Engineering Applications of Artificial Intelligence 124 (2023) 106485

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Monolingual, multilingual and cross-lingual code comment classification

Marija Kostić [a,b,*], Vuk Batanović [b], Boško Nikolić [a]

[a] University of Belgrade, School of Electrical Engineering, Bulevar kralja Aleksandra 73, 11120 Belgrade, Serbia
[b] University of Belgrade, Innovation center of the School of Electrical Engineering, Bulevar kralja Aleksandra 73, 11120 Belgrade, Serbia

ARTICLE INFO

Dataset link: Code comment categorization corpus (Original data)

Keywords:
Code comments
Code comment classification
Pre-trained language models
Transformers
Annotated dataset

ABSTRACT

Code comments are one of the most useful forms of documentation and metadata for understanding software implementation. Previous research on code comment classification has focused only on comments in English, typically extracted from a few programming languages. This paper addresses the problem of code comment classification not only in the monolingual setting, but also in the multilingual and cross-lingual one, in order to examine whether they can outperform the traditional monolingual approach. To tackle this task, we introduce a novel, publicly available code comment dataset, consisting of over 10,000 code comments collected from software projects written in eight programming languages (C, C++, C#, Java, JavaScript/TypeScript, PHP, Python, and SQL). About half of them are written in Serbian while the other half are written in English. This dataset was manually annotated according to a newly proposed taxonomy of code comment categories. We fine-tune and evaluate multiple monolingual and multilingual pre-trained neural language models on the code comment classification task and compare their performances to several baselines. The best results for Serbian comments are obtained using the monolingual neural model BERTić, trained on Serbian and closely related languages. On the other hand, the optimal choice for English is the multilingual neural model multilingual BERT, which successfully extracts useful patterns from data in both languages. Although the cross-lingual setting shows some promise for simple binary classification, it has yet to reach sufficiently high performance levels for practical use. We also analyze model performance across different programming languages.

1. Introduction

Source code comments are an integral part of computer programs and, in many cases, make up for the lack of proper software documentation. Comments are the second most used documentary artifact after code (de Souza et al., 2005). They carry rich information about the underlying software implementation, such as descriptions of the code functionality and properties, license and authorship information, usage instructions, notes about potential or observed bugs and issues, etc. In addition to being an excellent documentation source, comments are particularly important for source code maintainability (Hartzman and Austin, 1993) and readability. For instance, Woodfield et al. (1981) conducted one of the first experiments showing that commented code is easier to understand and read compared to code without comments. These findings were later confirmed by Tenny (1985, 1988). Code commenting has been recognized as a good programming practice (de Souza et al., 2005), and comments play a vital role in helping developers comprehend source code (Chen et al., 2021).

Code comments can greatly differ depending on their purpose, target audience, or other traits. For instance, some comments target developers, while others target end-users of the program. Some describe the functionality of the related code, some the design rationale or implementation details, while others are simply used as separators of logical blocks of code. Developers usually comment on standard code elements such as methods, classes, and variables. Hence, the interpretation of a given comment can heavily depend on its position in the source code. Moreover, unlike source code which must follow the rules of a programming language, code comments allow substantial freedom of expression. Currently, there exist several style guides for writing comments. Some of them are designed for a particular programming language, e.g., C++ (Google, 2022a), Python (Google, 2022b) or Java (Oracle, 2022), whereas some of them are more general (Kramer, 1999; Tan et al., 2007). Regardless, following them is not obligatory, and they do not precisely define the instructions for writing all types of comments in a program. The structure and style of comments are mostly left to the developers' discretion. Furthermore, code comments are usually written in a natural language, adding to their complexity. Because of this, comments can become outdated and hard to maintain. To enforce more discipline in writing and maintaining comments there are a few approaches to write comments as code in a formal language. For example, Nie et al. (2019) developed a framework

[*] Corresponding author at: University of Belgrade, Innovation center of the School of Electrical Engineering, Bulevar kralja Aleksandra 73, 11120 Belgrade, Serbia.

E-mail addresses: marija.kostic@ic.etf.bg.ac.rs (M. Kostić), vuk.batanovic@ic.etf.bg.ac.rs (V. Batanović), bosko.nikolic@etf.bg.ac.rs (B. Nikolić).

https://doi.org/10.1016/j.engappai.2023.106485
Received 6 November 2022; Received in revised form 26 April 2023; Accepted 17 May 2023
Available online 10 June 2023

that automatically evaluates, at each build, trigger-action comments written in executable format. This framework takes actions based on users' specifications. For Java, the Javadoc markup language and tool have existed since Java's first release in 1995 and Doxygen, a similar but cross-language tool appeared two years later. Tools like Jdoctor developed by Blasi et al. (2018) automatically construct executable procedure specifications from comments in this format.

Code comments provide information that has been leveraged to help perform a wide range of software engineering tasks, such as bug detection (Zhai et al., 2020), comment propagation (Zhai et al., 2020), code summarization (Chen et al., 2021), code synthesis (Gvero and Kuncak, 2015; Zhai et al., 2016), specification inference (Pandita et al., 2012; Zhong et al., 2009), etc. Depending on the downstream task in focus, not all code comments are equally important. Therefore, for many of the tasks, the first step in tackling them is the identification of relevant comments, typically framed in the form of code comment classification.

In recent years, multilingual and cross-lingual studies have become the standard in Natural Language Processing (Devlin et al., 2018; Conneau et al., 2019; Grave et al., 2018). However, previous work in code comment classification has been focused solely on monolingual settings, since no publicly available multilingual datasets for this task were available. Furthermore, previous research in this field is highly fragmented, both in terms of the programming languages from which code comments were collected and in terms of the used comment classification taxonomy. Typically, each paper presents its own set of code comment categories, most often tailored to a particular downstream task in mind (Kostić et al., 2022).

Padioleau et al. (2009), Haouari et al. (2011), and Steidl et al. (2013) presented the earliest results in comment classification. Padioleau et al. (2009) studied code comments to understand developers' needs regarding the creation of new tools and languages or the improvement of the existing ones. Haouari et al. (2011) investigated developers' commenting habits by looking at the distribution of comments concerning the program construct type that follows it. Steidl et al. (2013) developed a semi-automated model for comment quality analysis that is based on classifying comments into seven categories. On the other hand, Shinyama et al. (2018) focused only on the comments inside functions or methods, i.e., *inline comments*, which are not visible in the documentation and often give insight into a developer's intent. Zhai et al. (2020) wanted to leverage program analysis to systematically propagate comments so that they can be passed on to uncommented code entities and help detect bugs. Here comment classification was necessary because comments with different content related to similar programming entities cannot be propagated in the same way. Chen et al. (2021) investigated the use of the relationship between code blocks and the categories of the corresponding comments to improve code summarization. They have found that different summarization models work best for different comment categories. Using comment classification, they could design a composite summarization model that outperforms other approaches. Although code comment classification is mostly researched in the context of different downstream tasks, the study of Pascarella et al. (2019) focused on the empirical analysis of code comments to understand the types of comments developers write in their source code. Another such example is the work of Zhang et al. (2018), which focused on code comment classification without having any downstream task in mind.

This paper addresses the classification of code comments written in English, the most common language used in programming documentation, and Serbian, our native language, which is under-resourced in terms of NLP support. We wanted to see whether models trained on a high-resource language like English can help a low-resource language like Serbian, especially since words in English are frequently found in comments written in some other language. This may be because a specific term has an established usage in its English form or lacks an appropriate translation. We perform classification in three settings: monolingual (comments from English and Serbian are used separately), multilingual (models are trained on comments in both languages and then evaluated on each language separately), and cross-lingual (models trained on comments in one language are evaluated on comments in the other language). Another reason behind our interest in code comment classification are two downstream tasks – semantic code search (Husain et al., 2019) and cross-level semantic similarity (Jurgens et al., 2014, 2016) – for which the proper identification of comment types is very important. The main contributions of the work presented here are as follows:

- To the best of our knowledge, this is the first work that tackles multilingual and cross-lingual code comment classification. Previous research efforts have focused solely on monolingual classification, likely due to the scarcity of datasets containing code comments in multiple natural languages.
- A new code comment categorization taxonomy is presented, designed to support various programming languages and paradigms and multiple natural languages.
- A new dataset of around 10,000 comments is compiled, with half of the comments written in Serbian, while the other half is in English. These comments were taken from eight programming languages (C, C++, C#, Java, JavaScript/TypeScript, PHP, Python, and SQL) and were manually annotated according to the newly proposed taxonomy.
- We evaluate multiple statistical classifiers, ranging from linear baselines to state-of-the-art transformer-based neural language models, using the aforementioned annotated dataset.
- This work represents the first effort to perform automated code comment classification in Serbian. The developed models could also be of use in classifying comments in closely related languages, like Bosnian and Croatian.

The remainder of this paper is structured as follows: Section 2 presents previous work on code comment classification. Sections 3 and 4 describe the dataset creation process and analysis, respectively. Classification evaluation and its results are shown in Section 5, whereas they are discussed in Section 6. Finally, Section 7 contains conclusions and pointers for future work.

2. Related work

All of the previously mentioned studies used comments in English written within one of the following programming languages: C/C++, Java, or Python. The ones that implemented automated systems for comment classification usually did so by extracting specific features from the comments and using traditional machine learning models. Pascarella et al. (2019) used probabilistic classifiers such as multinomial Naïve Bayes and Decision Tree classifiers such as J48 and Random Forest. They worked with 40,000 Java comment lines from open source and closed source industry projects. They designed their classification taxonomy consisting of six high-level and 16 inner categories. The preprocessing of comments was executed in the following order: (1) tokenization on spaces and punctuation, (2) splitting the identifiers based on camel-casing, (3) lowercasing, (4) removing numbers and rare symbols, and (5) splitting the comments at line level. The input to the models were counts of occurrences of each word in the bag of unique words, and two groups of custom features. The first group relates to the context of the line, such as the text length, the number of rows, the comment position in the file, etc. The second group contains category-specific features recognized by different regular expressions. The metrics considered were precision and recall for each category and the average weighted true positive rate after a standard 10-fold cross-validation. Even though the overall accuracy was higher for Random Forest classifiers, the authors found that the best classifier is based on the probabilistic approach. The performance of the multinomial Naïve Bayes model is promising for the six high-level categories: for open source projects, precision and recall are always above 93%. In contrast, a performance drop down to 70% per category is detected for the industry projects. The same trend is visible in the inner categories. On average, the precision for inner categories is again better in open source projects compared to industrial ones. However, metrics drop down to 50% for some categories.

Steidl et al. (2013) used 830 Java and 500 C/C++ comments to train a J48 Decision Tree to classify comments into one of the seven proposed categories. They also created custom features like the number of words in a comment, booleans indicating whether the comment contains some special terms (ex. `copyright`, `license`, `fixme`, etc.), the percentage of special characters, etc. For evaluation, a standard five-fold cross-validation was used. The decision Tree achieved a weighted average precision and recall of 96% on Java comments. Similar results are obtained with the J48 tree applied to the C/C++ comments.

Another work that uses Decision Trees is the work of Shinyama et al. (2018). They used 1000 Java comments to find the best Decision Tree to perform classification into one of 11 categories. Unlike previous papers, here the features are based on part-of-speech (POS) tagging. The tagger assigns one of the 36 POS tags to each word (ex. *VBZ* for a verb in 3rd person or *NNS* for a plural noun). Although the metrics vary depending on the category, the classifier has a reasonable performance (61% precision and 89% recall) for the *postcondition* comment category, which was the target category of the research.

The comment classification described by Zhai et al. (2020) is unique for several reasons. First, the taxonomy contains two dimensions: *perspective* and *code entity*. Second, a comment can belong to multiple categories of one dimension simultaneously, making this a multi-label classification problem. Furthermore, the authors used a convolutional neural network (CNN) in addition to the Decision Tree and Random Forest models. Their training dataset contained 5000 Java comments. These were preprocessed in the following steps: (1) class/method/variable names were replaced with corresponding placeholders, (2) stop words were removed, (3) the remaining words were stemmed and (4) lowercased. To get the embeddings, word2vec algorithm was applied to the comment dataset instead of using the existing word embeddings trained on news articles. Also, eight custom features were generated based on part-of-speech tags. The Decision Tree and Random Forest used these features. Standard five-fold cross-validation was used, along with the four metrics: precision, recall, f1 score and Hamming loss. For the *perspective* dimension, CNN gives the best results, with the f1 score of 94%. However, the *code entity* perspective was best classified with the Random Forest (f1 score 99%). Here, CNN produced significantly lower scores (f1 score 82%).

Our interest in code comment classification was driven by two downstream tasks — semantic code search (Husain et al., 2019) and cross-level semantic similarity (Jurgens et al., 2014, 2016). In semantic code search, the goal is to retrieve the most relevant code block(s) for a given query in a natural language. To do so, most models rely not only on the code blocks but also on the accompanying comments, which describe code functionality (Husain et al., 2019). Cross-level semantic similarity, a variant of the semantic textual similarity task (Agirre et al., 2012), aims to assign a numerical semantic similarity score between two texts of different lengths written in a natural language (Jurgens et al., 2016). Such a similarity measure is of great use in semantic code search since it can detect semantic links between queries (usually limited to a couple of words) and code comment blocks (whose length varies from a phrase to a paragraph). Both downstream tasks focus on comments describing code functionality and/or usage. This makes it necessary to properly identify such comments within given source code files, regardless of the programming and natural languages used. However, the literature review showed no code comment classification systems and taxonomies designed with these two downstream tasks in mind.

