed a Flan-T5-based [9, 43] classifier. They used the VMware/flan-t5-large-alpaca model, which is further instruction fine-tuned on the Alpaca dataset [51]. A single model was trained for all repositories. The authors reported that preprocessing negatively impacted the performance of the model. For this reason, they did not apply any preprocessing to the text. Their approach achieved an average F_1-score of 0.829.

Aracena et al. [4] proposed a GPT-3.5-Turbo-based [5] classifier. They used the GPT-3.5-Turbo model, which powers ChatGPT in his default configuration. The authors experimented with two different preprocessing approaches. In the first they remove emojis, URLs, HTML tags, special characters and punctuation, and double quotation marks. The second approach is similar to the first, but URLs and HTML tags are replaced with a special token, as also done for user mentions and image links. In this case, also the markdown syntax is removed. They observed that different preprocessing approaches have different impact on the performance of the model depending on the repository. Analyzing the results, they observed that the model struggles to classify issues belonging to the *Question* class, raising concerns about the quality of the labels, as also observed by Colavito et al. [10] in the previous edition of the competition. Their approach achieved an average F_1-score of 0.828.

2.4 Classifier Evaluation and Results

Based on the replication package provided by each team, we replicated the results reported in their papers [3, 4, 17, 20, 47]. We executed the code using a workstation equipped with a RTX 3090 GPU. Training and fine-tuning of the SetFit baseline lasted 7 minutes, and GPU memory usage peaked at 17.4 GB.

In Table 1 we report the classification performance obtained by the proposed classifiers on the test set. All the proposed approaches were able to outperform the baselines. In particular, the approach based on RoBERTa with adapters proposed by Ebrahim and Joy [17] achieved the best results. The adapters approach not only performs best, but also more practical on supporting multiple repository-specific models. That is because the underlying model, RoBERTa, remains unchanged during the fine-tuning process and each repository-specific adapter model only adds around 6 MB.

It is also important to note that approaches based on GPT-3.5-Turbo [4] and Flan-T5 [3], which are much larger models than RoBERTa, achieved comparable performances to the SetFit baseline. This evidence is of crucial importance, as it shows that issue report classification can be performed with much smaller models, which are also much cheaper to train and serve. In a realistic scenario in which it is needed to classify issue reports at scale, having a smaller model that requires less computational resources can be game-changing. While the use of GPT-3.5-Turbo through API removes the need for infrastructure to deploy and serve the model, this could raise concerns about the privacy of the data, as the issue reports need to be shared with a third party to perform the classification.

The performance achieved by the CFFitST [20] framework shows that it is possible to surpass the SetFit baseline by sampling the training set and repeating the training process multiple times using Sentence Transformer embeddings. Still, the RoBERTa model, and in general standard BERT-based models, remain a viable option for issue report classification, as shown by Rejithkumar et al. [47].

Looking at the performances per class, we observe that all the approaches struggle to classify issues belonging to the *Question* class, except for the approach implemented by Ebrahim and Joy [17], which reports a well-balanced performance across all labels. Furthermore, the proposed approach performs consistently also across repositories, except for the "Tensorflow" repository, for which it achieves a slightly worse performance.

Based on the classification results, we rank the five contestant teams as follows:

a) Ebrahim and Joy [17] take the first place in the competition with their Adapter RoBERTa-based approach;
b) Gómez-Barrera et al. [20] occupy the second place with their CFFitST model;
c) Alam et al. [3] are placed third with their RoBERTa-based apprach;
d) Rejithkumar et al. [47] are ranked fourth with their T5-based apprach;
e) Aracena et al. [4] are positioned fifth with their GPT-3.5-Turbo finetuning;
f) Our baselines [29] occupy positions six, seven, and eight, with models based on SetFit [10], RoBERTa [24], and FastText [30, 31], respectively;

3 CODE COMMENT CLASSIFICATION

The code comment classification competition consisted of building and testing a set of binary classifiers to classify code comment sentences as belonging to one or more categories. These categories represent the types of information a sentence conveys in comments of code classes. Overall, the competition followed a structure similar to the previous edition [32]. However, compared to the previous edition, in this competition, we (i) have extended the dataset of code comments for Java projects and (ii) have changed the baseline. More in detail, we provided (i) a dataset of code comment sentences and (ii) baseline classifiers based on the Sentence Transformer architecture, detailed later in subsections 3.1 and 3.2, respectively. The competition called for participants that proposed classifiers with the goal of outperforming the baseline classifiers. We provided a GitHub repository[4] and a Colab notebook[5] to guide and inform potential participants about the competition.

