Journal of Innovations in Computer Science and Engineering
Shahid Beheshti Unversity

January 2024, Volume 1 Issue 2

A Framework for Evaluating Word Boundary Detection in Persian Tokenizers

Mostafa Karimi Manesh [a], ORCID: 0009-0001-7410-3999
Mehrnoush Shamsfard[1] [b], ORCID: 0000-0002-7027-7529
[a] Faculty of computer science and engineering, Shahid Beheshti University, Tehran, Iran, m_karimimanesh@sbu.ac.ir
[b] Faculty of computer science and engineering, Shahid Beheshti University, Tehran, Iran, m-shams@sbu.ac.ir

ABSTRACT

Tokenization is a critical stage in text preprocessing and presents numerous challenges in languages like Persian, where there is no deterministic word boundary. These challenges include the identification of multi-function morphemes, separation of punctuation marks, dealing with the omission of spaces between tokens, and handling extra spaces inside words.

Typically, the evaluation of tokenizers focuses on comparing their overall performance, and the test data does not necessarily cover all challenging linguistic phenomena. As a result, the strengths and weaknesses of tokenizers in handling these specific challenges are not independently assessed. However, this paper examines the challenges posed by the Persian script in detecting word boundaries in tokenizers and evaluates the performance of eight different tokenizers (six tokenizer tools and two language models) when addressing each of these challenges. For this purpose, a test set consisting of 4091 tokens across 483 sentences has been prepared, of which 1010 tokens have been considered as challenging tokens. The evaluation of the tokenizers with respect to word boundary detection has been conducted based on this dataset.

The results indicate that each tokenizer shows different performance when handling various aspects of Persian orthography. For instance, some tokenizers performed better in separating compound words, while others were more successful in correctly identifying and preserving zero-length-joiner (half-space). A detailed comparison of these tools reveals that none of the tokenizers is entirely capable of addressing all challenges, highlighting the necessity for improving algorithms and developing more intelligent solutions for Persian word boundary detection in tokenization. Introducing a comprehensive benchmark and identifying the strengths and weaknesses of available tokenizers, this study paves the way for the better development of Persian language processing tools.

KEYWORDS

Tokenization, NLP, Evaluation of Tokenizers, Evaluation Benchmark, the Persian language

1. INTRODUCTION

Natural Language Processing (NLP) is a branch of artificial intelligence that enables machines to read, understand, and extract meaning from human language texts. This widely-used technology is applied for various purposes, such as sentiment analysis, generating chatbots and intelligent assistants, text classification, intent detection, machine translation, automatic text correction, and more. To achieve the goals in the field of NLP, certain preprocessing steps, such as normalization, tokenization, stemming, and lemmatization, need to be performed. In this paper, we focus on one of the most fundamental preprocessing tasks, namely tokenization, and examine the behavior of tokenizers in relation to the orthographic challenges of the Persian language.

Tokenization in NLP is the process of breaking down text into smaller units called tokens. These tokens can be words, subwords, numbers, symbols, punctuation marks, or other units found in the text. Tokenization can occur at different levels, from morphemes to compound words.

At the word level, text is segmented into a list of words, which may include common or proper nouns, adjectives, adverbs, verbs, prepositions, etc. in the form of simple or depending on the tokenization policy, compound words. This type of tokenizers which are the main focus of this paper, act as word boundary detectors. On the other hand, some tokenizers further refine this process by working with sub-word tokens, breaking down text into parts of words or individual morphemes. At the morpheme level, each morpheme (including letters, digits, stems, affixes, and punctuation marks) can be treated as a token.

Tokenization methods are generally divided into two main categories: rule-based tokenization and machine learning-based tokenization. In rule-based tokenization, explicit rules and patterns are used to define how text is segmented, often relying on

Submit Date: 2024-09-29
Revise Date: 2024-11-11
Accept Date: 2024-11-26
[1] Corresponding author

deterministic rules to split text into tokens. Rule-based tokenization is commonly used in structured languages like English, where white spaces (spaces, punctuation marks, control characters) serve as reliable word boundaries [1]. Machine Learning-based tokenization, is a more modern and flexible approach, especially useful for handling unknown words. It uses a training corpus to learn how to split the text into tokens.

On the other hand, tokenization algorithms can be divided into top-down and bottom-up approaches. In top-down algorithms, the sentences is segmented into words and sub-words. Space-to-space tokenization is the simplest method of this category which splits texts from the location of white spaces and considers any string of characters between two white spaces as a single token. Such an algorithm can be considered a baseline model, against which other algorithms can be compared. Additionally, according to the features of a language this tokenizer may be enhanced by using some rules to split or concatenate extracted tokens.

In bottom-up approaches tokenizer starts from smaller morphemes (such as letters) and merge them to build tokens. An example of this method is Byte-Pair Encoding (BPE), a popular algorithm that merges frequent pairs of characters to create subwords, allowing for the capture of morphological features (e.g., "low" + "er" = "lower"). This method is particularly beneficial for languages with complex morphology or when dealing with out-of-vocabulary words.

The choice between these methods depends on the complexity of the language and the specific NLP application. For example, in languages like Persian, where spaces are not deterministic word boundaries and morphological complexity is high, simple space-to-space tokenization would not have good results.

In this paper we introduce a benchmark for evaluating the word boundary detection property of Persian tokenizers according to various challenges of this task.

The contributions of this paper are:

- Investigating the Persian challenges in tokenization and word boundary detection and extracting a list of challenges (phenomena) on which the benchmarks can be built
- Introducing a multi criteria benchmark (dataset and evaluation algorithm) which can evaluate tokenizers on word boundary detection task from different points of view (different challenges)
- Presenting an analytical survey on Persian tokenizers and language models.