3. Dataset construction and annotation

To examine code comment classification in multilingual and cross-lingual settings, it is necessary to use datasets in multiple natural languages that are hand-labeled according to the same set of categories and use the same annotation instructions. Since no such datasets were available, it was necessary to construct one. We focused on two natural languages: Serbian, our native language, and English. This allowed us to compare model behaviors in different settings. Serbian is a minor language, under-resourced in terms of NLP support, whereas English is the most prominent language used in software development. Serbian is also a morphologically rich language, whereas English is not. This means that substantial grammatical information in Serbian is expressed at a word level, using different word forms. For example, nouns are declined across seven grammatical cases, they can belong to one of three grammatical genders, and can sometimes have two distinct plural forms depending on the number (less or greater than five). Similarly, verbs can typically appear in dozens of different forms, depending on the tense, person, number, and grammatical gender. This allows words to change their position in a sentence, i.e., the word order is quite free and there are no articles (Popović and Arcan, 2015). To avoid limiting the research to merely one or two of the most popular programming languages, we included in the dataset code comment samples from multiple languages belonging to different programming paradigms and use cases. The selection of programming languages was also determined partly by the availability of adequate data in Serbian. Therefore, the set of programming languages that are considered includes the following: C, C++, C#, Java, JavaScript/TypeScript, PHP, Python, and SQL.

First, we compiled a dataset of around 65,000 comments with ∼70% of comments in English and ∼30% of comments in Serbian. These comments were extracted from various sources, including student projects (∼13% of comments), coursework at the University of Belgrade (∼10% of comments), software projects developed at the Computing Center of the School of Electrical Engineering, University of Belgrade and other industry partners (∼1% of comments), as well as public repositories such as GitHub (∼76% of comments). The information about their source file and line number was also retained for most comments. This allowed us to use the surrounding context of the comment in the annotation process, making it easier to select the appropriate comment category. Exceptions were some comments in Serbian obtained from industry partners, where only the comment texts were placed at our disposal, rather than entire source code files. After removing duplicates, 10,000 code comments, divided approximately equally between the two natural languages and the eight programming languages were randomly sampled from the remaining ∼49,000 comments. Table 1 shows one entry from the dataset for both Serbian (SR) and English (EN).

Next, we had to select a set of code comment categories to be used in data annotation. As mentioned in the previous section, the existing code comment taxonomies were neither designed with the downstream tasks of interest in mind nor applicable to multiple natural and programming languages (Kostić et al., 2022). The taxonomy of Padioleau et al. (2009) does not have explanations of all its categories. Zhai et al. (2020) classify comments according to two dimensions whereas Haouari et al. (2011) have even more dimensions (four), which is too complex for our use case. Shinyama et al. (2018) focused only on the inline comments, but other functional comments are just as important for our downstream tasks. The taxonomy introduced by Zhang et al. (2018) is tailored for Python comments and therefore cannot be easily applied to comments written in other programming languages. Chen et al. (2021) distinguish between different content perspectives and do not have a separate category for functional comments. The remaining two taxonomies (Pascarella et al., 2019; Steidl et al., 2013) do put the emphasis on the comments that describe code functionality. That is why they are used as inspiration for our own taxonomy.

For the downstream tasks of interest, the most important comments are the ones that describe code functionality. The first step is

Table 1
One entry from the dataset for each of the natural languages.

Natural language	Programming language	Source path and line number	Comment text
EN	PHP	symfony/src/Symfony/Component/ Stopwatch/ Section.php/88	`@return string The identifier of the section.`
SR	C#	samedb/Studentska-sluzba/Aplikacija/Studentska-služba/DatabaseController.cs/106	`Funkcija ComputeHash prima i vraca byte[] pa zbog toga ova konverzija string u byte[] i nazad` (English: `The ComputeHash function receives and returns byte[] thus the conversion of the string to the byte[] and back.`)

distinguishing between functional and non-functional comments via two top-level categories. These two categories are then divided into eight subcategories. Functional comments are further divided based on the type of the corresponding source code because we believe that will be useful for the semantic code search. Even though non-functional subcategories are not required for the downstream tasks of interest, we still included them in our taxonomy to make the annotated datasets usable for other tasks, such as code comment summarization, automated code comment generation, etc. We designed our own non-functional categories since the taxonomy of Pascarella et al. (2019) is very demanding to work with (it contains as many as thirteen non-functional categories). On the other hand, the taxonomy of (Steidl et al., 2013) has only four non-functional categories, all closely related to Java and C++ programming languages, which are not sufficient to cover all the diverse comments from various programming languages we are using in our research. Our categories are mutually exclusive. A detailed description of the proposed categories can be found in Kostić et al. (2022), but they are presented briefly below as well:

I. The **Functional** category contains comments describing the functionality of the corresponding code, its purpose, behavior, why something is implemented in a certain way, etc. These comments can respond to the questions *What?*, *Why?* and *How?*. Based on the type of the corresponding source code, three subcategories can be distinguished:

(1) **Functional-Module** comments that describe the functionality of a particular module like a class or an interface (e.g., `Base class for events thrown in the HttpKernel component.`);
(2) **Functional-Method** comments that describe the functionality of a function or a method (e.g., `Returns a generated Firestore Document Id.`);
(3) **Functional-Inline** comments that describe the functionality of a variable or an expression (e.g., `create an appropriate Entry object`).

II. The **Non-functional** category covers all comments that do not describe code functionality. It consists of five subcategories:

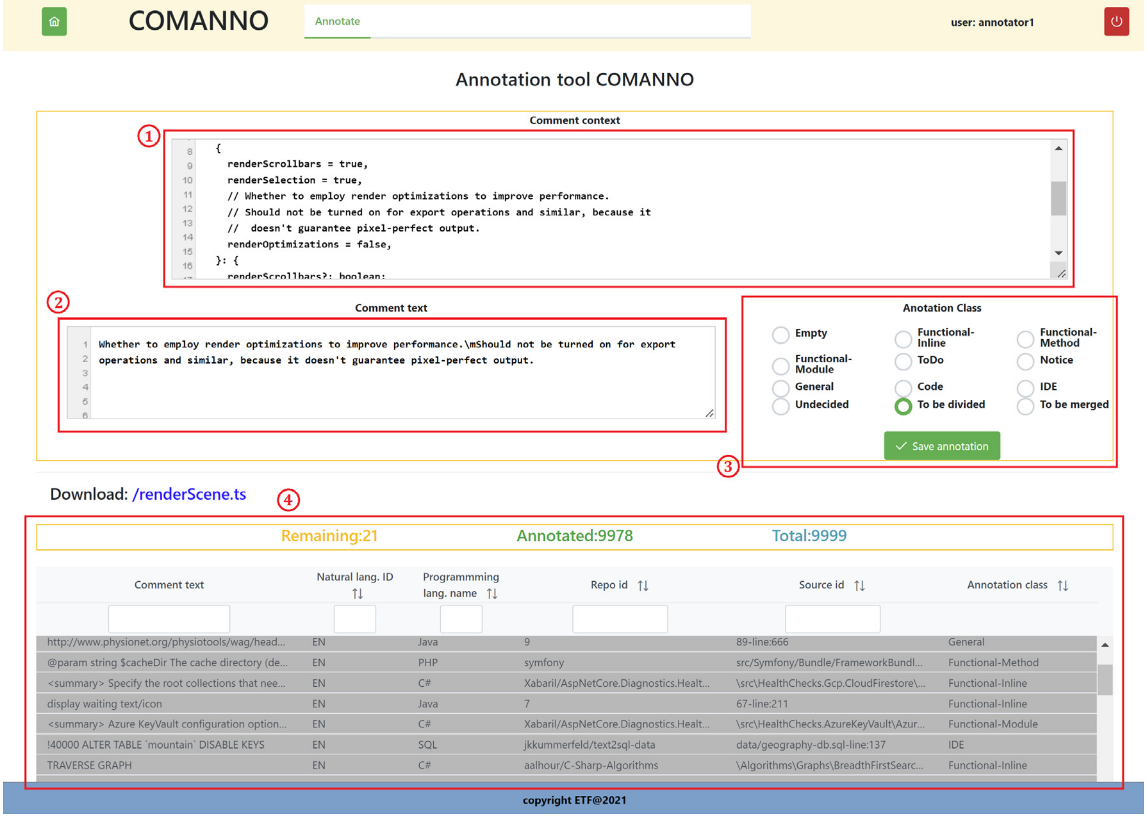
(4) **Code** — source code that has been commented out (e.g., `size = round_up(map->value_size, 8);`);
(5) **Notice** — warnings, alerts, and messages intended for the developers or users of the code, deprecated artifacts and instructions regarding their replacements, motivation for some implementation decisions, usage examples and suggestions (e.g., `Implement IDisposable. Do not make this method virtual. A derived class should not be able to override this method.`);
(6) **IDE** — comments that communicate with the Integrated Development Environment (IDE) or the compiler (e.g., `tslint:disable:no-big-function`);
(7) **ToDo** — explicit tasks to be done and notes about bugs that need to be fixed (e.g., `@todo Simplify the logic`);
(8) **General** — meta-information such as license, copyright, authorship, version, the name or the path of the file, etc. (e.g., `Copyright(c) 2019 Intel Corporation.`).

The initial annotation instructions consisted of comment category definitions and examples of typical kinds of comments, with instructions on how they should be labeled. Calibration of these instructions was performed by two annotators who used them to label a set of 500 randomly chosen comments in parallel. The first author served as the main annotator, with another graduate student in software engineering being the secondary annotator. Discussions were conducted for each point of disagreement between them, and the annotation instructions were updated accordingly. Finally, the main annotator assigned labels to all the comments, while the secondary one annotated a random subset of 20% of the comments. We used the labels on this subset to estimate the inter-annotator agreement. Since the obtained agreement levels were quite high, as discussed in Section 4 of the paper, it was unnecessary to search for additional annotators. The resulting dataset and the final annotation instructions (in Serbian) are publicly available at the following GitHub repository: https://github.com/ETF-NLP/AVANTES-Classification.

Annotation was performed using a custom-built web application called *Comanno*. This application was developed to expedite the repetitive and error-prone task of manual classification. Fig. 1 displays a screenshot of the application. *Comanno* shows the currently selected comment (②) and its surrounding code — context (①). An annotator can choose one of the predefined categories from the group of radio buttons on the right hand side (③). The table of all comments at the bottom of the screen (④) contains additional information such as the natural language, programming language, repository and file identifiers, and the assigned category. A user of this application can search through, and sort this table based on any of the provided columns.

The initial set of comment texts was obtained via direct extraction from source code files, where multiline comments were treated as single units. However, during the annotation process, such comments were found to sometimes consist of two or more separate and unrelated statements, belonging to different comment categories. To solve this issue, such comments were manually divided into multiple shorter ones, each belonging to a particular comment category. This led to slightly different final numbers of comments between the English and the Serbian subset. This is the reason why categories in the *Commano* tool include not only eight categories from the newly proposed taxonomy but also some additional categories that are useful for the calibration and annotation processes (e.g. *To be divided* category means that the corresponding comment should be manually divided into multiple comments outside of the tool. Another example is the *Undecided* category that indicates that the annotator is not sure which category to assign, and that the comment in question should be discussed as part of the calibration).

The manual annotation process was fairly similar for both English and Serbian comments. One minor difference is that *ToDo* comments in English almost always contain the keyword `TODO` or `FIXME`, making them easy to identify (e.g., `TODO: perform the search on a per txq basis.`). Conversely, their counterparts in Serbian tend to be much longer and more flexible in terms of the vocabulary they



```
  {
    renderScrollbars = true,
    renderSelection = true,
    // Whether to employ render optimizations to improve performance.
    // Should not be turned on for export operations and similar, because it
    //  doesn't guarantee pixel-perfect output.
    renderOptimizations = false,
  }: {
    renderScrollbars?: boolean;
```

```
Whether to employ render optimizations to improve performance.\mShould not be turned on for export
operations and similar, because it doesn't guarantee pixel-perfect output.
```

○ Empty ○ Functional-Inline ○ Functional-Method
○ Functional-Module ○ ToDo ○ Notice
○ General ○ Code ○ IDE
○ Undecided ● To be divided ○ To be merged

Download: /renderScene.ts

Remaining:21 Annotated:9978 Total:9999

Comment text	Natural lang. ID ⇅	Programmming lang. name ⇅	Repo id ⇅	Source id ⇅	Annotation class ⇅
http://www.physionet.org/physiotools/wag/head...	EN	Java	9	89-line:666	General
@param string $cacheDir The cache directory (de...	EN	PHP	symfony	src/Symfony/Bundle/FrameworkBundl...	Functional-Method
<summary> Specify the root collections that nee...	EN	C#	Xabaril/AspNetCore.Diagnostics.Healt...	\src\HealthChecks.Gcp.CloudFirestore\...	Functional-Inline
display waiting text/icon	EN	Java	7	67-line:211	Functional-Inline
<summary> Azure KeyVault configuration option...	EN	C#	Xabaril/AspNetCore.Diagnostics.Healt...	\src\HealthChecks.AzureKeyVault\Azur...	Functional-Module
!40000 ALTER TABLE `mountain` DISABLE KEYS	EN	SQL	jkkummerfeld/text2sql-data	data/geography-db.sql-line:137	IDE
TRAVERSE GRAPH	EN	C#	aalhour/C-Sharp-Algorithms	\Algorithms\Graphs\BreadthFirstSearc...	Functional-Inline

Fig. 1. Screenshot of the Comanno annotation tool.

employ (e.g., `dodaj opciju da ako lista ima jedan element, vrati tu vrednost i ne zove evalExp.` (English: `add an option that if the list has one element, that value is returned and evalExp is not called.`)).