3.1 Benchmark Dataset

The competition included a dataset composed of 14,875 manually labeled comment sentences in 19 categories. The code comments are from 1,733 unique classes and belong to 20 open-source projects written in three different programming languages: Java, Python, and Pharo. This dataset has been created as a subset of the two publicly available datasets. The first reflects the same dataset of the previous edition and is provided by Rani *et al.* [45], while the second extends the latter by adding further code comments classified by Pascarella and Bacchelli [40]. In merging the two datasets, we avoided duplicates by including a single instance per file. In other words, during the merging process, if the filename appeared twice, we included only the code comments coming from the dataset of Rani *et al.* [45].

Regarding the dataset extracted from the work of Rani *et al.* [45], It contains class comments of various open-source, popular, and heterogeneous projects that vary in terms of contributors, size, and development ecosystem. The Java projects are Apache Spark, Guava, Guice, Eclipse, Vaadin, and Apache Hadoop. The Python projects are Pandas, IPython, PyTorch, Mailpile, Request, PipeEnv, and Django. The Pharo projects are Pillar, Petit, PolyMath, Seaside, GToolkit, Roassal, and Moose. Regarding the dataset extracted from the work of Pascarella and Bacchelli [40]. It aimed to create a taxonomy of code comments in Java programming language. The taxonomy emerged from the manual analysis of 2,000 Java files, highlighting 15,000 blocks of code comments, depicts six top and 16 inner categories. The Java open-source projects coincide with those analyzed by Rani *et al.* [45] and include Apache Spark, Guava, Guice, Eclipse, Vaadin, and Apache Hadoop.

A sample of comments extracted from the aforementioned projects has been manually analyzed to identify the information that each comment sentence conveys. Based on the analysis, more than 19 types of information are found in the comment sentences across the three programming languages. For the competition, we focused on the most frequent categories, *i.e.,* with 50+ comment sentences per category, for a total of 19 code comment categories. Specifically, we

[4]Code Comment Repository: github.com/nlbse2024/code-comment-classification
[5]Colab notebook: tinyurl.com/d6m37293

selected seven Java categories: *summary*, *pointer*, *deprecation*, *rational*, *ownership*, *usage*, and *expand*; five Python categories: *summary*, *parameters*, *usage*, *development notes*, and *expand*; and seven Pharo categories: *key messages*, *intent*, *class references*, *example*, *key implementation*, *responsibilities*, and *collaborators*. The definitions of these categories can be found in the original paper by Rani *et al.* [45] and Pascarella and Bacchelli [40]. The 19 categories are found in 1,049 class-level comments for Java, 340 for Pharo, and 344 for Python, for a total of 1,733 unique class-level comments.

The applied methodology reflects the same approach of the previous edition [32]. In particular, we applied various pre-processing steps to the comments. We split the comments into sentences based on the NEON tool [15], changed the sentences to lowercase, transformed multiple line endings into one ending, and removed special characters, *e.g.,* @#&%., !?\n. These symbols were removed to ensure uniformity across languages, as they are used differently in each language. We also removed periods in numbers or special abbreviations, such as "*e.g.*", "*i.e.*", and numbers to minimize incorrect comments splitting into sentences.

Each comment sentence can belong to one or more categories, to a maximum of 5 to 7 categories, depending on the language. Each category represents the type of information that the sentence is conveying. While one sentence can belong to multiple categories, the competition focused on binary classification for each category, rather than multi-class classification. In other words, participants were meant to build multiple binary classifiers, each focusing on one category to determine if a sentence does or does not belong to such category. Therefore, for each category, we built the sets of positive and negative sentences used for binary classification, *i.e.,* belonging and not belonging to a category, based on the ground-truth categories of the 14,875 unique comment sentences in our dataset. The distribution of positive and negative sentences across categories is reported in Table 2.