In the second section, we will review the related work, introducing the prominent Persian tokenizers and discussing previous research on their evaluation. In the third section, we will present the challenges that tokenizers face when dealing with Persian texts. Based on the categorization provided in this section, the fourth section will introduce the evaluation benchmark and the evaluation algorithm used. Subsequently, the fifth section will present the evaluation results and analysis of seven prominent Persian tokenizers based on the designed corpus and algorithm.

2. RELATED WORK

Due to the importance of tokenization in text processing tasks, various studies have been conducted in this field for the Persian language, resulting in the development of several toolboxes such as Stanza [2], Parsivar [3], ParsiNorm [4], Hazm [2], Step1 [5], Verb Farsi [6], Dadmatools [7] and Polyglot [3]. Some of the mentioned toolboxes offer various services including normalization, tokenization, stemming, part-of-speech tagging, dependency parsing, syntactic parsing, chunking, and named entity recognition. In this paper, normalization and tokenization from these tools have been studied.

It should be noted that there are some tools, like the tokenizers of language models such as ParsBERT [8], which use sub-word tokenization with the WordPiece method at the morpheme level. As it was mentioned earlier, the main focus of the paper is to compare the word boundary detection of word level tokenizers, although we use some samples of morpheme level tokenizers in the comparison as well.

In the tokenization process, similarities can be observed among the mentioned toolboxes. For instance, in toolboxes like Hazm, ParsiNorm, DadmaTools, STeP-1 and Stanza, tokenization begins with the normalization of the input text, correction of non-standard characters, adjustment of spacing in punctuation marks, and common spacing issues in the Persian language.

However, different methods are used for identifying word boundaries: for instance, Hazm, DadmaTools, and Parsivar rely on linguistic patterns and regular expressions; ParsiNorm and Polyglot utilize linguistic patterns and training data; Stanza employs an LSTM model; Farsi Verb [6] focuses on verbs using regular expressions and Persian verb patterns.

Even in the same category of methods, the details supported by each tokenizer is different with others. For example, Step-1 performs normalization and correction of spaces through several stages using a computational script. This tool checks the correct spelling of words using a dictionary and adjusts spaces where necessary. It uses a combination of rule-based and dictionary-based methods to tokenize the text into smaller units. First, word boundaries are detected using a space-to-space method. Then, the tokenizer identifies

[2] https://github.com/roshan-research/hazm
[3] https://github.com/EleutherAI/polyglot

multi-part verbs, numbers, dates, abbreviations, and some proper nouns and concatenate different parts of compound verbs and words to form a single token or split the large segments to create smaller tokens and converts the final tokens into a standardized format. This tokenizer utilizes a database containing over 57,000 entries, including nouns, adjectives, verbs, prefixes, and suffixes, to match the recognized words.

In addition to researches which focus on developing tokenizers, various studies, have been conducted to examine challenges, evaluate, and analytically compare existing tools in this field. To evaluate tokenizers, four steps must be undertaken:

1. Identifying the challenges and error-prone areas that the tokenizer needs to address.
2. Preparing a benchmark corpus or dataset for evaluation.
3. Developing the tools and testing environment for evaluation.
4. Presenting and analyzing the results.

Some of the conducted studies are as follows:

Sharifi Atashgah and Bijankhan [9] through their analysis of the Perso-Arabic script and its challenges, have introduced various types of token collocations, including compositional, non-compositional, and new semi-compositional structures. Next, static and dynamic multi-token units are presented to illustrate generative and non-generative structures in the main categories, including verbs, infinitives, prepositions, conjunctions, adverbs, adjectives, and nouns. According to their research, defining multi-token unit templates for these categories is one of the key outcomes of their study. The authors believe that providing solutions for handling multi-token units can have a direct impact on establishing rules for the design and implementation of tokenizers, morphological analyzers, syntactic parsers, and more. According to the four-stage tokenizer evaluation process mentioned earlier, this paper focuses on the first stage, which is the identification of challenges.

Farhoodi and colleagues [10] initially introduced non-simple linguistic structures, which are formed by the combination of various linguistic elements such as words, affixes, and syntactic groups. These structures are categorized into three groups: compound words, semi-compound words, and syntactic groups. From the authors' perspective, compound words should be recognized as a single token in the tokenization process, whereas if the structure is a syntactic group, its components should be divided into separate units. Additionally, semi-compound words exhibit intermediate behavior, acting like compound words in some respects and like syntactic groups in others. The authors claim that one of the major challenges in the tokenization process is the segmentation of compound, semi-compound words, and syntactic groups, as well as distinguishing between them. In one section of the article, the authors discuss the criteria for differentiating compound words from syntactic groups, followed by the distinction between compound and semi-compound words. At the end of the paper, the efforts made in preparing a benchmark dataset are described. The benchmark dataset, collected from news sources, possesses three key features: at the first level, the compound or semi-compound nature of each data point is specified; at the second level, their POS tags are determined; and at the third level, the type of challenge a tokenizer faces when encountering that data is identified. Some of the challenge categories mentioned include conjunctions, numerals, infinitives, inflectional forms, and named entities. The authors labeled 602 sentences from 9 news sources as benchmark data, with a total of 21,183 words and an average sentence length of 40.28 words. As seen in the explanations, this paper focuses on the first and second stages of the four-stage tokenizer evaluation process, namely the identification of challenges and the provision of benchmark data.