4. Data analysis

The final dataset consists of 10,132 comments out of which both annotators annotated 2040 comments. There are 5039 comments in Serbian (SR) and 5093 comments in English (EN). No relevant code repositories with PHP code and comments in Serbian could be found. Regarding the JavaScript/TypeScript family of programming languages, the Serbian portion of the dataset contains only comments written in TypeScript. In contrast, the English portion includes approximately the same number of comments in both language variants. Fig. 2 shows the number of comments per programming language, natural language and category, whereas Fig. 3 shows the number of comments per category and natural language. In both natural languages, *Functional-Inline* comments make up around or over 50% of all comments, whereas the total number of *Functional* comments of all kinds reaches over 75%. *Code*, *Notice* and *ToDo* comment categories each make up around 5%–8% of data in both languages, with *General* comments around 2%. *IDE* comments are the least numerous, representing about 2% of comments in English, and are practically non-existent in Serbian data since all tools generate such comments only in English. The IDE category was therefore necessarily omitted from all model evaluations in Serbian.

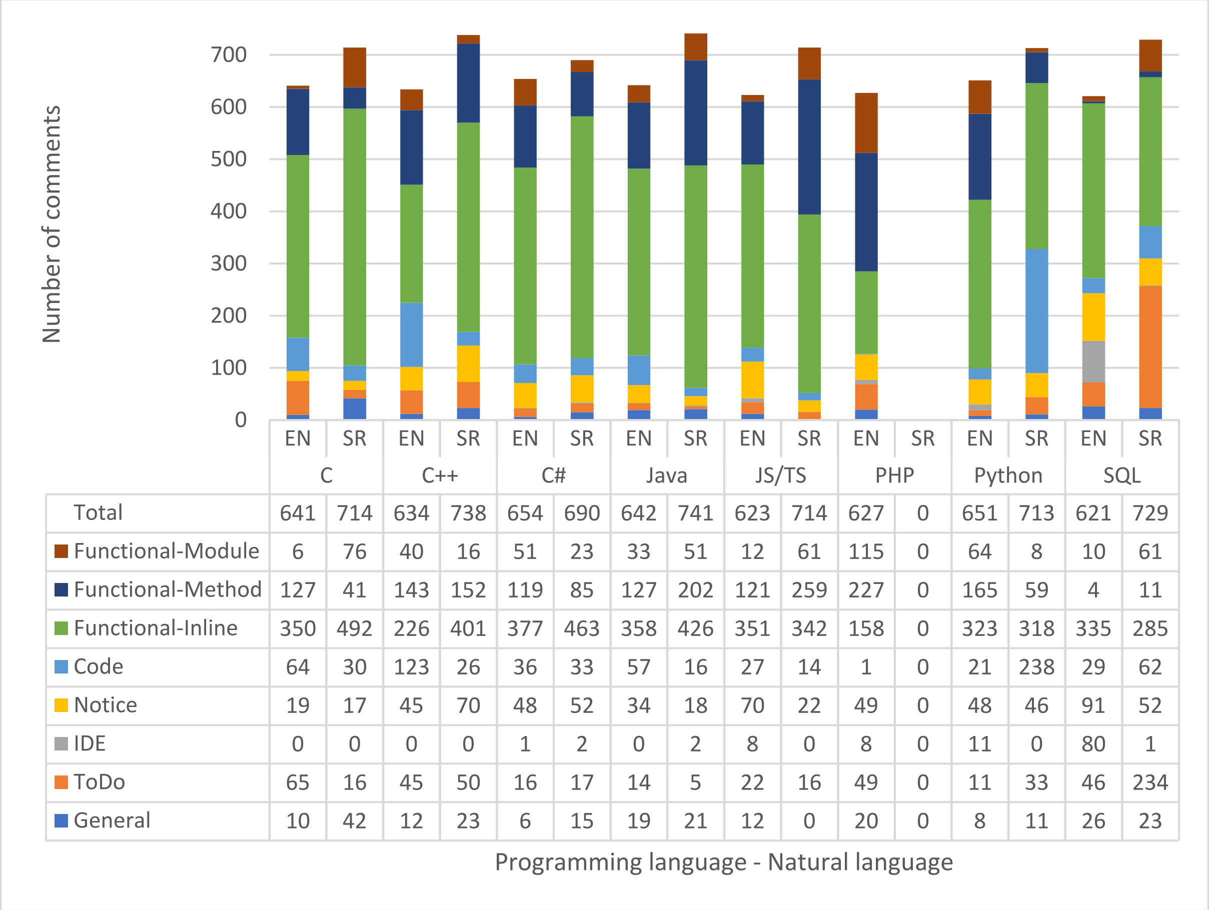
Another factor to consider is the distribution of comment lengths. We used the NLTK tokenization (Bird et al., 2009) for the comments in English, whereas for the comments in Serbian the ReLDI tokenizer[1] was applied. The average comment lengths, expressed in terms of the number of tokens, are shown in Fig. 4. For every comment category except *ToDo*, comments in English are longer, especially in the case of the *Functional-Method* and *General* categories. This is probably due to the difference in comment sources between the two languages, with English comments taken mostly from open-source project repositories with clearer documentation standards. Conversely, a major source for comments in Serbian were student projects, where code commenting was not a priority, leading to comments in Serbian being, on average, shorter and less detailed than the comments in English.

We measured the inter-annotator agreements using Cohen's Kappa coefficient (Cohen, 1960) and Krippendorff's alpha coefficient (Krippendorff, 2004). The agreement is measured for three variants of label interpretation:

1. **Binary** classification — functional vs. non-functional comments.
2. **Full** class set classification — all subcategories for both functional and non-functional comments are treated separately.
3. **Reduced** class set classification — all functional comments are combined into a single category, while the non-functional comments are divided into subcategories.

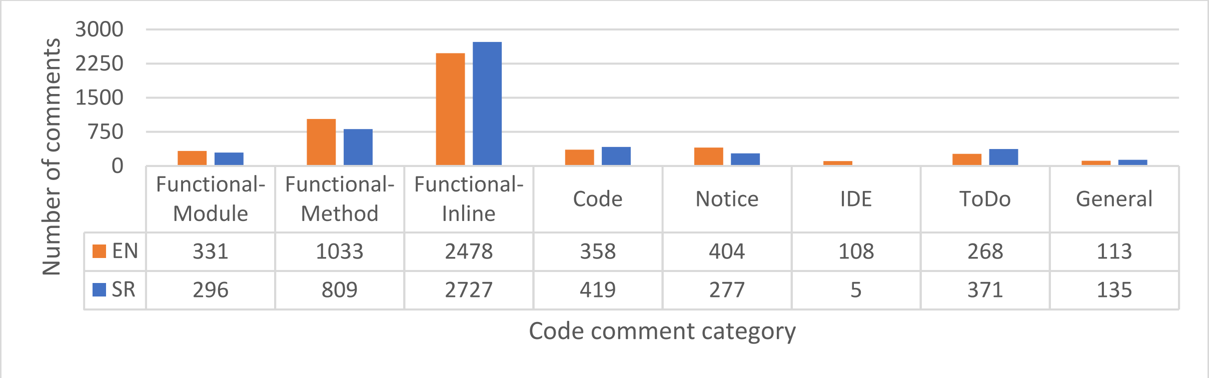
Table 2 shows these measures for different classification settings for all comments and comments in each of the two natural languages. The agreement scores are very similar between the Binary and the Reduced class sets, but the scores decrease for the Full class set, particularly for data in Serbian. This indicates that the main source of confusion is identifying the subtypes of *Functional* comments. To confirm this assumption, we look at the confusion matrix shown in Table 3. Indeed, the highest disagreements between the annotators are found for comments that only one annotator has labeled as *Functional-Method* or as *Functional-Inline*. A probable cause of this issue is the lack of context,

[1] https://github.com/clarinsi/reldi-tokeniser.



	C		C++		C#		Java		JS/TS		PHP		Python		SQL	
	EN	SR	EN	SR	EN	SR	EN	SR	EN	SR	EN	SR	EN	SR	EN	SR
Total	641	714	634	738	654	690	642	741	623	714	627	0	651	713	621	729
🟫 Functional-Module	6	76	40	16	51	23	33	51	12	61	115	0	64	8	10	61
■ Functional-Method	127	41	143	152	119	85	127	202	121	259	227	0	165	59	4	11
🟩 Functional-Inline	350	492	226	401	377	463	358	426	351	342	158	0	323	318	335	285
🟦 Code	64	30	123	26	36	33	57	16	27	14	1	0	21	238	29	62
🟨 Notice	19	17	45	70	48	52	34	18	70	22	49	0	48	46	91	52
⬜ IDE	0	0	0	0	1	2	0	2	8	0	8	0	11	0	80	1
🟧 ToDo	65	16	45	50	16	17	14	5	22	16	49	0	11	33	46	234
🟦 General	10	42	12	23	6	15	19	21	12	0	20	0	8	11	26	23

Fig. 2. The distribution of comments across code comment categories, natural languages, and programming languages.



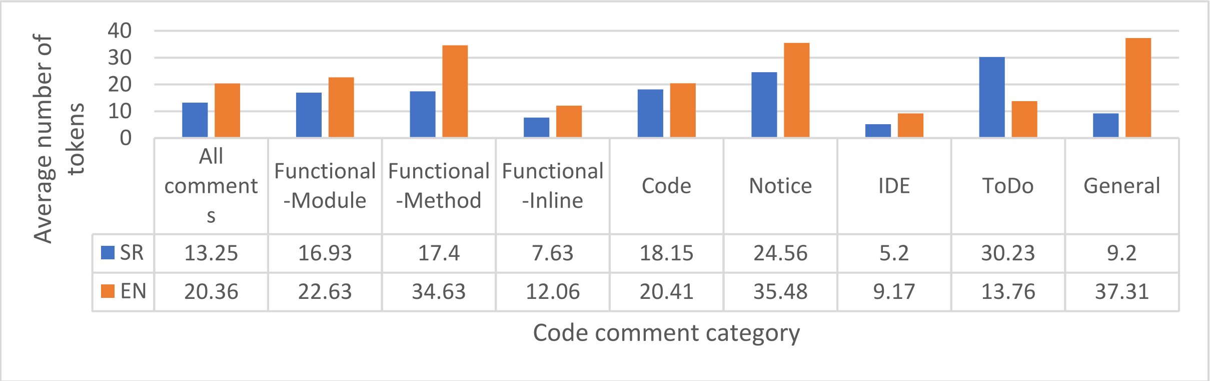
	Functional-Module	Functional-Method	Functional-Inline	Code	Notice	IDE	ToDo	General
🟧 EN	331	1033	2478	358	404	108	268	113
🟦 SR	296	809	2727	419	277	5	371	135

Fig. 3. The distribution of comments across code comment categories and natural languages.

i.e., surrounding source code for some of the annotated comments. Without such contextual information, it is often difficult to confidently determine the location of a functional comment and whether it pertains to a single line of code, an entire method, or even a class. Nevertheless, even when all subcategories are considered, the agreement levels are consistently over 0.8 in terms of Krippendorff's alpha coefficient, indicating that the agreement is reliable (Krippendorff, 2004; Artstein and Poesio, 2008). The only more prominent disagreement between the annotators was the differentiation between the *Functional-Inline* and *Notice* categories, as some comments can be regarded as either functional or as a warning (e.g., `The requested capture operation is not supported.`). There are no significant differences in agreement scores across natural languages for the Binary and the Reduced class sets.

Table 2
Inter-annotator agreement scores.

Setting	ALL		EN		SR	
	Kappa	Alpha	Kappa	Alpha	Kappa	Alpha
Full	0.843	0.843	0.866	0.866	0.815	0.814
Reduced	0.897	0.897	0.902	0.903	0.892	0.892
Binary	0.902	0.902	0.903	0.903	0.901	0.901



	All comments	Functional-Module	Functional-Method	Functional-Inline	Code	Notice	IDE	ToDo	General
SR	13.25	16.93	17.4	7.63	18.15	24.56	5.2	30.23	9.2
EN	20.36	22.63	34.63	12.06	20.41	35.48	9.17	13.76	37.31

Fig. 4. Average comment length in tokens across code comment categories and natural languages.