We randomly partitioned the comment sentence dataset into training (80%) and testing (20%) sets, both containing a similar proportion of positive and negative sentences as the entire set of sentences for a category. The dataset was provided in CSV files where the attribute *ID* represents the unique sentence ID, *class* represents the class name referring to the source code file where the sentence comes from, *sentence* represents the text of the sentence, *partition* denotes the dataset it belongs to, *i.e., one* for training and *zero* for testing, *category* denotes the ground-truth category the sentence belongs to. The distribution of these sentences in both training and test sets is reported in Table 2.

3.2 Baselines and Competition Rules

We trained and tested 19 binary classifiers (one for each category) using the Sentence Transformer architecture on the provided training and test sets. The baseline classifiers, coined as STACC and proposed by Al-Kaswan et al. [2] (the winners of the NLBSE'23 Code Comment Classification Competition [32]), are lightweight classifiers developed using SetFit, an efficient and prompt-free framework for few-shot fine-tuning of Sentence Transformers. For fine-tuning, Al-Kaswan et al. relied on the Optuna backend with SetFit to find the best hyperparameters. The dataset is pre-processed by appending the code file name to the corresponding comments, separated by the '|' symbol. We make the models available on the HuggingFace Hub.[6] The replication package is available on GitHub.[7]

The participants were expected to train their classification models using the provided training dataset and evaluate them on the testing dataset. However, we restricted the use of any external sources beyond the class comment sentences and associated source code of the classes. Note that the dataset provides the mapping of a class name to its class comment sentences and to its project so that the participants could identify the source code of the class from the project and thus can leverage it to fine-tune their models. The projects' source code was released in our GitHub repository.

Despite the restriction on external sources, the participants were permitted to use pre-trained models as long as they were fine-tuned on the given training set. Also, they were allowed to perform pre-processing, sampling, over/under-sampling, and feature selection and engineering on the training dataset, but they were prohibited from performing the same steps on the testing set except for pre-processing and feature engineering.

Since the competition focused on binary classification for a given category, *i.e.,* a sentence does or does not belong to a category, we evaluated the classification performance of each classifier using Precision, Recall, and F_1-score on the testing set. Although the participants were expected to report these three metrics, we used the F_1-score to measure the overall performance of the models. The 19 F_1-scores of the proposed classifiers were compared against the 19 F_1-scores achieved by the baseline classifiers to rank the participants and determine a winner. We only allowed the classifiers to implement a single model, *e.g.,* BERT or SVM, for all categories, rather than implementing distinct models for different categories.

To guarantee the usability of the tools we put some lower bounds on the computational resources used for inference. We measured the runtime of the models in a free Google Colab T4 instance so that all participants could measure the runtime on a unified platform. The participants were allowed to use all the features available on the instance, including GPU acceleration. Only the time required to do inference is measured, time needed to load the models and data is not considered. We provide a testbench with instructions on how to load the data and models. [8]

The winner of the competition was the model with the highest score as determined by the following formula:

$$score(m) = (avg.\ F_1) \times 0.75 + (\frac{max\_avg\_rt - m\_avg\_rt}{max\_avg\_rt}) \times 0.25$$

where *score(m)* represents the score of the model m, $avg.\ F_1$ is the average of the F_1-scores achieved by the proposed model across all the 19 categories, *max_avg_rt* indicates the maximum average inference runtime (set to 0.005 seconds), and *m_avg_rt* is the actual average inference runtime of the proposed classifiers. The runtime should be measured ten times and should averaged across all categories and samples. With this formula, we encourage the participants to maximize the overall classification effectiveness and minimize the runtime of their models for practical purposes.

[6] STACC baseline models: https://huggingface.co/collections/AISE-TUDelft/stacc-65254a9b4d1fadc125a731dc
[7] Replication package: https://github.com/nlbse2024/code-comment-classification
[8] Testbench: https://colab.research.google.com/drive/1lvXuzdl_vSwMTCGIEfqTyQC1nzl22WCy

Table 2: Distribution of positive/negative comment sentences per category, language, and dataset (training and testing).