Qayoumi [11] argues that Persian texts suffer from two issues: the first issue is multi-unit words, which result from the attachment of one word to the following words. The second issue is multi-word units, which occur due to the separation of words that together form a single lexical unit. In his paper, he introduces an algorithm designed to automatically reduce these two issues in Persian text. This algorithm prepares the groundwork for text tokenization by correcting the spaces within the text, and based on these spaces, the tokenization process can be performed. The proposed algorithm consists of three stages: In the first stage, multi-unit words are separated, and multi-word units are joined together. For this stage, a basic algorithm based on a language model is introduced, which handles the segmentation of multi-unit words into independent words. In this stage, a morphological analyzer is used to examine inflectional and derivational affixes. Additionally, to prevent incorrect segmentations, a matching method with Bijankhan corpus words is utilized. In the second stage, continuous prefixes and inflectional clitics that have been separated from the verb are reattached to the verb to form a single word. The third stage repeats the first stage to address any new issues in the text created after the execution of the second stage. Using this algorithm on Persian language databases, 72.40% of the spelling errors in the test dataset were corrected. The accuracy of this correction in the test data was 97.80%, and the spelling error rate introduced by this algorithm in the test data was 0.02%. This paper focuses on the third stage of the four-stage tokenizer evaluation process, namely, the testing and evaluation framework.

Kamali and colleagues [12] studies some of the most widely used tokenizers for Persian and compared and evaluated their performance on Persian texts. In their paper, a part of Persian Dependency treebank was used as the test data. The authors presented an algorithm to compare the results of the tokenizers with benchmark data and, after analyzing tokenizer errors, found that word-bound morphemes were the main issue causing tokenization problems. Therefore, to improve the results, a list of word prefixes and suffixes was prepared, and their spacing with words was corrected before tokenizing the text. They evaluated the tokenizers in two ways: in the first approach, the results of each tokenizer were evaluated independently on 29,982 sentences from Persian Dependency

treebank and got the F1 scores of 94.46% for Stanza, 97.63%, for Parsivar, 98.75% for Hazm, 94.47% for Trankit, 94.47% for SetPer and 97.12% for FarsiVerb [6]. The second evaluation method involved using the FarsiVerb tokenizer alongside each tokenizer to assist in identifying Persian verbs. In such cases, the results showed slight improvements. For example, the F1 score of Hazm increased from 98.75%, 98.84% using Farsi Verb. As explained, this paper focused on the third and fourth stages of the four-stage tokenizer evaluation process and made the total assessment not a parametric one per challenge.

Qayoumi [13] considers issues of human origin to be the source of challenges in linguistic data processing. Therefore, before data processing, it is necessary for the data to undergo preprocessing. This preprocessing creates a relative uniformity in the data, resulting in increased efficiency of the processing algorithm. In this survey paper, he examines several problems related to Persian script and addresses fundamental processes such as tokenization. In one section of the paper, the issue of multi-word expressions in the Persian language and various research conducted around it are explained. In the tools section, he studied some tools and reported their performance according to the tool creators' claims. In his study the performances reported by various authors are just gathered in a table and as the results are not obtained on a unique dataset the comparison is not valid or fair. It can be just used as a piece of information about the results that each author has obtained from testing his own tool on his own test data. This research reports the accuracy, 97% for Persianp [14], 87% for Step-1 (testing on a small challenging dataset) and F1 score 96.48% for Seraji and colleagues. [15] and 89.5% for Parsivar (testing on Bijankhan).

3. TOKENIZATION CHALLENGES IN THE PERSIAN LANGUAGE

We classify the main challenges of Persian tokenization into four categories: multi-function morphemes, non-textual elements, missing spaces, and extra spaces. These categories are shown in Table 1. As the table shows, each category consists of various phenomena and each phenomenon has some types.

Multi-function morphemes refer to morphemes that have more than one function or role. Sometimes, they act as a standalone token, while at other times, they need to be combined with the preceding or following token as affixes or clitics. In this category there are two linguistics phenomena for affixes and punctuations. For example, the morpheme "می" can be pronounced "mey" and mean "wine" as a standalone token, or it can be pronounced "mi" and serve as a prefix for continuous verbs. Similarly, a period (.) can be an independent token marking the end of a sentence, or it can be used as a decimal point, an abbreviation marker, or a separator in dates.

Non-textual elements include numbers, symbols, and non-Persian text (e.g., English characters) that appear within the Persian text. These elements require the establishment of standards for identifying the encompassing tokens or their surrounding tokens.

Additionally, since in the Persian language, "white space" does not serve as a deterministic boundary for words, tokenization faces various challenges. On one hand, spaces between words may be omitted in some cases (e.g., ازدیروزتاحالا (Tr: FromYesterdayTillNow))[4], creating challenges similar to those encountered in Chinese and Japanese languages. On the other hand, a full space might be used instead of a zero-length-joiner (half-space) within a word (e.g. کتاب ها (Tr: book s)), making it difficult to determine the correct word boundaries. When spelling errors are added to these issues (e.g., بینالمللی), tokenization becomes even more complex.

So far, various tokenizers have addressed these four categories of issues and provided solutions. However, they may not cover all possible cases including various phenomena and types. For example, a tokenizer might have solutions for handling extra spaces in plural forms or the prefix "می" (mi) for continuous verbs, but it may not consistently handle definitive Ya (eg. خانه ای) or possessive pronouns (e.g., خانه شان vs. کتابشان). Therefore, the generated dataset has been designed to break down these challenges into finer-grained categories (phenomena and types), allowing for a more detailed evaluation of each tokenizer's capabilities.

Table 1 contains challenging cases that a tokenizer must be sensitive to and correctly address, which will be assessed accordingly.