Table 3
Inter-annotator confusion matrix.

	Functional-Module	Functional-Method	Functional-Inline	Code	Notice	IDE	ToDo	General	Total
Functional-Module	**85**	7	20	0	7	0	1	0	120
Functional-Method	1	**302**	78	0	0	0	0	0	381
Functional-Inline	5	17	**1008**	4	22	0	0	1	1057
Code	0	1	4	**151**	0	0	0	0	156
Notice	1	1	17	0	**107**	0	2	0	128
IDE	0	4	4	0	0	**12**	0	2	22
ToDo	0	0	5	0	7	0	**109**	0	121
General	0	0	0	0	0	1	0	**54**	55
Total	92	332	1136	155	143	13	112	57	**2040**

5. Model evaluation

We evaluate multiple supervised statistical models on the task of code comment classification in three settings: monolingual, multilingual, and cross-lingual. In the monolingual setting, datasets from the two natural languages are used separately. In the multilingual setting, models are trained with comments written in both natural languages but tested only on comments from one natural language at a time. Lastly, in the cross-lingual setting, models are trained using comments written only in one natural language and tested on comments written in the other natural language.

5.1. Evaluation setup

The models in all experiments are trained and tested using the annotated data described in the previous sections. For the dataset's initial cleanup, we replace whitespace character sequences with a single space character. Sequences of repeating characters (three or more) are also replaced with a single occurrence of that character. The aim is to remove decorative sequences of dashes, asterisks, backslashes, and similar characters.

Evaluations in the monolingual setting are performed using 10-fold stratified cross-validation. The same method is used in the multilingual setting — train folds have comments in Serbian and English, while test folds always consist only of comments in a single natural language. On the other hand, in the cross-lingual setting, there is no need for cross-validation since the models are trained on data in one natural language and tested on data in the other. To avoid bias towards the larger classes in the dataset, the macro-averaged f1 score is utilized as the performance metric.

As a baseline for monolingual classification, we use traditional machine learning algorithms, such as multinomial Naïve Bayes (MNB), Logistic Regression (LOG), and Support Vector Machine (SVM). These models are applied to two different approaches for comment vectorization: bag-of-words (BOW) and bag-of-embeddings (BOE). A nested 10-fold stratified cross-validation is used to optimize their hyperparameters: C ∈ {0.001, 0.01, 0.1, 1, 10} for L2-regularized Logistic Regression and linear Support Vector Machine, and alpha ∈ {0.001, 0.01, 0.1, 1, 10} for multinomial Naïve Bayes. We utilize the scikit-learn implementations of these classifiers (Pedregosa et al., 2011), opting for the L-BFGS-B solver for Logistic Regression and the liblinear solver for the Support Vector Machine.

We also evaluate several state-of-the-art pre-trained monolingual and multilingual neural language models, based on Transformer architectures, after fine-tuning them on the presented dataset. The monolingual models included in the evaluation are ELECTRA (Efficiently Learning an Encoder that Classifies Token Replacements Accurately) (Clark et al., 2020) for comments in English, and BERTić (Ljubešić and Lauc, 2021) for comments in Serbian. Monolingual models are used only in the monolingual setting, while multilingual models are evaluated in the multilingual and cross-lingual settings. The multilingual models included in the evaluation are BERT (Bidirectional Encoder Representations from Transformers) multilingual base model (Devlin et al., 2018) and XLM-RoBERTa (Conneau et al., 2019) since they both support not only English but also Serbian. All the models have 110 million parameters and 12 transformer layers and attention heads. We employ the Simple Transformers library[2] to interface with the HuggingFace Transformers implementations (Wolf et al., 2020). The presented results for all transformer models are those obtained as a mean of five runs with different random seed values.

For all settings and models, the same three variants of the code comment classification task described in the previous section are considered: Binary classification into functional and non-functional comments, Full class set classification with all subcategories for both functional and non-functional comments, and a Reduced class set variant with all non-functional subcategories and one category that encompasses all functional comments.

5.2. Monolingual classification

In the monolingual setting, comments in English are separated from the ones in Serbian and hence viewed as distinct sets of data.

[2] https://simpletransformers.ai.

5.2.1. Bag-of-words

The evaluation starts by using bag-of-words (BOW) features. Here, a text is represented as an unordered set of words. In this context, the following preprocessing techniques are examined: different tokenization approaches (whitespace tokenization, scikit-learn tokenizer, NLTK tokenizer for English, ReLDI tokenizer for Serbian[3]), removing digits, removing punctuation, lowercasing, splitting tokens based on camel-casing (e.g., "HashMap" becomes "Hash Map") and snake-casing (e.g., "hash_map" becomes "hash map"), stemming (Porter stemmer for English (Porter, 1980), stemmer of Ljubešić and Pandžić for Serbian[4].), and lemmatization (WordNet for English,[5] ReLDI lemmatizer for Serbian (Ljubešić and Dobrovoljc, 2019)). The extension of the feature set with n-grams is also explored — sequences of two (bigrams) or three (trigram) consecutive tokens, as well as TF (term frequency) and IDF (inverse document frequency) weighting, which aims to reflect how important the word is. The preprocessing technique selection is combined with the selection of the best n-gram and TF/IDF weighting settings. Although our aim is to select the optimal configuration of all these options, it is not viable to try out all the combinations. Hence, each preprocessing technique is considered and enabled if it improves the performance of all classifiers in all settings.

The results for comments in the English language are presented in Table 4. The first step is to pick the best tokenizer. Among the three, the NLTK tokenizer is by far the best choice, which is unsurprising, since it is the only one that is not based on simple regular expressions. It is therefore included in the evaluation of subsequent techniques. The scores in the table represent the difference compared to the baseline. Splitting tokens based on camel or snake casing occasionally improves the classification results, but most often lowers them. The same is true for token lowercasing or using lemmatization and stemming. Furthermore, punctuation tokens are important for classification quality — when removed, scores decrease by more than 5% in almost all settings. On the other hand, the addition of bigram features consistently improves the scores, so it is added to the list of selected techniques. The extension of the feature set to bigrams (and trigrams) is of particular use for multinomial Naïve Bayes. TF weighting is most helpful for the SVM classifier and is usually superior to TFIDF. Therefore, the optimal preprocessing techniques are NLTK tokenizer, bigrams, and TF weighting.

For the optimal set of preprocessing techniques, we looked at the specific words that the models picked up as the most discriminative. We found that the most informative tokens are those related to operators and command terminators. In addition to those, specific keywords are quite important (e.g., `TODO`, `FIXME`, `copyright`, `return`, `param`, `author`, `include`, `https`, `TABLE`, `public` etc.).

The results for the Serbian language are presented in Table 5. The usefulness of particular techniques is very similar to the previous English language analysis. Again, the best tokenizer is the one not based on simple regex — ReLDI. In contrast, the splitting of tokens based on camel or snake case, token lowercasing, lemmatization, and stemming do not consistently help. Removing punctuation again causes a significant drop in the performance scores.

The main difference compared to English happens when looking at bigrams and trigrams. Although these techniques again improve the results of MNB models, for the others, the benefits of higher-order n-grams decrease with the increase in the number of categories. Unlike with English, this actually results in a performance reduction when compared to the base modes. This is the reason why neither bigrams nor trigrams are selected for Serbian. The cause of this discrepancy between the natural languages likely lies in the greater morphological complexity of Serbian, leading to greater data sparsity, which, combined with higher-order n-grams, can easily lead to overfitting. Conversely, TF weighting again proves useful, particularly for the SVM, and superior to TFIDF. Therefore, the final list of techniques is ReLDI tokenizer and TF weighting.

We again looked at the most informative tokens for the best set of preprocessing techniques. The results are similar to the results for English. Operators, command terminators and keywords such as `TODO`, `@param`, `@author` are important. Some discriminative words in Serbian are: `korak`, `klasa`, `funkcija`, `upit`, `metoda` (engl. `step`, `class`, `function`, `query`, `method`).

Fig. 5 concludes the analysis of the behavior of monolingual baseline models with BOW features. It contains an overview of all the best scores previously presented, with the SVM being consistently the best classifier among the three. The scores across the two natural languages look very similar.

5.2.2. Bag-of-embeddings

The second part of the monolingual evaluation is based on word embeddings. Each comment is represented by the mean of the *fastText*[6] embeddings of its tokens. The embeddings for English are taken from the two datasets: *wiki_news* — one million word vectors trained on Wikipedia 2017, UMBC web base corpus and statmt.org news dataset, and *crawl* — two million word vectors trained on Common Crawl (Mikolov et al., 2018). Both datasets have embeddings of size 300. For the Serbian language, the embeddings trained on Common Crawl and Wikipedia (*cc_sr*) (Grave et al., 2018) are used, as well as those trained on the Serbian srWaC web corpus (*embed_sr*) (Ljubešić, 2018). Here only the SVM classifier is applied since it showed the best results in the previous setting. We use the same approach as for the BOW features and explore the same preprocessing techniques. The only difference is that stemming is not considered here because stemmed tokens are not available in the pre-trained sets of embeddings.

The results for the comments in English are presented in Table 6. Again, the best tokenizer is the one from the NLTK package. However, no other examined technique brings any improvement to the f1 score. The biggest drop in the scores happens when punctuation is removed, which means that punctuation characters and their embeddings represent a significant classification signal. *Crawl* embeddings consistently give better results, which is expected as that set of embeddings was created using a bigger training set. Moreover, when using *crawl* embeddings, there are fewer tokens without an embedding compared to *wiki* embeddings.

Table 7 contains the corresponding results for the comments written in Serbian. Once again, the best tokenizer is the ReLDI one. In contrast to the English language, where lowercasing improves the results only in the case of Binary classification, here it improves them in all variants. Consequentially, lowercasing is added to the list of preprocessing techniques. Again, embeddings for punctuation are important for the quality of the classification. *Cc_sr* embeddings lead to significantly better results since the percentage of tokens without embeddings is much lower than in the case for the *embed_sr* set of vectors.

When comparing scores obtained in the two natural languages (Fig. 6), it can be seen that Binary classification performances are on par, whereas multiclass classification results are noticeably lower in Serbian. One of the causes for this may be that Serbian embeddings are of lower quality than English ones because of the smaller amount of data they were trained on, especially given the greater morphological complexity of the language.

[3] https://github.com/clarinsi/reldi-tokeniser.

[4] http://nlp.ffzg.hr/resources/tools/stemmer-for-croatian

[5] https://wordnet.princeton.edu.

[6] https://fasttext.cc.

Table 4
Monolingual classification for English — bag-of-words scores.

Preprocessing	**Full**			**Reduced**			**Binary**		
	SVM	**LOG**	**MNB**	**SVM**	**LOG**	**MNB**	**SVM**	**LOG**	**MNB**
NLTK tokens	**78.84%**	**78.43%**	**72.89%**	**83.73%**	**82.67%**	**78.46%**	**87.31%**	**87.47%**	**85.46%**
whitespace tokens	73.92%	73.26%	70.15%	78.36%	77.00%	74.15%	81.67%	81.43%	80.25%
scikit-learn tokens	73.21%	73.08%	67.32%	78.17%	78.03%	73.17%	82.25%	82.39%	80.29%
List of selected preprocessing techniques: **NLTK tokens**.									
baseline	78.84%	78.43%	72.89%	83.73%	82.67%	78.46%	87.31%	87.47%	85.46%
camel case	-0.79%	-0.31%	-1.06%	0.05%	-0.07%	-1.29%	-0.10%	-0.34%	-0.88%
snake case	-0.27%	0.07%	-0.23%	-0.36%	-0.15%	-1.30%	0.49%	0.20%	-0.12%
camel & snake case	-0.98%	-0.25%	-1.13%	-0.32%	-0.48%	-2.07%	0.05%	-0.08%	-1.24%
lowercase	-0.41%	-0.05%	-0.32%	0.24%	0.42%	-0.63%	0.03%	0.35%	0.12%
lemmatization	-0.11%	-0.31%	-0.08%	0.51%	-0.08%	-0.05%	0.05%	0.24%	-0.30%
stemming	-0.21%	-1.04%	-0.53%	0.27%	0.51%	-0.82%	0.01%	-0.11%	-0.76%
no numbers	-0.21%	-0.02%	-0.61%	0.05%	0.61%	-0.06%	-0.30%	0.08%	-0.26%
no punctuation	-5.49%	-4.83%	-4.94%	-5.25%	-3.71%	-4.73%	-4.78%	-4.57%	-5.58%
no both	-5.54%	-4.62%	-5.12%	-5.44%	-5.32%	-5.83%	-4.92%	-4.96%	-5.46%
bigrams	**0.68%**	**0.43%**	**3.23%**	**0.14%**	**0.05%**	**2.49%**	**0.76%**	**0.68%**	**0.92%**
trigrams	-0.21%	-0.48%	2.72%	-1.44%	-1.23%	3.44%	0.09%	0.07%	0.73%
List of selected preprocessing techniques: **NLTK tokens**, **bigrams**.									
baseline	79.52%	78.86%	76.12%	83.87%	82.72%	80.95%	88.06%	88.15%	86.37%
TF	**1.97%**	**0.28%**	**0.06%**	**2.51%**	**0.09%**	**0.76%**	**1.31%**	**0.25%**	**1.36%**
TFIDF	1.87%	-0.13%	-1.09%	2.15%	0.52%	-0.11%	1.65%	0.36%	-0.12%
The final list of selected preprocessing techniques: **NLTK tokens**, **bigrams**, **TF**.									
best scores	**81.49%**	**79.13%**	**76.18%**	**86.38%**	**82.81%**	**81.71%**	**89.37%**	**88.41%**	**87.73%**

Table 5
Monolingual classification for Serbian — bag-of-words scores.