Language	Categories	Training			Testing			Training + Testing		
		Positive	*Negative*	*Total*	*Positive*	*Negative*	*Total*	*Positive*	*Negative*	*Total*
Java	Expand	662	7,779	8,441	166	1,948	2,114	828	9,727	10,555
	Ownership	438	8,001	8,439	112	2,004	2,116	550	10,005	10,555
	Deprecation	143	8,299	8,442	37	2,076	2,113	180	10,375	10,555
	Rational	422	8,019	8,441	106	2,008	2,114	528	10,027	10,555
	Summary	3,643	4,796	8,439	915	1,201	2,116	4,558	5,997	10,555
	Pointer	1,094	7,345	8,439	276	1,840	2,116	1,370	9,185	10,555
	Usage	2,374	6,067	8,441	595	1,519	2,114	2,969	7,586	10,555
		8,776	50,306	59,082	2,207	12,596	14,803	10,983	62,902	73,885
Pharo	Responsibilities	267	1,139	1,406	69	290	359	336	1,429	1,765
	Key messages	242	1,165	1,407	63	295	358	305	1,460	1,765
	Key impl. points	184	1,222	1,406	48	311	359	232	1,533	1,765
	Collaborators	99	1,307	1,406	28	331	359	127	1,638	1,765
	Example	596	812	748	152	205	357	748	1,017	1,765
	Class references	60	1,348	1,408	17	340	357	77	1,688	1,765
	Intent	173	1,236	1,409	45	311	356	218	1,547	1,765
		1,621	8,229	9'850	422	2,083	2'505	2'043	10'312	12,355
Python	Expand	402	1,637	2,039	102	414	516	504	2,051	2,555
	Parameters	633	1,404	2,037	161	357	518	794	1,761	2,555
	Summary	361	1,678	2,039	93	423	516	454	2,101	2,555
	Dev. notes	247	1,792	2,039	65	451	516	312	2,243	2,555
	Usage	637	1,401	2,038	163	354	517	800	1,755	2,555
		2,280	7,912	10,192	584	1,999	2,583	2,864	9,911	12,775

3.3 Submitted Classification Models

Hai and Bui [21] were the single team that participated in the competition with their Transformer-based Code Comment Classifier called Dopamin. The team implemented a number of design decisions for their approach to outperform the baseline classifier, namely backbone model selection, domain post-training, model checkpoint selection, and multilevel aggregation.

Dopamin is based on a single backbone model. They performed model selection with CodeBERT [19], RoBERTa [35], and ALBERT [34] as potential backbone models and found that CodeBERT performs best on a validation set created from 10% of the training set.

Domain post-training consisted of combining the data of all languages to post-train the backbone model. The category was concatenated to the comment sentences and the model was trained to predict whether the comment belongs to the category or not. The purpose of this strategy was to perform knowledge transfer from high-resource languages to low-resource languages before performing individual training of models for each category.

Dopamin, after model post-training, was then trained on the training set of each category and the best model checkpoint was selected based on the performance obtained on the validation set. The best model checkpoint for each category was trained on the full training set.

The architecture of the backbone model was adapted such that the output of one upper layer of the BERT model is used as input to another upper layer, then the output of all upper layers is aggregated by applying the mean operator, and this result is taken as input to a linear classifier layer that predicts the category of a comment sentence. The goal was to obtain a more comprehensive representation of the comment sentence.

3.4 Classifier Evaluation and Results

We followed the instructions provided in the replication package of the participating team. The submission contained both a GitHub repository containing the scripts to completely rerun the training and evaluation scripts as well as a HuggingFace Model Collection of the trained models. We loaded and tested the provided trained models and we trained the models from scratch using the provided scripts. For this submission, we successfully replicated the results reported in the corresponding paper in both cases.