Table 1 : Challenging Phenomena for Tokenizers

Category	Phenomenon	type	Example	code
Identifying multifunctional morphemes;	Distinguishing independent word from derivational affix	ای (Pr: ei) (Pr: ey)	رفته ای، ای پسر	1
		با (prefix:full) (Pr: ba) (Tr: with)	با ادب، با ماشین	2

[4] Throughout the paper, to make it readable for non-Persian speakers we show the translation (after Tr:) and pronunciation (after Pr:) of Persian words or phrases in parentheses after the Persian writing when necessary.

		می (Pr: mey) (Tr:wine) (Pr:mi)(continuous prefix)	می و شراب، می رفتم	3
		به (Pr: beh) (Tr: good) (Pr: be) (Tr:to)	به ورزی، به مدرسه	4
	Identifying tokens containing punctuation marks	Abbreviation (GCD) End of sentence decimal point Date etc	ب.م.م رفتیم. ۲.۳ ۹۹.۹.۲۹ ...	5
Separating Non-Textual Persian Elements;	Differentiating tokens containing numbers from standalone numbers	(Pr: F16) (Pr: 16kilos)	اف۱۶ ۱۶کیلو	6
	Identifying tokens containing punctuation marks	.")(}{[]....:	دیدم. (فراخوان)	7
	Identifying English tokens without spaces		استفاده ازcpuدر کامپیوترها	8
Extra Space;	Presence of extra spaces between words		من رفتم.	9
	Presence of extra spaces within an inflected word	Verb inflections	می آمدم، آمده ام، می خوانده ای، انداخته خواهد شد،	10
		Plural markers with or without indefinite or possessive clitics	کتاب ها، خانه هایی، کوشش هایم	11
		indefinite enclitic and EZAFE marker	لانه ای، سینی ای، دوره ی	12
		Compound verb	دستم انداخت، نفس نمی کشید	13
	The presence of an extra space between a word and its attached clitic	possessive clitics	آینه ام، بینی اش، خانه تان، رویه شان، دایی ات،	14
		Copulative verb	خسته ام، درمانده ای، آماده اند	15
		Object attached to the verb	می گریانم شان	16
		Combination of plural markers and possessive clitics	پیچیدگی هایش، دانسته هایشان	17
	Presence of space within a derived word		برق آسا، شب کار، بی ادب، جلوه گر	18
	Presence of space within a compound word		خدا ترس، حاصل خیز، سیب زمینی،	19

			لباس شویی	
Deleted space or half-Space	Deletion of space between two words	without spelling error	کامپیوتر ولپتاپ، بادوباران، دربرابرباد، اورادیدم	20
		With spelling error	کتابمن، شبیلدا	21
	Deletion of half-space within a word	Derived word	عاشقپیشه، میخواند	22
		Compound word	لباسشویی	23

4. THE EVALUATION BENCHMARK

To make a benchmark we need an evaluation corpus (testset) and an evaluation algorithm. To build the corpus we first gathered and categorized the challenges in Persian tokenization as shown in Table 1 and assigned a unique code to each phenomenon stated in the table. Then a group of 15 participants was asked to generate sentences for various phenomena, similar to those in Table 2. The result was a set of 2264 sentences supporting 3496 challenging phenomena. At last, we selected the most appropriate sentences from the generated ones and added some additional sentences to balance the representation of phenomena, and complete the set.

Table 2 shows samples of data with their designed tagging model. As shown in Table 2, the first column contains the generated sentences. The second column lists the correctly tokenized sentences (golden standard), with the "#" character used as the boundary between each token. In the third column, each token is assigned a label. Label "0" indicates regular tokens, while non-zero labels represent one of the phenomena in Table 1. According to the proposed algorithm, the output of the evaluated tokenizers is compared with the content of the second column, and based on the labels in the third column, the error rate for each phenomenon is calculated.

Table 2: A part of the evaluation corpus

Sentence	Correctly Tokenized Output	Labeling Format
انسان ها در وجودشان هاله ای از نور است	انسان ها #در #وجودشان #هاله ای #از# نور #است	#0#0#0#011#0#0#82
در نوشته های فلسفی مان کمتر به می حلال پرداخته ایم	در #نوشته های #فلسفی مان #کمتر #به #می #حلال #پرداخته ایم	081#0#013#0#0#091#082#0
آدم هایی که اچ.آی.وی می گیرند باید دارو هایشان را همراه داشته باشند	آدم هایی#که #اچ.آی.وی #می گیرند #باید #داروهایشان #را #همراه #داشته باشند	۰۸۱۱#۰#۰#۰۹۸#۰#۰۱۳#۰۲۱#۰#۰۸۲
تلخ زبانی انسان با فهم نشانه ی بی ثباتی شخصیت اوست	تلخ زبانی #انسان #با فهم #نشانه ی #بی ثباتی #شخصیت #اوست	۰#۰#۱۰#۰۸۳#۰۱۲#۰#۱۱

For further explanation, for example, in the last row of the table, we expect the tokenizer to recognize the tokens"بی " " and "ثباتی" as a single token. In the third column, the label "10" is assigned to this token. This label refers to the derived word in Table 1.

The characteristics of the constructed corpus can be seen in Table 3.