List of selected preprocessing techniques: **ReLDI tokens**.									
Baseline	79.31%	79.13%	74.20%	82.27%	81.58%	79.01%	89.22%	89.12%	87.26%
camel case	0.16%	-0.11%	-0.33%	0.15%	0.02%	-0.13%	-0.04%	-0.05%	-0.07%
snake case	-0.02%	0.04%	-0.02%	-0.15%	0.19%	-0.33%	0.19%	-0.10%	-0.32%
camel & snake case	-0.39%	-0.13%	-0.33%	-0.14%	0.00%	-0.32%	0.18%	0.03%	-0.26%
lowercase	-0.74%	-0.52%	-1.02%	-0.61%	0.34%	-0.09%	-0.06%	-0.02%	-0.04%
lemmatization	-1.75%	-2.03%	-2.52%	-0.46%	0.00%	-0.63%	-0.43%	-0.36%	-0.80%
stemming	-1.33%	-1.70%	-1.74%	-0.90%	-0.17%	-0.85%	-0.46%	-0.45%	-0.91%
no numbers	-0.10%	-0.21%	-0.41%	-1.02%	0.12%	-0.23%	-0.11%	0.11%	-0.15%
no punctuation	-2.44%	-2.92%	-1.43%	-4.73%	-4.20%	-1.46%	-3.85%	-3.97%	-1.67%
no both	-2.70%	-2.49%	-1.53%	-4.60%	-3.54%	-1.58%	-3.83%	-3.81%	-2.65%
bigrams	-0.19%	-0.31%	1.28%	-0.13%	-0.57%	1.19%	0.50%	0.42%	1.43%
trigrams	-0.94%	-1.55%	1.64%	-0.70%	-1.87%	0.82%	0.01%	-0.10%	1.73%
TF	**1.67%**	**0.60%**	**1.01%**	**1.77%**	**1.02%**	**0.78%**	**0.42%**	**0.14%**	**0.51%**
TFIDF	1.53%	0.61%	-0.74%	1.81%	0.93%	0.43%	0.97%	0.51%	0.39%
The final list of selected preprocessing techniques: **ReLDI tokens**, **TF**.									
Best scores	**80.97%**	**79.74%**	**75.21%**	**84.04%**	**82.60%**	**79.79%**	**89.64%**	**89.26%**	**87.77%**

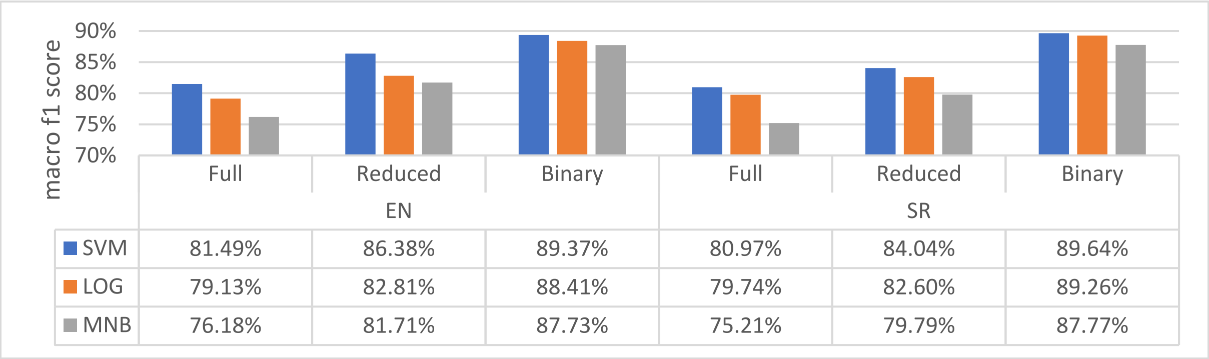


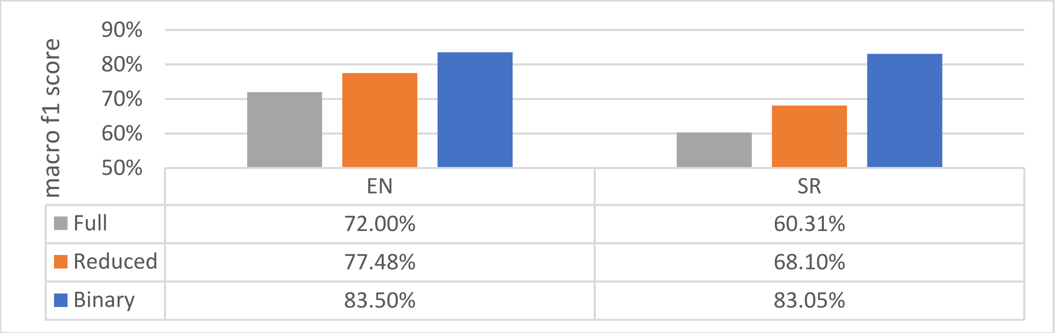
Fig. 5. Monolingual classification — best bag-of-words scores across natural languages.

Table 6
Monolingual classification for English — bag-of-embeddings scores.

Preprocessing	**Full**		**Reduced**		**Binary**	
	wiki news	**crawl**	**wiki news**	**crawl**	**wiki news**	**crawl**
NLTK tokens	**70.26%**	**72.00%**	**76.79%**	**77.48%**	**83.17%**	**83.50%**
whitespace tokens	58.21%	58.94%	63.94%	63.99%	73.24%	75.85%
scikit-learn tokens	63.08%	64.66%	68.25%	70.59%	75.51%	77.21%
List of selected preprocessing techniques: **NLTK tokens**.						
baseline	70.26%	72.00%	76.79%	77.48%	83.17%	83.50%
camel case	-1.99%	-2.91%	-3.63%	-2.23%	-0.96%	-0.39%
snake case	-1.18%	-1.89%	-2.22%	-1.21%	-0.83%	-0.55%
camel & snake case	-2.92%	-3.72%	-5.62%	-3.33%	-2.06%	-1.34%
lowercase	-0.22%	-1.67%	-0.55%	-0.34%	0.76%	1.05%
lemmatization	0.36%	-0.24%	-0.74%	0.54%	0.00%	0.19%
no numbers	-0.41%	-1.47%	-1.11%	-1.19%	-0.30%	-0.07%
no punctuation	-6.52%	-6.95%	-8.78%	-6.52%	-6.62%	-6.10%
no both	-7.01%	-6.93%	-8.33%	-6.72%	-7.59%	-7.29%
The final list of selected preprocessing techniques: **NLTK tokens**.						
best scores	**70.26%**	**72.00%**	**76.79%**	**77.48%**	**83.17%**	**83.50%**

Table 7
Monolingual classification for Serbian — bag-of-embeddings scores.

Preprocessing	**Full**		**Reduced**		**Binary**	
	embed_sr	**cc_sr**	**embed_sr**	**cc_sr**	**embed_sr**	**cc_sr**
ReLDI tokens	**44.35%**	**58.43%**	**58.57%**	**65.94%**	**78.50%**	**82.93%**
whitespace tokens	38.70%	53.96%	47.76%	56.88%	64.25%	72.21%
scikit-learn tokens	37.26%	55.31%	49.94%	61.08%	72.20%	76.42%
List of selected preprocessing techniques: **ReLDI tokens**.						
baseline	44.35%	58.43%	58.57%	65.94%	78.50%	82.93%
camel case	-0,73%	-0,61%	0,68%	0,31%	-0,10%	-0,76%
snake case	-1,26%	0,07%	-0,86%	-0,53%	-0,65%	-0,94%
camel & snake case	-1,46%	-0,48%	-1,08%	-0,67%	-0,64%	-1,43%
lowercase	**6,87%**	**1,88%**	**4,54%**	**2,16%**	**2,55%**	**0,12%**
List of selected preprocessing techniques: **ReLDI tokens**, **lowercase**.						
baseline	51.22%	60.31%	63.11%	68.10%	81.05%	83.05%
lemmatization	-1,01%	-0,17%	-0,86%	-1,00%	-0,20%	0,19%
no numbers	-0,90%	0,09%	-0,90%	-0,73%	-0,17%	-0,04%
no punctuation	-3,81%	-1,72%	-5,49%	-3,75%	-4,69%	-4,43%
no both	-4,15%	-1,48%	-5,67%	-2,84%	-4,42%	-4,28%
best scores	**51.22%**	**60.31%**	**63.11%**	**68.10%**	**81.05%**	**83.05%**
The final list of selected preprocessing techniques: **ReLDI tokens**, **lower**.						



	EN	SR
Full	72.00%	60.31%
Reduced	77.48%	68.10%
Binary	83.50%	83.05%

Fig. 6. Monolingual classification — best bag-of-embeddings scores for both natural languages.

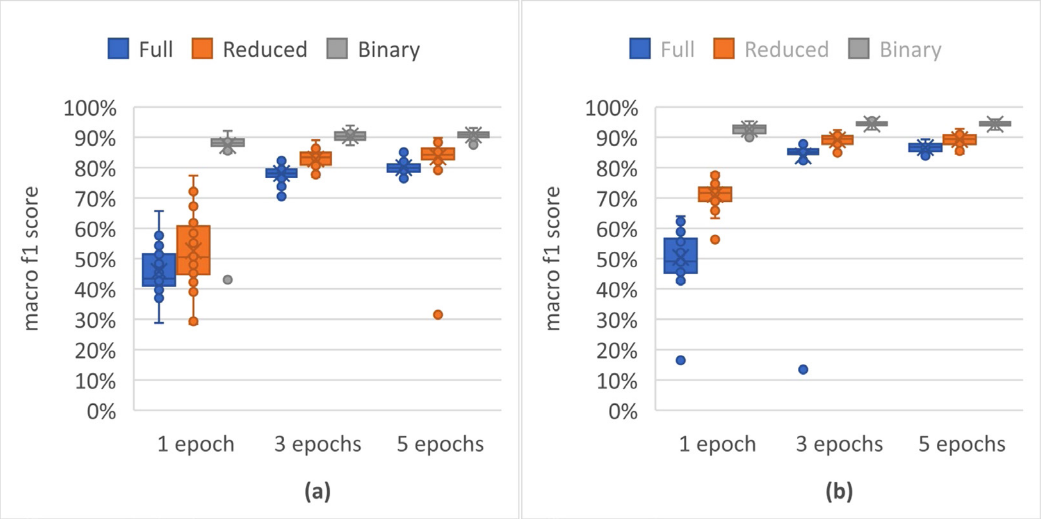


Fig. 7. Monolingual classification for English — ELECTRA (a) and Serbian — BERTić (b).

5.2.3. Transformer models

The final part of the monolingual evaluation focuses on the state-of-the-art neural language models based on Transformer architectures — ELECTRA (Clark et al., 2020) for English and BERTić (Ljubešić and Lauc, 2021) for Serbian. For the monolingual classification, we split the dataset based on the language. The models are fine-tuned for one, three, and five epochs. For each classification class set, there are 50 data points (5 different seeds * 10 cross-validation folds) that are represented via the box and whisker plots.

Fig. 7 shows the results of monolingual classification for both natural languages. The scores improve by increasing the number of fine-tuning epochs, as expected. The improvement is not drastic for Binary classification, unlike for the Full and the Reduced class sets, where the macro f1 score rises by more than 30pp. Additionally, when fine-tuning the models only for one epoch the score variance is quite noticeable, but it becomes less pronounced with longer lengths of fine-tuning.

5.3. Multilingual classification

For multilingual classification, multilingual models — multilingual BERT (Devlin et al., 2018) and XLM-RoBERTa (Conneau et al., 2019) are fine-tuned using comments written in both languages. To compare the results in the two natural languages, evaluation is first performed using comments only in English, and then only those in Serbian.

The two models give very different results for classifying English comments (Fig. 8). Multilingual BERT produces excellent results even after fine-tuning for only one epoch. Also, the spread of the scores is very low. On the other hand, the scores of XLM-RoBERTa show a significant variance. To produce good and consistent results with this model, it is necessary to fine-tune it for more than one epoch. Of course, for both models, the results improve as the fine-tuning length increases, though the differences between three and five epochs are slight for multilingual BERT.

Classification of comments in Serbian in a multilingual setting (Fig. 9) produces excellent results in the Binary classification. Fine-tuning for more than one epoch does not significantly improve the results of Multilingual BERT (same as English), but it does help XLM-RoBERTa. The score spread is very similar for Multilingual BERT regardless of the number of epochs, whereas for XLM-RoBERTa it decreases with longer fine-tuning.