Table 3 shows the performance of both the baseline (STACC [2]) and the proposed approach by Hai and Bui [21]. Dopamin outperforms the baseline's avg. precision (0.73 vs. 0.69, while achieving a similar avg. recall (0.74 vs. 0.74), thus leading to an overall higher avg. F1 score (0.74 vs 0.71). These improvements came mostly from a subset of 11 categories as Dopamin was not able to outperform the baseline for 8 categories. Dopamin outperforms the baseline across programming languages, each having categories that resulted in both performance improvement and degradation. According to Hai and Bui, Dopamin works better in categories that require context and semantics understanding such as Summary, Usage, and Ownership. The Python categories Parameters, Summary, and Usage presented performance improvement compared to the baseline, and the fact that they are also Java categories indicates the knowledge transfer from the model post-training step is beneficial for classification. The authors present an ablation study that tested the combinations of model pos-training and layer aggregation, obtaining the best results when both strategies are used.

In summary, Dopamin, the model proposed by Hai and Bui [21] outperforms the baseline model STACC, while achieving a similar inference runtime performance. These results make Dopamin, and its team, the winner of the competition.

Table 3: Results of the code comment classification competition. The models are ranked using the score given in section 3.2.

Participants	Classification Model	Average Precision	Average Recall	Average F_1-score	Outperformed Categories	Avg. Inference Runtime (secs.)	Ranking Score
Al-Kaswan et al.	STACC (baseline) [2]	0.69	0.76	0.71	-	0.00215	0.675
Hai and Bui	Dopamin [21]	0.73	0.75	0.74	11/19	0.00201	0.703

4 CONCLUSIONS AND FINAL REMARKS

The NLBSE'24 Tool Competition attracted six teams that proposed a diverse set of classification models to automatically classify issue reports or code comments.

The five issue report classification contestants were provided a new dataset sampled from five popular GitHub repositories, resulting in more consistent labeling rationale across issue reports. Variability in labeling rationale of the dataset was the main item of critique in previous editions. All submissions utilized models based on the Transformer architecture, leveraging various information sources from the issues. While most of these classifiers achieved comparable average classification performance, Ebrahim and Joy [17] have outranked the other contestants by a considerable margin. The choice of Adapters on top of the RoBERTa model appears to be the main factor for achieving such performance. Adapters not only perform best in our benchmark, they also are parameter-efficient and modular, making them more practical. The smaller dataset likely attracted more submissions as it lowered the computational cost of model fine-tuning.

A single team participated in the code comment classification competition, their approach outperforms the baseline model based on SetFit [2]. Dopamin [21] is a Transformers-based approach. Hai and Bui use a multi-level layer aggregation strategy to build their model on a CodeBERT [18] backbone. Similar to Al-Kaswan et al. [2] the classname is appended to the comment sentence and provided as input to the model for pre-training. For post-training all the training data was combined to facilitate the transfer of knowledge, as Pharo and Python have far fewer training instances. The combination of layer aggregation post-training allowed Dopamin to beat STACC in most categories. Combined with a similar runtime, it allowed Dopamin to beat the submission score set by STACC.

We expect that future editions of the competition would lead to more accurate models as well as their application to additional software engineering tasks that require the analysis and processing of (non)code-related textual artifacts. We also plan to extend the competition with techniques previously used for user review analysis [12, 14, 36, 39], categorizing safety-related issues [16], or fine-grained analysis of bug reports [7, 8, 48–50].

ACKNOWLEDGMENTS

We thank all the participants of the competition for their support in launching the third edition of the NLBSE'24 Tool Competition. We gratefully acknowledge the Horizon 2020 (EU Commission) support for the project *COSMOS* (DevOps for Complex Cyber-physical Systems), Project No. 957254-COSMOS. Chaparro was supported in part by grant CCF-1955853 from the NSF.

REFERENCES

[1] Alireza Aghamohammadi, Maliheh Izadi, and Abbas Heydarnoori. 2020. Generating summaries for methods of event-driven programs: An Android case study. *Journal of Systems and Software* 170 (2020), 110800. Publisher: Elsevier.

[2] Ali Al-Kaswan, Maliheh Izadi, and Arie van Deursen. 2023. STACC: Code Comment Classification using Sentence Transformers. In *Proceedings of The 2nd International Workshop on Natural Language-based Software Engineering (NLBSE'23)*. (to appear).

[3] Khubain Amjad Alam, Ashish Jumani, Harris Aamir, and Muhammad Uzair. 2024. ClassifAI: Automating Issue Reports Classification using Pre-Trained BERT (Bidirectional Encoder Representations from Transformers) Models. In *Proceedings of The 3rd International Workshop on Natural Language-based Software Engineering (NLBSE'24)*. Lisbon, Portugal.