Table 2 : Corpus characteristics

Number of Sentences	**483**
Total Number of Tokens	4091
Number of challenging tokens	1010
Average Number of Tokens per Sentence	9.06
Average Number of Labeled Tokens per Sentence	2.2
Number of unique tokens	1602

Number of Tokens in the Longest Sentence	17
Number of Tokens in the Shortest Sentence	4
Average Number of Characters per Token	4.55
Standard Deviation of the Number of Characters per Token	6.98
Median Number of Characters per Token	4

The average number of characters per token indicates that, on average, each token in the dataset has approximately 4.55 characters. This number suggests that most tokens in the dataset are close to this length. However, the standard deviation represents the spread of the data relative to the mean. The standard deviation of 6.98, compared to the mean of 4.55, is relatively large. This means that the length of tokens in the dataset is highly variable, with significant differences between the lengths of different tokens. In other words, some tokens may be very short (e.g., 1 or 2 characters), while others may be very long (e.g., 10 or more characters).

4-1-EVALUATION ALGORITHM

This section provides the pseudocode for the method of evaluating tokenizer results.

To evaluate the results, we consider two sets of tokens; the reference set of tokens (R) and the output of the tokenizer (T).

- In each evaluation step, we call the current token in R (next token in the reference list) by C_R and the current token in T by C_T and two pointers called P_R and P_T point to C_R and C_T respectively. If C_T equals to C_R then both pointers move forward by one unit. This match indicates that the tokenizer has performed correctly and the number of correct output tokens increases by one.
- If C_T and C_R do not match, it means that the tokenizer did not function correctly and it probably has either splitted a single token into two tokens or combined two tokens into one. In this case, the following steps are taken:
- Among the current tokens in sets G and T, the shorter token is considered the "smaller token," which is then combined with the next token in its set.
- The combined result is compared with the "bigger token" (the token in the other set).
- If they match, the pointer in the bigger token set advances by one unit, while the pointer in the smaller token set advances by two units and the number of false output tokens increases by the number of pointer movements in the R list. This shifting of tokens allows the alignment process to continue.

If there is still no match after the initial combination, the combination process continues in the smaller token set until either a match is achieved or the size of the combined token exceeds the size of the bigger one or we reach the end of the list and the number of false output tokens is calculated similar to the previous case.[5]It should be noted that the number of correct and false output tokens are counted for each phenomenon as the system knows which reference token belongs to which phenomenon (see table 2).

[5] As the tokenizer tools do not generate or delete the tokens (substrings) there is no need to consider unknown or missed tokens in the output. But if this was not the case we should change the algorithm to cover unseen or missed tokens too.

```
Initialize index_token to 0
Initialize index_model to 0

For j from 0 to length of tokens:
    list_tokens = tokens from index_token to j+1
    concatenated_tokens = concatenate elements of list_tokens
    list_codes = codes from index_token to j+1

    For k from 0 to length of model:
        list_model_tokens = model from index_model to k+1
        concatenated_model_tokens = concatenate elements of list_model_tokens

        If check_tokens(concatenated_tokens, concatenated_model_tokens) is True:
            Update index_model to k + 1
            Update index_token to j + 1

            If length of list_tokens is not equal to length of list_model_tokens:
                Increment incorrect_count by 1
            Else:
                Increment TP_count by 1

            Break out of the inner loop (model loop)

```

Figure 1 : Evaluation algorithm

To implement such a structure, two loops are considered: the outer loop corresponds to the benchmark data, and the inner loop corresponds to the tokenizer's output. The process is as follows: in each iteration of the outer loop, a token from the benchmark data list is selected. After this step, we enter the inner loop, where scenario 2 is followed. If we reach the end of the list without finding a match, we proceed to scenario 3.

5. EXPERIMENTAL RESULTS

For the evaluation of Persian tokenizers, the tools, Hazm, Dadma, Polyglot, Parsivar, Parsinorm, and Step-1 were accessible and could be installed and run. Despite considerable efforts, other well-known tools were not evaluated due to either lack of access or installation and runtime errors. To compare the performance of these tokenizers with newer technologies on using large language models, we chose two language models ParsBERT and Llama3-8b to be added to this benchmark. We know that the tokenization algorithms in these models are usually different from a-word-as-a-token models but we want to see how is their performance in detecting word boundaries comparable to available Persian tokenization tools.

We chose Llama3-8b among the LLMs because the larger models with more than 8 B parameters are too large and need substantial computational resources and using them for word boundary detection which is a primary preprocess is not cost effective. In Table 4, the accuracy of each tool for each phenomenon from Table 1 is presented separately. The second column lists the number of instances of a phenomenon in the test data.

Table 4 : Accuracy of different tokenizer on Phenomena

Phenomenon	Number of Samples	Accuracy percentage in ParsBert	Accuracy percentage in Hazm	Accuracy percentage in Dadma	Accuracy percentage in Polyglot	Accuracy percentage in parsivar	Accuracy percentage in parsiNorm	Accuracy percentage in Step-1	Accuracy percentage in Llama3-8b
Multi-functional Morpheme «ای»	44	**95.46**	65.91	93.19	93.19	63.64	93.19	**95.46**	89.11
Multi-functional Morpheme «یا»	35	8.58	0	2.86	2.86	2.86	2.86	**45.72**	0