5.4. Cross-lingual classification

For cross-lingual classification, the same multilingual pre-trained models are fine-tuned on comments in one language and tested on comments in the other. Due to this, cross-validation is not used, so for each setting, there are only 5 scores, one per seed value. Nevertheless, the results are presented in the same manner as the previously shown ones for consistency.

Fig. 10 shows the test results on comments written in English. Multilingual BERT's results improve as the fine-tuning length is extended, but not significantly. That is not the case for XLM-RoBERTa, which produces dramatically higher scores in the non-binary classification when fine-tuned for more than one epoch. However, neither model performs well on the Reduced and the Full class set classification, having a macro f1 score of around 50%.

6. Discussion and analysis

This section compares this work to the papers described in Section 2. Additionally, the best results achieved in each classification setting and classification class set variant are discussed. Last part of this section considers Full class set classification performance of best-performing models across different programming languages.

Scores for the Serbian comments look similar (Fig. 11). Multilingual BERT again performs rather consistently regardless of the number of epochs. In contrast, XLM-RoBERTa needs to be fine-tuned for more than one epoch to give somewhat good scores for the Reduced and the Full class set classification.

6.1. Comparison to related work

In this subsection, we present a comparison between methods, previously described in Section 2, and our own work. We are the first to address the problem of code comment classification in multilingual and cross-lingual settings. All previous studies used traditional machine learning models (Decision Tree, Random Forest, multinomial Naïve Bayes) or earlier neural network architectures (LSTM, Convolutional Neural Networks) in a monolingual setting. None of them used state-of-the-art transformer-based neural language models. Annotations from previous papers were mostly done on small sets of code comments written in English and taken from one or two programming languages. In this research, we have compiled a new dataset of around 10,000 comments written not only in English, but also in Serbian, and these comments were taken from eight programming languages.

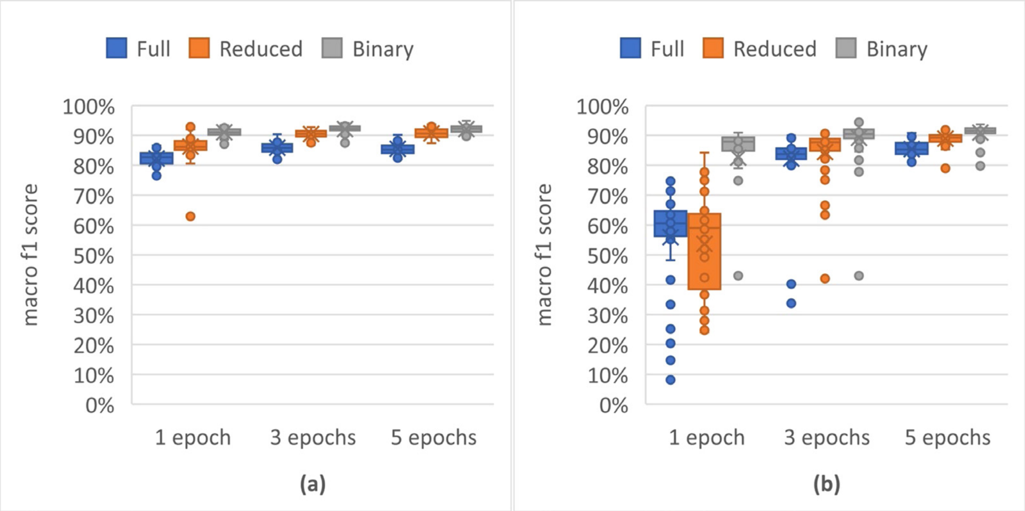


Fig. 8. Multilingual classification for English — Multilingual BERT (a) and XLM-RoBERTa (b).

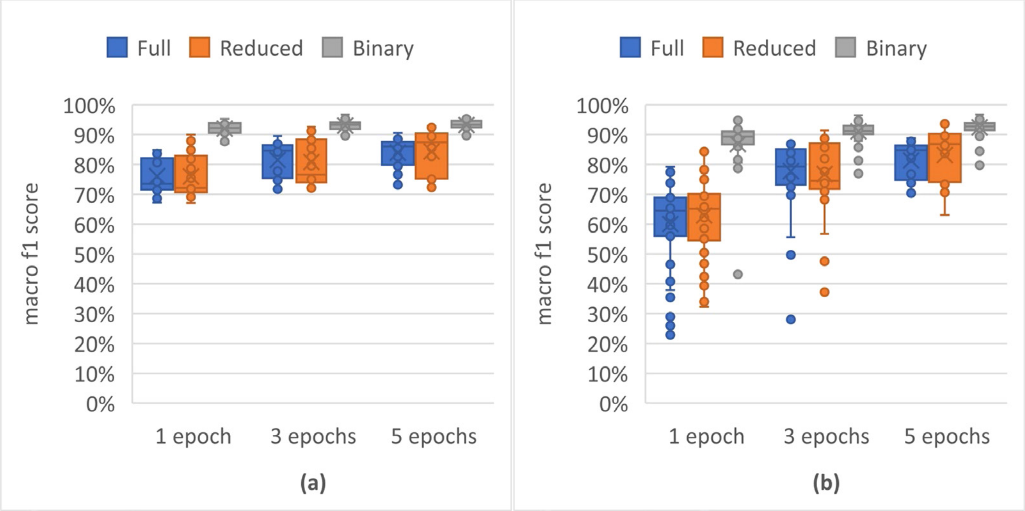


Fig. 9. Multilingual classification for Serbian — Multilingual BERT (a) and XLM-RoBERTa (b).

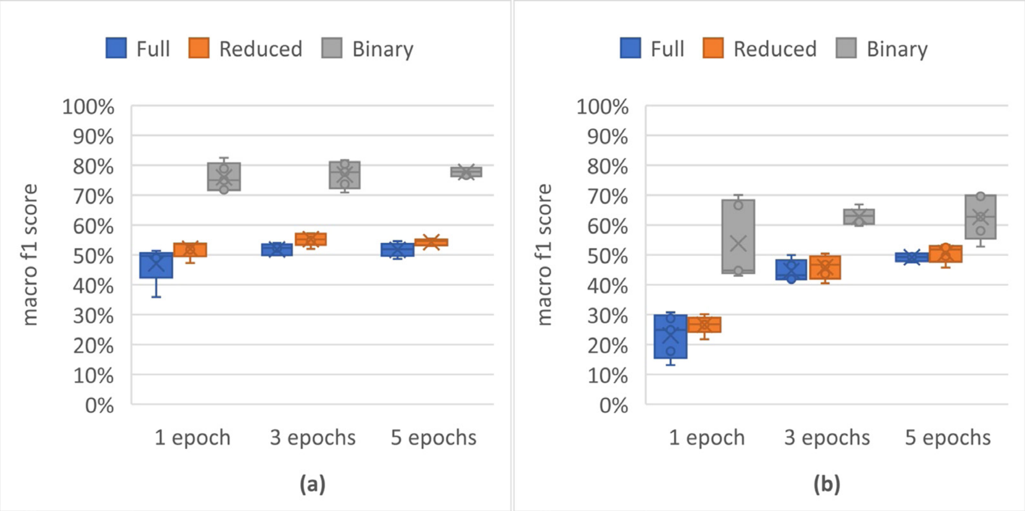


Fig. 10. Cross-lingual classification for English — Multilingual BERT (a) and XLM-RoBERTa (b).

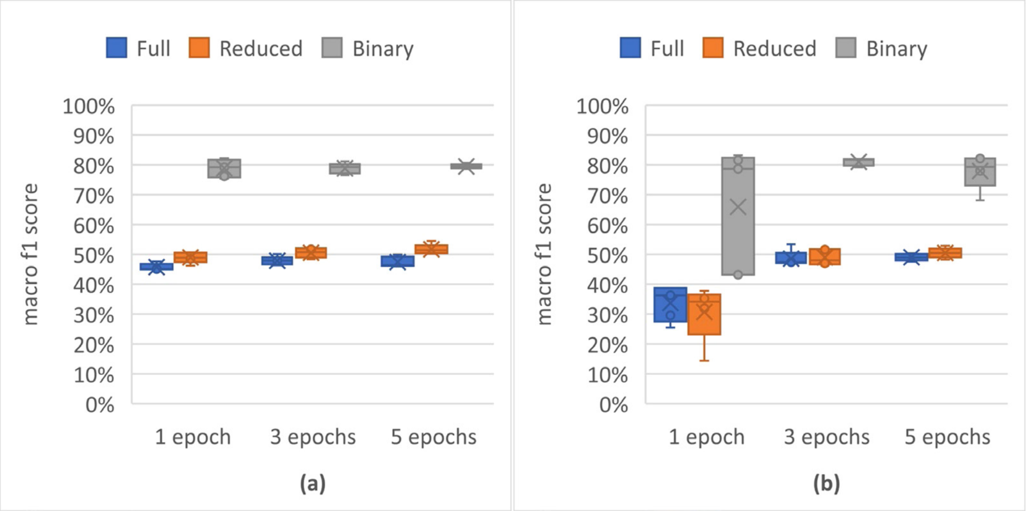


Fig. 11. Cross-lingual classification for Serbian — Multilingual BERT (a) and XLM-RoBERTa (b).

No previously proposed code comment classification taxonomies were created for the tasks of semantic code search and cross-level semantic similarity. In addition, some of them observed more than one perspective, which does not apply to these tasks (Zhai et al., 2020; Padioleau et al., 2009; Haouari et al., 2011). Previous taxonomies are either too complex regarding the number of different categories (Shinyama et al., 2018; Zhang et al., 2018) or they do not consider the position of functional comments (Chen et al., 2021) which is important for the tasks of interest. For these reasons, the taxonomy we propose in this paper builds upon the work of Steidl et al. (2013) and Pascarella et al. (2019).

Unfortunately, it is hard to compare annotation agreements since agreement measures and the number of annotators differ from paper to paper. Comparing model performances across different proposed methods is equally difficult since no evaluation metrics are universally applied. Moreover, the presented evaluation scores are typically calculated solely on proprietary datasets which are not publicly available, making it impossible to verify the original findings and contrast them to the performances of other models on the same data.

6.2. Discussion about our evaluation results

A comparison between the best results we achieved in each classification setting and class set variant is shown in Fig. 12 for English and Fig. 13 for Serbian. Unsurprisingly, the classifier error rates tend to increase as the number of classes expands from Binary classification to the Full class set. It is evident that the simple bag-of-words method, coupled with linear classifiers, still presents a rather strong baseline across all settings and languages, with f1 scores of only ∼2pp–5pp below the best-performing model. This baseline proves to be superior even to the monolingual ELECTRA model on English data in all classification variants except the Binary one. This performance differential is likely due, at least in part, to the higher variance of the neural model. On the other hand, the bag-of-embeddings systems tend to behave rather poorly in comparison and are consistently worse than all other non-cross-lingual options. A probable cause is the relatively high percentage of tokens for which no embeddings could be found in the utilized embedding sets, ranging from 9.5 to 10.5%. However, the drop in performance is more significant for the data in Serbian. The reason for this may be the greater morphological complexity of the Serbian language which makes it harder to create quality embeddings.

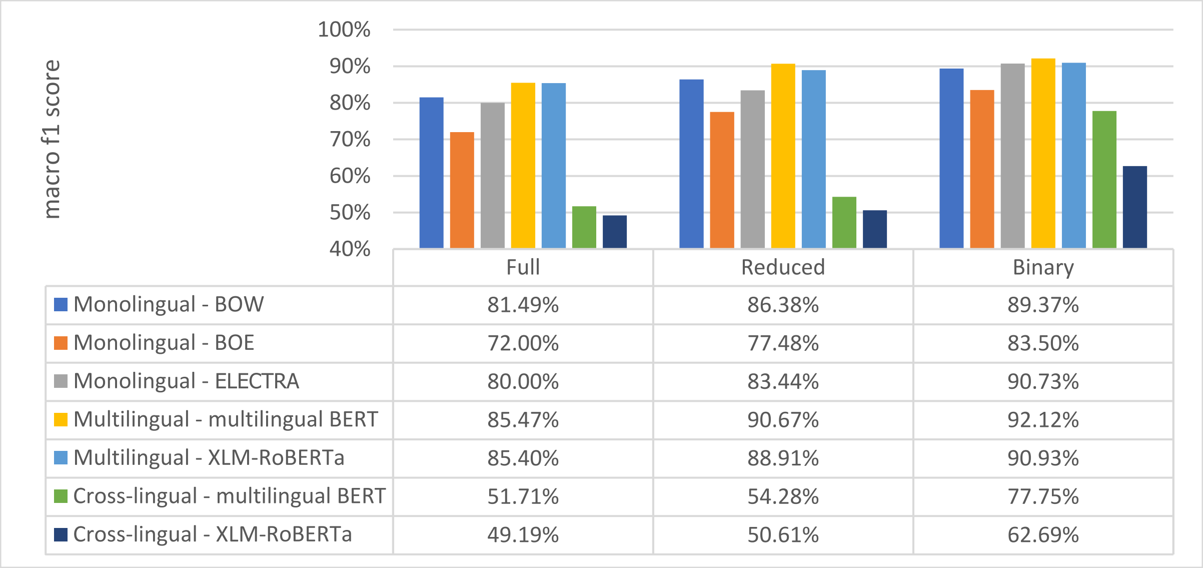
Moving on to transformer-based models, the multilingual variants are superior to monolingual ones in English. This demonstrates the usefulness of multilingual fine-tuning, with multilingual models extracting useful patterns from data in both languages, rather than being limited only to English. This is particularly clear when comparing ELECTRA with multilingual BERT, which have been pre-trained on the same datasets in English.