[4] Gabriel Aracena, Kyle Luster, Fabio Santos, Igor Steinmacher, and Marco Aurelio Gerosa. 2024. Applying Large Language Models API to Issue Classification Problem. In *Proceedings of The 3rd International Workshop on Natural Language-based Software Engineering (NLBSE'24)*. Lisbon, Portugal.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris 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. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[6] Jordi Cabot, Javier Luis Cánovas Izquierdo, Valerio Cosentino, and Belén Rolandi. 2015. Exploring the use of labels to categorize issues in open-source software projects. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 550–554.

[7] Oscar Chaparro, Carlos Bernal-Cárdenas, Jing Lu, Kevin Moran, Andrian Marcus, Massimiliano Di Penta, Denys Poshyvanyk, and Vincent Ng. 2019. Assessing the quality of the steps to reproduce in bug reports. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'19)*. 86–96.

[8] Oscar Chaparro, Jing Lu, Fiorella Zampetti, Laura Moreno, Massimiliano Di Penta, Andrian Marcus, Gabriele Bavota, and Vincent Ng. 2017. Detecting missing information in bug descriptions. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'17)*. 396–407.

[9] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416* (2022).

[10] Giuseppe Colavito, Filippo Lanubile, and Nicole Novielli. 2023. Few-Shot Learning for Issue Report Classification. In *Proceedings of The 2nd International Workshop on Natural Language-based Software Engineering (NLBSE'23)*. (to appear).

[11] Sergio Cozzetti B de Souza, Nicolas Anquetil, and Káthia M de Oliveira. 2005. A study of the documentation essential to software maintenance. In *Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information*. 68–75.

[12] Andrea Di Sorbo, Giovanni Grano, Corrado Aaron Visaggio, and Sebastiano Panichella. 2021. Investigating the criticality of user-reported issues through their relations with app rating. *J. Softw. Evol. Process.* 33, 3 (2021).

[13] Andrea Di Sorbo and Sebastiano Panichella. 2023. Summary of the 1st Natural Language-based Software Engineering Workshop (NLBSE 2022). *ACM SIGSOFT Softw. Eng. Notes* 48, 1 (2023), 101–104. https://doi.org/10.1145/3573074.3573101

[14] Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Junji Shimagaki, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C. Gall. 2016. What would users change in my app? summarizing app reviews for recommending software changes. In *International Symposium on Foundations of Software Engineering*. ACM, 499–510. https://doi.org/10.1145/2950290.2950299

[15] Andrea Di Sorbo, Corrado Aaron Visaggio, Massimiliano Di Penta, Gerardo Canfora, and Sebastiano Panichella. 2021. An NLP-based Tool for Software Artifacts Analysis. In *IEEE International Conference on Software Maintenance and Evolution, ICSME, Luxembourg*. IEEE, 569–573. https://doi.org/10.1109/ICSME52107.2021.00058

[16] Andrea Di Sorbo, Fiorella Zampetti, Corrado A. Visaggio, Massimiliano Di Penta, and Sebastiano Panichella. 2022. Automated Identification and Qualitative Characterization of Safety Concerns Reported in UAV Software Platforms. *ACM Trans. Softw. Eng. Methodol.* (Sept. 2022). https://doi.org/10.1145/3564821 Place: New York, NY, USA Publisher: Association for Computing Machinery.

[17] Fahad Ebrahim and Mike Joy. 2024. Few-Shot Issue Report Classification with Adapters. In *Proceedings of The 3rd International Workshop on Natural Language-based Software Engineering (NLBSE'24)*. Lisbon, Portugal.

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

[19] Zhangyin Feng, Daya Wang, Minghui Li, Lijun Tang, Xiaohan Wang, Yingming Liu, Jingjing Shao, and Guolin Zhou. 2020. CodeBERT: A Pre-trained Model for Programming and Natural Languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 5871–5880. https://doi.org/10.18653/v1/2020.acl-main.520

[20] Daniel Gómez-Barrera, Luccas Rojas Becerra, Juan Pinzón Roncancio, David Ortiz Almanza, Juan Arboleda, Mario Linares, and Ruben Manrique. 2024. Lessons from the NLBSE 2024 Competition: Towards Building Efficient Models for GitHub Issue Classification. In *Proceedings of The 3rd International Workshop on Natural Language-based Software Engineering (NLBSE'24)*. Lisbon, Portugal.