Multi-functional Morpheme «می»	81	25.93	**85.19**	24.7	27.17	80.25	24.7	82.72	25.26
Multi-functional Morpheme «به»	33	3.04	6.07	9.1	12.13	12.13	9.1	**18.19**	6.76
Multi-functional Morpheme «.»	37	24.33	24.33	40.55	**81.09**	45.95	37.84	21.63	77.39
Token Containing Numbers	36	58.34	**66.67**	47.23	47.23	52.78	63.89	63.89	41.88
Separation of Punctuation Marks	52	90.39	94.24	92.31	**96.16**	94.24	88.47	92.31	86.15
Detection of English Tokens	36	91.67	19.45	16.67	16.67	16.67	**94.45**	88.89	86.89
Detection of Extra Spaces	40	97.5	97.5	97.5	97.5	97.5	97.5	**100**	94.45
Extra Space in (Verb)	36	11.12	**100**	13.89	11.12	47.23	11.12	97.23	9.46
Extra Space in (Plural Markers)	68	20.59	**98.53**	20.59	23.53	97.06	20.59	97.06	19.10
Extra Space in (indefinite enclitic)	50	88	96	94	96	94	96	**98**	90
Extra Space in (Compound Verb)	43	16.28	86.05	13.96	16.28	18.61	11.63	**90.7**	13.27
Extra Space in (Possessive clitic)	56	35.72	53.58	37.5	37.5	55.36	37.5	**98.22**	30.31
Extra Space in (Attached Copular Verb)	34	32.36	73.53	32.36	32.36	58.83	32.36	**97.06**	28.37
Extra Space in (Direct Object Attached to Verb)	32	3.13	43.75	3.13	3.13	31.25	3.13	**96.88**	3.13
Extra Space in (Plural Markers and Possessive clitic)	40	0	0	0	0	**90**	0	**90**	0
Extra Space in (Derived Word)	46	10.87	58.7	8.7	10.87	47.83	8.7	**84.79**	9.42
Extra Space in (Compound Word)	96	6.25	42.71	4.17	8.34	25	4.17	**70.84**	6.08
Space Omission (Without Error)	37	**81.09**	8.11	5.41	8.11	5.41	8.11	**81.09**	71.83
Space Omission (With Error)	37	**83.79**	5.41	5.41	10.82	5.41	5.41	8.11	79.45
Non-challenging tokens	3120	**94.62**	93.88	92.83	93.34	92.89	93.95	92.31	92.71

As it can be seen, the results of LLaMA3-8b do not outperform other tokenizers and, in some cases, are even weaker. Thus, according to the computational and memory resources it uses, it is not effective to use it to serve as a word boundary detector. Therefore we ignore Llama in our further analysis and just show its overall performance in table 5. In the next sections we focus on the other tools and analyze them in more details for each phenomenon under study.

5-1 COMPARISON OF TOKENIZER ACCURACY FOR THE MULTI-FUNCTION MORPHEME GROUP

Figure 2 presents a comparison of tool accuracy within the multi-function morpheme group.

Key observations from this figure are as follows:

- Most tools treat a period as a sentence-ending character and recognize it as a separate token, whereas a period can also be used in decimal numbers, dates, and abbreviated names with different meanings.

- With the exception of Step-1, the word "با" is recognized as a separate token by all tools. However, in a phrase like "او با انصاف است", this word should be combined with "انصاف" into a single token.
- Similarly, the word "به" exhibits the same issue, and this oversight is present in all tools.
- The tools Hazm and Parsivar exhibit nearly identical behavior in most cases.
- The treatment of the word "می" is also noteworthy. In this case, tools are divided into two groups: Hazm, Step-1, and Parsivar in one group, and the remaining tools in another. Further examination reveals that both groups have made errors in similar sentences, suggesting that the algorithm used at this stage has common points of failure.

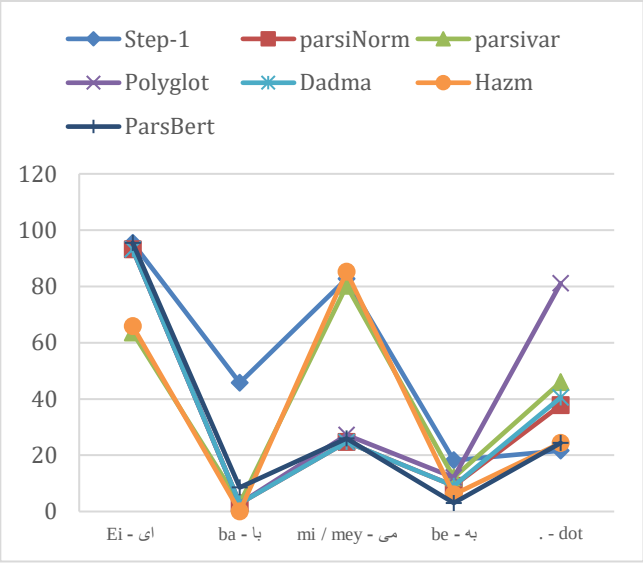

Figure 2 : Comparison of tokenizer Accuracy for multi-functional morphemes

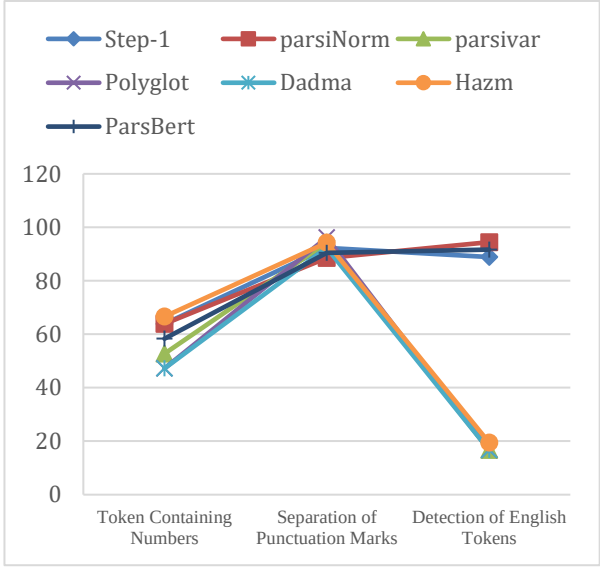

Figure 3 : Comparison of tokenizer Accuracy for non-textual Persian

5-2 COMPARISON OF TOKENIZER ACCURACY FOR NON-TEXTUAL PERSIAN SECTIONS

Figure 3 presents a comparison of tool accuracy in the category of non-textual Persian symbols. Non-textual symbols in this section include numbers, punctuation marks, and English tokens. The tokenizers' output for each of these elements is as follows:

- The tokenizer should recognize a number as a separate token unless the combination of the number with letters suggests a well-known concept, such as "تفنگ ژ۳".
- Punctuation marks such as (. , : ; ? !) are typically attached to the preceding word in typesetting principles; therefore, when tokenizing text, these marks should be separated from the word to which they are attached.
- English words used in Persian text should be identified as separate tokens, and if an English word is incorrectly attached to a part of the Persian text, it should be properly separated.

Key observations from this figure are as follows:

- The tools have performed well in separating punctuation marks.
- The tools exhibit errors in separating tokens containing numbers, often separating the number from its textual part without considering the token, creating a numeric token.
- In identifying tokens containing English phrases, Parsinorm demonstrates the best performance.

5-3 COMPARISON OF TOOL ACCURACY FOR THE EXTRA-SPACE GROUP

This section represents one of the most frequent cases encountered by tokenizers. Aside from identifying compound words, most of the other cases can be recognized by defining rules and accessing a list of known words. Therefore, due to the rule-based nature of these cases, tokenizers can potentially achieve the lowest error rate in this section.

Key observations from figure 4 are as follows:

- In terms of the fewest errors, Step-1 ranks first, followed by Hazm in second place.
- Parsinorm, Polyglot, Dadma, and ParsBERT exhibit nearly identical performance in almost all cases, and this level of similarity in their performance across various phenomena is noteworthy.
- Given the irregular nature of compound words, identification in this area does not yield satisfactory results. It was expected that ParsBERT, given its underlying infrastructure, would perform better in this section.

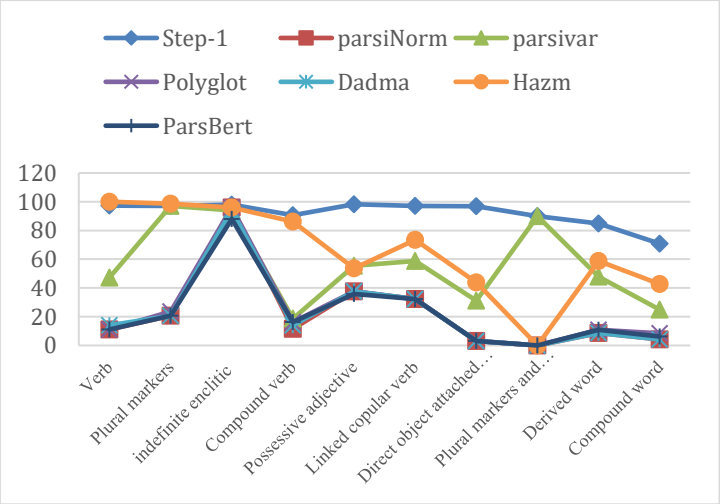

Figure 4 : Comparison of tokenizer Accuracy for the extra space group

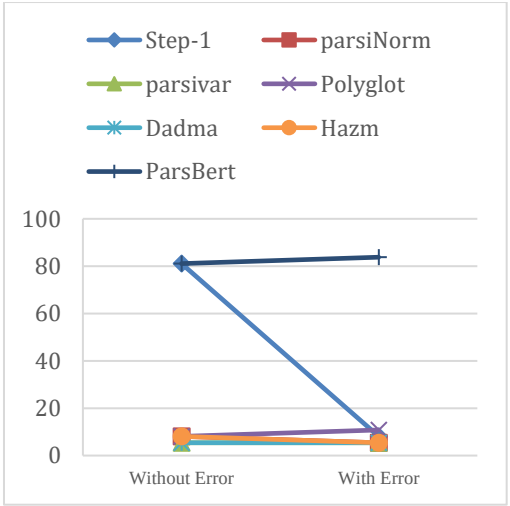

Figure 5 : Comparison of tokenizer Accuracy for space removal

5-4 COMPARISON OF TOKENIZER ACCURACY FOR THE SPACE REMOVAL GROUP

In this section, a token is formed by removing the space between two words, and it is expected that the tokenizer will recognize this token, separate it, and convert it into two distinct tokens. The removal of space results in two types of word forms:

- A token that is not technically incorrect in writing but has reduced readability (e.g., "مادرمن").
- A token that is incorrect in writing (e.g., "هوشمصنوعی").

The results in this section are not particularly remarkable, but we encounter a phenomenon with ParsBERT, which performs better than the other tools. This indicates that ParsBERT, due to its underlying language model, can effectively handle the correction of spelling errors. Table 4 shows that the best results of LLama3-8b were also in this section.

5-5 COMPARISON OF TOKENIZER ACCURACY

Figure 6 displays the accuracy of each tokenizer across four categories: "Multi-function morpheme ", "Non-Textual Persian", "Extra Space" and "Space Removal". As observed, in “Multi-function morphemes” section although the performance of tools are nearly the same, Step-1 performs slightly better than the others. In the "Non-Textual” section, most tools exhibit similar behavior with no particularly notable results. In the "Extra Space" category, which encompasses a significant portion of the phenomena, Step-1 demonstrates a considerably better performance compared to the other tools and in the “Space Removal” section ParsBert is considerably better than the others and SteP-1 is in the second rank while the others have very low performance.