For Serbian comments, the reverse holds true, with monolingual BERTić outperforming its multilingual counterparts. At first glance, this finding may seem counter-intuitive, particularly since Serbian comments often contain some terms in English and programming keywords that are, by definition, in English. For example, the comment `BEGIN i END funkcije idu u paru i nikad ne mogu ici disjunktno, odnosno jedno bez drugog!` (engl. `functions BEGIN and END go in pair and can never be disjunct, that is, one cannot go without the other!`) is correctly classified as *Notice* comment by BERTić, whereas multilingual BERT classifies it as *Functional-Inline*. A likely explanation for this discrepancy can be found upon closer examination of the datasets used to pre-train multilingual BERT and BERTić. The multilingual BERT's training data most relevant to the comment dataset in Serbian consisted of entries taken from Wikipedia in Serbian and closely related languages, such as Croatian, Bosnian, and Serbo-Croatian. There are currently over 416 million words in these four Wikipedias taken together[7], but this number was lower in 2018/2019 when multilingual BERT was trained. On the other hand, BERTić was pre-trained using multiple large datasets in these languages, including several web corpora, totaling around 8400 million words. Such a large training set may have allowed BERTić to model the semantics of texts in Serbian much better than multilingual BERT, leading to the observed performance discrepancies. However, a comprehensive exploration of this issue is left for further work.

In both languages, multilingual BERT outperforms XLM-RoBERTa. This is consistently true in the multilingual setting and almost always true in the cross-lingual setting. With XLM-RoBERTa's propensity towards higher output variance and its need for longer fine-tuning to address the issue, it can be concluded that multilingual BERT is the superior option for the comment categorization task among the two, at least within the range of fine-tuning lengths that were considered. Given that XLM-RoBERTa was pre-trained on a significantly larger dataset than multilingual BERT (Conneau et al., 2019), it might be able to outperform multilingual BERT with sufficiently long fine-tuning. However, we found the computational costs of exploring this to be prohibitive.

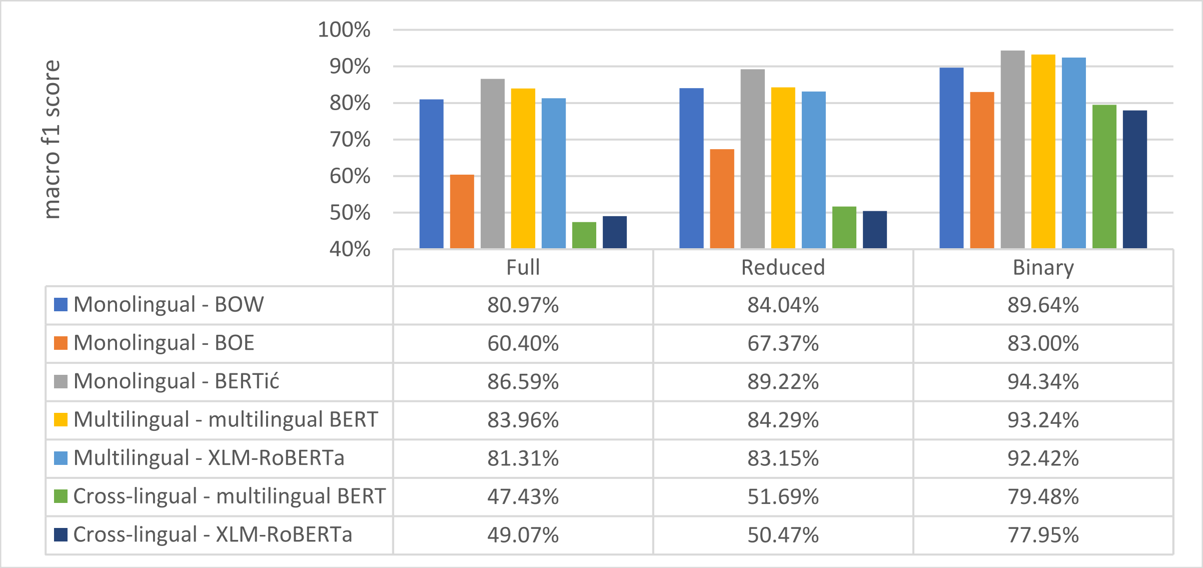
Cross-lingual performances are, as expected, noticeably worse than those obtained either in the monolingual or the multilingual setting.

[7] https://meta.wikimedia.org/wiki/List_of_Wikipedias.



	Full	Reduced	Binary
🟦 Monolingual - BOW	81.49%	86.38%	89.37%
🟧 Monolingual - BOE	72.00%	77.48%	83.50%
⬜ Monolingual - ELECTRA	80.00%	83.44%	90.73%
🟨 Multilingual - multilingual BERT	85.47%	90.67%	92.12%
🟦 Multilingual - XLM-RoBERTa	85.40%	88.91%	90.93%
🟩 Cross-lingual - multilingual BERT	51.71%	54.28%	77.75%
🟦 Cross-lingual - XLM-RoBERTa	49.19%	50.61%	62.69%

Fig. 12. Best scores for different classification settings for English.



	Full	Reduced	Binary
🟦 Monolingual - BOW	80.97%	84.04%	89.64%
🟧 Monolingual - BOE	60.40%	67.37%	83.00%
⬜ Monolingual - BERTić	86.59%	89.22%	94.34%
🟨 Multilingual - multilingual BERT	83.96%	84.29%	93.24%
🟦 Multilingual - XLM-RoBERTa	81.31%	83.15%	92.42%
🟩 Cross-lingual - multilingual BERT	47.43%	51.69%	79.48%
🟦 Cross-lingual - XLM-RoBERTa	49.07%	50.47%	77.95%

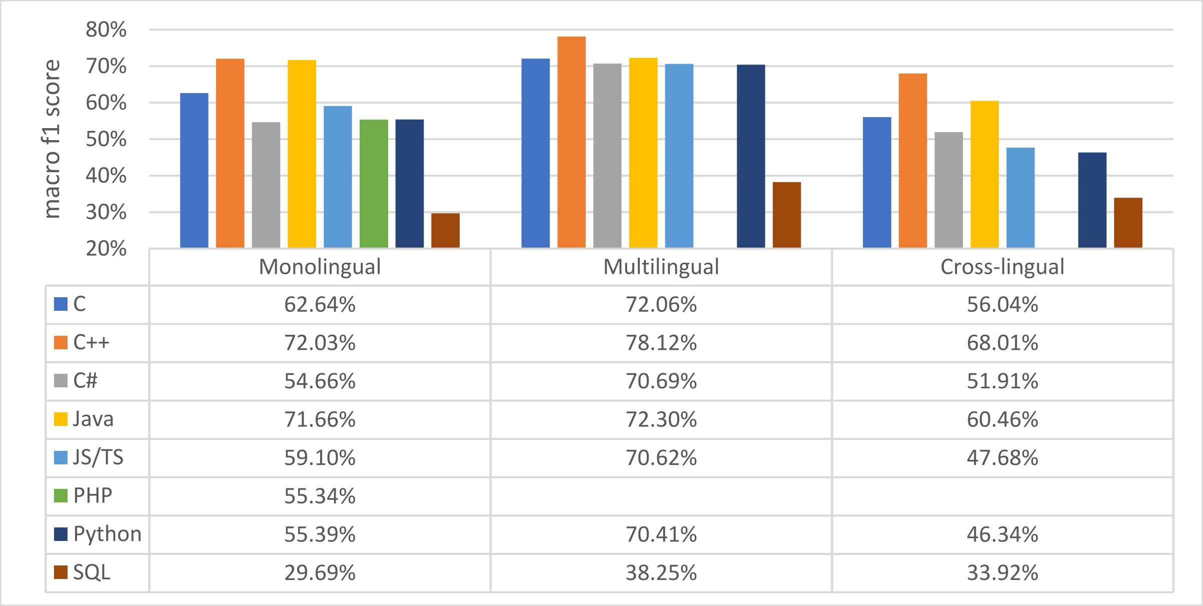
Fig. 13. Best scores for different classification settings for Serbian.

The discrepancy is quite dramatic for the Full and the Reduced class set, while in the Binary class set, it is less pronounced. This is particularly true for the Serbian dataset, where cross-lingual methods almost reach the performance of the bag-of-embeddings baseline. It must be concluded that while cross-lingual models show promising results in Binary classification, they still do not perform well enough to be of practical use.

6.3. Analysis across programming languages

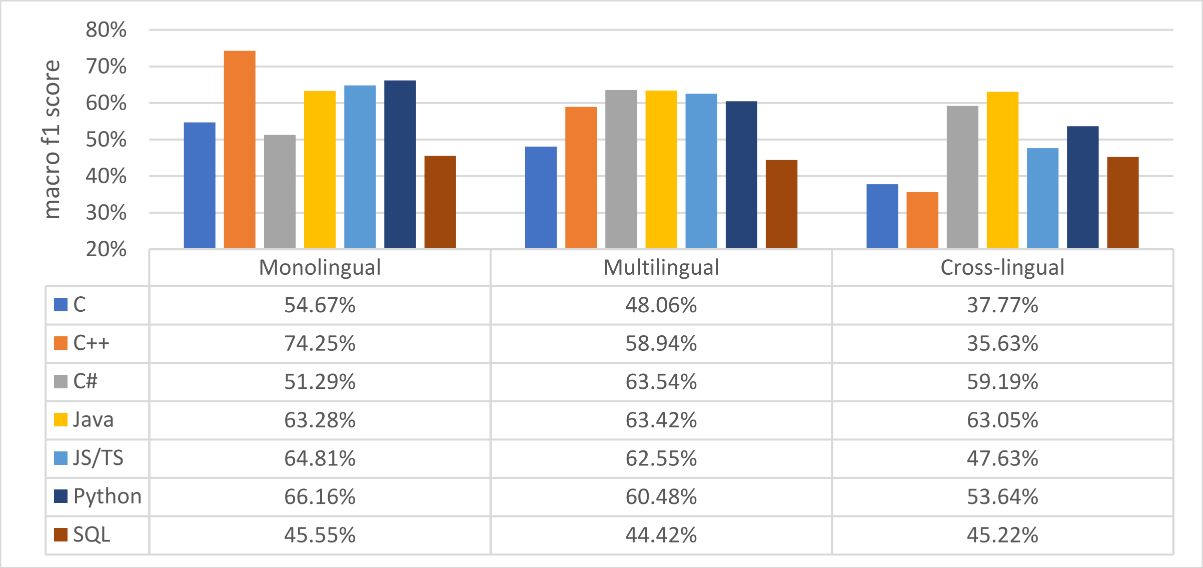
Previous results show model performances across the two natural languages we consider. To analyze the behavior of the best-performing models in more detail, we also look at the transformer performances for Full class set classification across different programming languages. For the evaluation, the test set contains comments from the programming language being evaluated, written in the selected natural language — English or Serbian. The training set in the monolingual setting is comprised of comments from all other programming languages, written in the same natural language. In the cross-lingual setting, the training set consists of all comments (from all programming languages) written in the other natural language. The multilingual training set is a union of mono- and cross-lingual training data. It should be noted that the scores we present here are not directly comparable to those shown in Section 5, since our evaluation method there was to divide data using stratified cross-validation, which does not focus on programming languages but instead ensures same class frequencies between the training and the test data. All models are fine-tuned for five epochs.

Fig. 14 shows the performances of the best transformer models on English comments. In the monolingual setting we evaluated ELECTRA, and in the multilingual and cross-lingual ones we evaluated multilingual BERT. In the multilingual and cross-lingual settings, we omit the results for PHP because we do not have PHP comments



	Monolingual	Multilingual	Cross-lingual
🟦 C	62.64%	72.06%	56.04%
🟧 C++	72.03%	78.12%	68.01%
⬜ C#	54.66%	70.69%	51.91%
🟨 Java	71.66%	72.30%	60.46%
🟦 JS/TS	59.10%	70.62%	47.68%
🟩 PHP	55.34%		
🟦 Python	55.39%	70.41%	46.34%
🟫 SQL	29.69%	38.25%	33.92%

Fig. 14. Performances across programming languages for comments in English and Full class set.



	Monolingual	Multilingual	Cross-lingual
🟦 C	54.67%	48.06%	37.77%
🟧 C++	74.25%	58.94%	35.63%
⬜ C#	51.29%	63.54%	59.19%
🟨 Java	63.28%	63.42%	63.05%
🟦 JS/TS	64.81%	62.55%	47.63%
🟦 Python	66.16%	60.48%	53.64%
🟫 SQL	45.55%	44.42%	45.22%

Fig. 15. Performances across programming languages for comments in Serbian and Full class set.

in Serbian that would be part of the training set. Our results show that the multilingual setting is superior to others in all programming languages, which is consistent with the findings in Section 5. Also, the results for different programming languages are relatively uniform with the exception of SQL. For SQL comments the cross-lingual setting outperforms the monolingual one, in contrast to all other programming languages. SQL is the only non-general purpose programming language, which makes its comments highly distinct from the ones belonging to other programming languages. Due to this, classifiers trained solely on comments from the other programming languages, even though they are in the same natural language (English), are actually trained on data which is less relevant to SQL comments than classifiers in the cross-lingual setting, whose training set includes SQL comments in Serbian.