[21] Nam Le Hai and Nghi DQ Bui. 2024. Dopamin: Transformer-based Comment Classifiers through Domain Post-Training and Multi-level Layer Aggregation. In *Proceedings of The 3rd International Workshop on Natural Language-based Software Engineering (NLBSE'24)*. Lisbon, Portugal.

[22] Steffen Herbold, Alexander Trautsch, and Fabian Trautsch. 2020. On the feasibility of automated prediction of bug and non-bug issues. *Empirical Software Engineering* 25, 6 (2020), 5333–5369. https://doi.org/10.1007/s10664-020-09885-w

[23] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-Efficient Transfer Learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 2790–2799. https://proceedings.mlr.press/v97/houlsby19a.html ISSN: 2640-3498.

[24] Maliheh Izadi. 2022. CatIss: An Intelligent Tool for Categorizing Issues Reports using Transformers. In *2022 IEEE/ACM 1st International Workshop on Natural Language-Based Software Engineering (NLBSE)*. 44–47. https://doi.org/10.1145/3528588.3528662

[25] Maliheh Izadi and Matin Nili Ahmadabadi. 2022. On the evaluation of NLP-based models for software engineering. In *2022 IEEE/ACM 1st International Workshop on Natural Language-Based Software Engineering (NLBSE)*. IEEE, 48–50.

[26] Maliheh Izadi, Kiana Akbari, and Abbas Heydarnoori. 2022. Predicting the objective and priority of issue reports in software repositories. *Empirical Software Engineering* 27, 2 (2022), 50. Publisher: Springer.

[27] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of Tricks for Efficient Text Classification. http://arxiv.org/abs/1607.01759 arXiv:1607.01759 [cs].

[28] Rafael Kallis, Oscar Chaparro, Andrea Di Sorbo, and Sebastiano Panichella. 2022. NLBSE'22 Tool Competition. In *Proceedings of The 1st International Workshop on Natural Language-based Software Engineering (NLBSE'22)*. https://doi.org/10.1145/3528588.3528664

[29] Rafael Kallis, Giuseppe Colavito, Oscar Chaparro, Luca Pascarella, Ali Al-Kaswan, and Pooja Rani. 2024. The NLBSE'24 Tool Competition. In *Proceedings of The 3rd International Workshop on Natural Language-based Software Engineering (NLBSE'24)*.

[30] Rafael Kallis, Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. 2019. Ticket Tagger: Machine Learning Driven Issue Classification. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 406–409. https://doi.org/10.1109/ICSME.2019.00070 ISSN: 2576-3148.

[31] Rafael Kallis, Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. 2021. Predicting issue types on GitHub. *Science of Computer Programming* 205 (May 2021), 102598. https://doi.org/10.1016/j.scico.2020.102598

[32] Rafael Kallis, Maliheh Izadi, Luca Pascarella, Oscar Chaparro, and Pooja Rani. 2023. The NLBSE'23 Tool Competition. In *2023 IEEE/ACM 2nd International Workshop on Natural Language-Based Software Engineering (NLBSE)*. IEEE, Melbourne, Australia, 1–8. https://doi.org/10.1109/NLBSE59153.2023.00007

[33] Muhammad Laiq. 2023. An Intelligent Tool for Classifying Issue Reports. In *Proceedings of The 2nd International Workshop on Natural Language-based Software Engineering (NLBSE'23)*. (to appear).

[34] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=H1eA7AEtvS

[35] 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. https://doi.org/10.48550/arXiv.1907.11692 arXiv:1907.11692 [cs].

[36] Sebastiano Panichella. 2018. Summarization techniques for code, change, testing, and user feedback (Invited paper). In *2018 IEEE Workshop on Validation, Analysis and Evolution of Software Tests, VST@SANER 2018, Campobasso, Italy, March 20, 2018*, Cyrille Artho and Rudolf Ramler (Eds.). IEEE, 1–5.

[37] Sebastiano Panichella, Gerardo Canfora, and Andrea Di Sorbo. 2021. Won't We Fix this Issue? Qualitative characterization and automated identification of wontfix issues on GitHub. *Information and Software Technology* 139 (2021), 106665. https://doi.org/10.1016/j.infsof.2021.106665

[38] Sebastiano Panichella and Andrea Di Sorbo. 2023. Summary of the 2nd Natural Language-based Software Engineering Workshop (NLBSE 2023). *ACM SIGSOFT Software Engineering Notes* 48, 4 (Oct. 2023), 60–63. https://doi.org/10.1145/3617946.3617957

[39] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C. Gall. 2015. How can I improve my app? Classifying user reviews for software maintenance and evolution. In *International Conference on Software Maintenance and Evolution*. IEEE, 281–290. https://doi.org/10.1109/ICSM.2015.7332474

[40] Luca Pascarella and Alberto Bacchelli. 2017. Classifying code comments in Java open-source software systems. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 227–237.

[41] Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020. AdapterHub: A Framework for Adapting Transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Qun Liu and David Schlangen (Eds.). Association for Computational Linguistics, Online, 46–54. https://doi.org/10.18653/v1/2020.emnlp-demos.7

[42] Clifton Poth, Hannah Sterz, Indraneil Paul, Sukannya Purkayastha, Leon Engländer, Timo Imhof, Ivan Vulić, Sebastian Ruder, Iryna Gurevych, and Jonas Pfeiffer. 2023. Adapters: A Unified Library for Parameter-Efficient and Modular Transfer Learning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Singapore, 149–160. https://aclanthology.org/2023.emnlp-demo.13

[43] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.* 21, 1, Article 140 (jan 2020), 67 pages.

[44] Pooja Rani, Arianna Blasi, Nataliia Stulova, Sebastiano Panichella, Alessandra Gorla, and Oscar Nierstrasz. 2023. A decade of code comment quality assessment: A systematic literature review. *Journal of Systems and Software* 195 (2023), 111515. https://doi.org/10.1016/j.jss.2022.111515

[45] Pooja Rani, Sebastiano Panichella, Manuel Leuenberger, Andrea Di Sorbo, and Oscar Nierstrasz. 2021. How to identify class comment types? A multi-language approach for class comment classification. *Journal of Systems and Software* 181 (2021), 111047.

[46] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. https://doi.org/10.48550/arXiv.1908.10084 arXiv:1908.10084 [cs].

[47] Gokul Rejithkumar, Preethu Rose Anish, and Smita Ghaisas. 2024. Text-To-Text Generation for Issue Report Classification. In *Proceedings of The 3rd International Workshop on Natural Language-based Software Engineering (NLBSE'24)*. Lisbon, Portugal.

[48] Yang Song and Oscar Chaparro. 2020. Bee: A tool for structuring and analyzing bug reports. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'20)*. 1551–1555.

[49] Yang Song, Junayed Mahmud, Nadeeshan De Silva, Ying Zhou, Oscar Chaparro, Kevin Moran, Andrian Marcus, and Denys Poshyvanyk. 2023. BURT: A Chatbot for Interactive Bug Reporting. In *Proceedings of the 45th International Conference on Software Engineering (ICSE)*. (to appear).

[50] Yang Song, Junayed Mahmud, Ying Zhou, Oscar Chaparro, Kevin Moran, Andrian Marcus, and Denys Poshyvanyk. 2022. Toward interactive bug reporting for (Android app) end-users. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'22)*. 344–356.

[51] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An Instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.

[52] Lewis Tunstall, Nils Reimers, Unso Eun Seo Jo, Luke Bates, Daniel Korat, Moshe Wasserblat, and Oren Pereg. 2022. Efficient Few-Shot Learning Without Prompts. *arXiv preprint arXiv:2209.11055* (2022).

[53] Xingquan Zhu and Xindong Wu. 2004. Class Noise vs. Attribute Noise: A Quantitative Study. *Artificial Intelligence Review* 22, 3 (Nov. 2004), 177–210. https://doi.org/10.1007/s10462-004-0751-8