Classification performances across programming languages for comments written in Serbian are shown in Fig. 15. We evaluated BERTić for the monolingual setting and multilingual BERT for the other two. The best scores are obtained in the monolingual setting for all programming languages except C#, which is also consistent with the findings in the previous section. Again, classification of SQL comments proves to be the most difficult.

For both natural languages, the multilingual setting produces more uniform results across the set of programming languages. This is likely due to the larger training set available in this setting, which minimizes the impact of individual programming languages' specificities. On the other hand, the cross-lingual setting generally produces less uniform results, since here the train/test set differences are the most pronounced.

7. Conclusions

This paper has, for the first time, considered the problem of code comment classification not only in a monolingual setting but also in a multilingual and cross-lingual one. To achieve this, we have created a dataset of code comments in English and Serbian and have annotated it according to a newly developed taxonomy of code comment categories. This dataset, encompassing over 10,000 comments drawn from various programming languages, has been made publicly available, which will enable progress tracking on this task in the future. We believe this resource will be particularly useful for the further development of NLP research in under-resourced languages such as Serbian.

The conducted experiments have shown that linear bag-of-words classifiers remain quite a strong baseline across all settings and both natural languages. However, transformer-based neural language models typically outperform them. For the Serbian data, the monolingual BERTić model proves to be the best option. On the other hand, on the English data, the multilingual BERT model excels. The cross-lingual setting falls behind the monolingual and multilingual settings, though their performances in binary classification show some promise. These findings are confirmed with the analysis of model performances across different programming languages.

The research presented here has two limitations. The first is that the maximum number of training epochs for transformer-based models was kept at 5 due to time considerations. It is possible that some of the evaluated models, particularly XLM-RoBERTa, could perform noticeably better if longer fine-tuning lengths were used. The second research limitation is the relatively small size of the created dataset and the imbalance in the number of samples for each comment category.

In the future, we plan to use the created code comment classifiers as a first step in tackling the task of Cross-Level Semantic Similarity in the code comment domain and for Semantic Code Search. There are also multiple avenues of further research into the code comment classification task. For instance, explicit positional features could be provided to classifiers, instead of relying solely on a comment's textual content. Another important issue would be the automatic treatment of longer comments belonging to multiple classes, a problem not encountered in the presented dataset since such comments were manually divided into multiple units during the annotation process.

CRediT authorship contribution statement

Marija Kostić: Conceptualization, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft. **Vuk Batanović:** Conceptualization, Methodology, Writing – review & editing, Supervision. **Boško Nikolić:** Resources, Writing – review & editing, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The link to the data and code can be found in the manuscript.

Code comment categorization corpus (Original data) (Github)

Acknowledgments

Authors would like to thank Aleksa Srbljanović for participating in the data annotation process, and Vuk Jovanović for developing the Commano annotation tool.

Funding

This work was supported by the Science Fund of the Republic of Serbia [grant number 6526093].

References

Agirre, E., Cer, D., Diab, M., Gonzalez-Agirre, A., 2012. SemEval-2012 task 6: A pilot on semantic textual similarity. In: The First Joint Conference on Lexical and Computational Semantics–Volume 1: Proceedings of the Main Conference and the Shared Task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation.

Artstein, R., Poesio, M., 2008. Inter-coder agreement for computational linguistics. Comput. Linguist. 34 (4), 555–596. http://dx.doi.org/10.1162/coli.07-034-R2.

Bird, S., Klein, E., Loper, E., 2009. Natural Language Processing with Python. O'Reilly Media, Inc..

Blasi, Goffi, A., Kuznetsov, K., Gorla, A., Ernst, M.D., Pezzè, M., Castellanos, S.D., 2018. Translating code comments to procedure specifications. In: Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis. New York, NY, USA, http://dx.doi.org/10.1145/3213846.3213872.

Chen, Q., Xia, X., Hu, H., Lo, D., Li, S., 2021. Why my code summarization model does not work: Code comment improvement with category prediction. ACM Trans. Softw. Eng. Methodol. 30 (2), http://dx.doi.org/10.1145/3434280.

Clark, K., Luong, M.-T., Le, Q.V., Manning, C.D., 2020. ELECTRA: Pre-training text encoders as discriminators rather than generators. http://dx.doi.org/10.48550/arXiv.2003.10555, arXiv.

Cohen, J., 1960. A coefficient of agreement for nominal scales. Educ. Psychol. Meas. 20 (1), 37–46. http://dx.doi.org/10.1177/001316446002000104.

Conneau, Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., Stoyanov, V., 2019. Unsupervised cross-lingual representation learning at scale. http://dx.doi.org/10.48550/arXiv.1911.02116, arXiv.

de Souza, S.C.B., Anquetil, N., de Oliveira, K.M., 2005. A study of the documentation essential to software maintenace. In: Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting & Designing for Pervasive Information. New York, NY, USA, http://dx.doi.org/10.1145/1085313.1085331.

Devlin, J., Chang, M.-W., Lee, K., Toutanova, K., 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. http://dx.doi.org/10.48550/arXiv.1810.04805, arXiv.

Google, 2022a. Google C++ style guide - Comments. [Online]. Available: https://google.github.io/styleguide/cppguide.html#Comments. [Accessed 27 June 2022].

Google, 2022b. Google Python style guide. [Online]. Available: https://google.github.io/styleguide/pyguide.html. [Accessed 27 June 2022].

Grave, E., Bojanowski, P., Gupta, P., Joulin, A., Mikolov, T., 2018. Learning word vectors for 157 languages. In: Proceedings of the Eleventh International Conference on Language Resources and Evaluation.

Gvero, T., Kuncak, V., 2015. Synthesizing java expressions from free-form queries. SIGPLAN Not. 50 (10), 416–432. http://dx.doi.org/10.1145/2858965.2814295.

Haouari, D., Sahraoui, H., Langlais, P., 2011. How good is your comment? A study of comments in Java programs. In: Proceedings of the International Symposium on Empirical Software Engineering and Measurement. http://dx.doi.org/10.1109/ESEM.2011.22.

Hartzman, C.S., Austin, C.F., 1993. Maintenance productivity: Observations based on an experience in a large system environment. In: Proceedings of the 1993 Conference of the Centre for Advanced Studies on Collaborative Research: Software Engineering.

Husain, H., Wu, H.-H., Gazit, T., Allamanis, M., Brockschmidt, M., 2019. CodeSearchNet challenge: Evaluating the state of semantic code search. http://dx.doi.org/10.48550/arXiv.1909.09436, arXiv.

Jurgens, D., Pilehvar, M.T., Navigli, R., 2014. SemEval-2014 task 3: Cross-level semantic similarity. In: Proceedings of the 8th International Workshop on Semantic Evaluation. pp. 17–26. http://dx.doi.org/10.3115/v1/S14-2003.

Jurgens, D., Pilehvar, M.T., Navigli, R., 2016. Cross level semantic similarity: an evaluation framework for universal measures of similarity. Lang. Resour. Eval. 5 (1), 5–33. http://dx.doi.org/10.1007/S10579-015-9318-3.

Kostić, M., Srbljanović, A., Batanović, V., Nikolić, B., 2022. Code comment classification taxonomies. In: Proceedings of the 9th International Conference on Electrical, Electronic and Computing Engineering.

Kramer, D., 1999. API documentation from source code comments: A case study of Javadoc. In: Proceedings of the 17th Annual International Conference on Computer Documentation. New York, NY, USA, http://dx.doi.org/10.1145/318372.318577.

Krippendorff, K., 2004. Content Analysis: An Introduction to its Methodology. Sage publications.

Ljubešić, N., 2018. Word Embeddings CLARIN.SI-Embed.Sr 1.0. Jožef Stefan Institute, http://hdl.handle.net/11356/1206.

Ljubešić, N., Dobrovoljc, K., 2019. What does neural bring? Analysing improvements in morphosyntactic annotation and lemmatisation of Slovenian, Croatian and Serbian. In: Proceedings of the 7th Workshop on Balto-Slavic Natural Language Processing. http://dx.doi.org/10.18653/v1/W19-3704.

Ljubešić, N., Lauc, D., 2021. BERTić - The transformer language model for Bosnian, Croatian, Montenegrin and Serbian. In: Proceedings of the 8th Workshop on Balto-Slavic Natural Language Processing.

Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., Joulin, A., 2018. Advances in pre-training distributed word representations. In: Proceedings of the Eleventh International Conference on Language Resources and Evaluation.

Nie, P., Rai, R., Li, J.J., Khurshid, S., Mooney, R.J., Gligoric, M., 2019. A framework for writing trigger-action todo comments in executable format. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York, NY, USA, http://dx.doi.org/10.1145/3338906.3338965.

Oracle, 2022. How to write doc comments for the Javadoc tool. [Online]. Available: https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html. [Accessed 27 June 2022].

Padioleau, Y., Tan, L., Zhou, Y., 2009. Listening to programmers — Taxonomies and characteristics of comments in operating system code. In: Proceedings of the IEEE 31st International Conference on Software Engineering. http://dx.doi.org/10.1109/ICSE.2009.5070533.

Pandita, R., Xiao, X., Zhong, H., Xie, T., Oney, S., Paradkar, A., 2012. Inferring method specifications from natural language API descriptions. In: Proceedings of the 34th International Conference on Software Engineering. http://dx.doi.org/10.1109/ICSE.2012.6227137.

Pascarella, L., Bruntink, M., Bacchelli, A., 2019. Classifying code comments in Java software systems. Empir. Softw. Eng. 24 (3), 1499–1537. http://dx.doi.org/10.1007/s10664-019-09694-w.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, É., 2011. Scikit-learn: Machine learning in Python. J. Mach. Learn. Res. 12 (85), 2825–2830.

Popović, M., Arcan, M., 2015. Identifying main obstacles for statistical machine translation of morphologically rich South Slavic languages. In: Proceedings of the 18th Annual Conference of the European Association for Machine Translation.

Porter, M., 1980. An algorithm for suffix stripping. Program: Electron. Libr. Inf. Syst. 14 (3), 130–137. http://dx.doi.org/10.1108/eb046814.

Shinyama, Y., Arahori, Y., Gondow, K., 2018. Analyzing code comments to boost program comprehension. In: Proceedings of the 25th Asia-Pacific Software Engineering Conference. http://dx.doi.org/10.1109/APSEC.2018.00047.

Steidl, D., Hummel, B., Juergens, E., 2013. Quality analysis of source code comments. In: Proceedings of the 21st International Conference on Program Comprehension. http://dx.doi.org/10.1109/ICPC.2013.6613836.

Tan, L., Yuan, D., Krishna, G., Zhou, Y., 2007. /*Icomment: bugs or bad comments?*/. In: Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles. New York, NY, USA, http://dx.doi.org/10.1145/1294261.1294276.

Tenny, T., 1985. Procedures and comments vs. the banker's algorithm. SIGCSE Bull. 17 (3), 44–53. http://dx.doi.org/10.1145/382208.382523.

Tenny, T., 1988. Program readability: procedures versus comments. IEEE Trans. Softw. Eng. 14 (9), 1271–1279. http://dx.doi.org/10.1109/32.6171.

Wolf, T., Debut, L., Sahn, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, Teven, Gugger, S., Drame, M., Lhoest, Q., Rush, A., 2020. Transformers: State-of-the-art natural language processing. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. http://dx.doi.org/10.18653/v1/2020.emnlp-demos.6.

Woodfield, S.N., Dunsmore, H.E., Shen, V.Y., 1981. The effect of modularization and comments on program comprehension. In: Proceedings of the 5th International Conference on Software Engineering.

Zhai, J., Huang, J., Ma, S., Zhang, X., Tan, L., Zhao, J., Qin, F., 2016. Automatic model generation from documentation for Java API functions. In: Proceedings of the 38th International Conference on Software Engineering. New York, NY, USA, http://dx.doi.org/10.1145/2884781.2884881.

Zhai, J., Xu, X., Shi, Y., Tao, G., Pan, M., Ma, S., Xu, L., Zhang, W., Tan, L., Zhang, X., 2020. CPC: automatically classifying and propagating natural language comments via program analysis. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. New York, NY, USA, http://dx.doi.org/10.1145/3377811.3380427.

Zhang, J., Xu, L., Li, Y., 2018. Classifying Python code comments based on supervised learning. In: Proceedings of the International Conference on Web Information Systems and Applications. http://dx.doi.org/10.1007/978-3-030-02934-0_4.

Zhong, H., Zhang, L., Xie, T., Mei, H., 2009. Inferring resource specifications from natural language API documentation. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering. http://dx.doi.org/10.1109/ASE.2009.94.