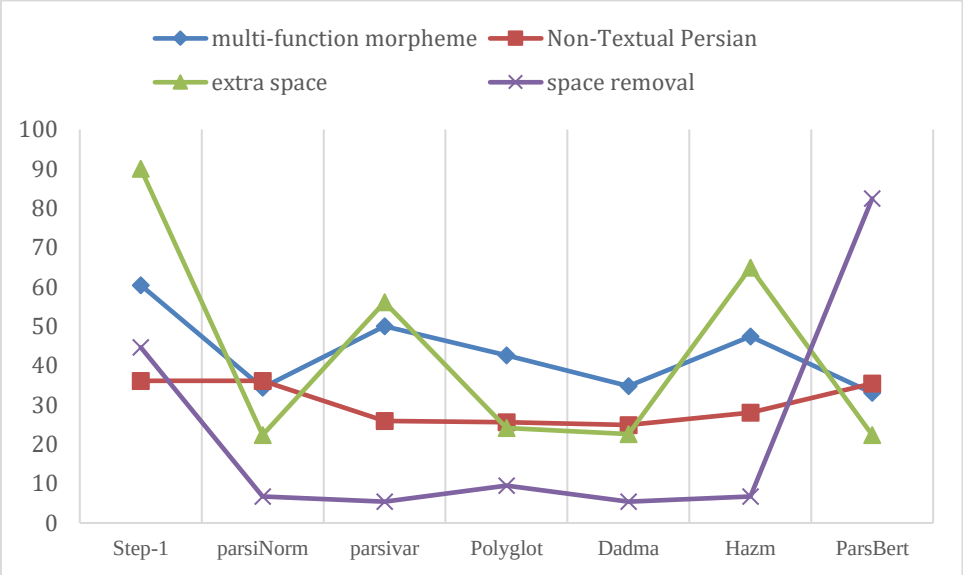

Figure 6: Comparison of tokenizer accuracy relative to phenomena

5-6 TOTAL EVALUATION

To evaluate the results of each tokenizer's performance, two approaches were taken:

1. In the first approach, the results were calculated based on all tokens in the test data (challenging and non-challenging tokens). Although non-challenging tokens do not pose significant challenges in tokenization, it was observed that some tokenizers made errors, either by merging tokens or by splitting a simple word into two tokens.
2. In the second approach, the results were calculated just based on the challenging tokens in the test data (1010 tokens).

Table 5 presents these results.

Table 5 : Evaluation of Metrics on All Tokens and Challenging Tokens of the Corpus

	Evaluation metrics based on all tokens of the corpus			Evaluation metrics based on challenging tokens of the corpus		
	Recall	**Precision**	**F1**	**Recall**	**Precision**	**F1**
ParsBert	75.62	63.36	68.63	41.29	35.83	37.90
Hazm	84.52	81.28	82.52	57.14	51.51	53.74
Dadma	74.75	66.03	69.64	34.05	29.19	31.05
Polyglot	74.97	66.32	69.90	37.47	32.76	34.55
parsivar	80.94	76.99	78.52	51.39	46.11	48.18
parsiNorm	76.26	66.41	70.59	37.74	32.69	34.62
LLama3-8b	74.10	66.01	69.42	36.11	29.42	31.92
Step-1	**88.51**	**87.16**	**87.57**	**77.87**	**73.54**	**75.24**

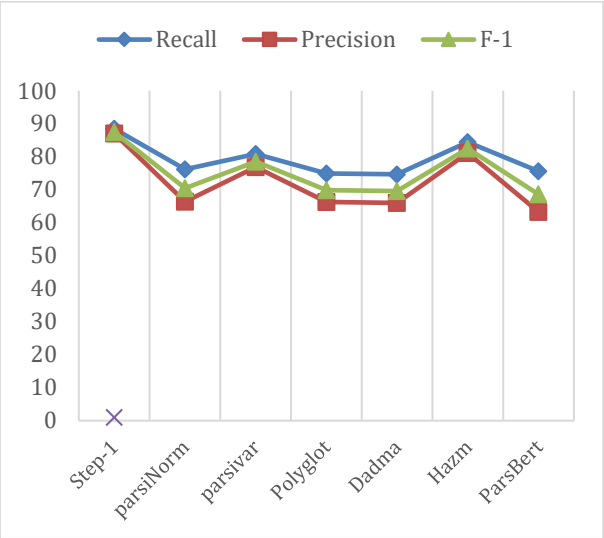

Figure 7: Evaluation metrics based on all tokens of the corpus

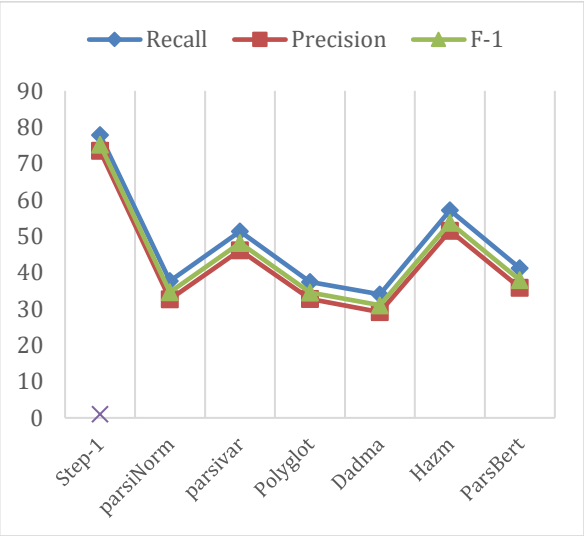

Figure 8: Evaluation metrics based on challenging tokens of the corpus

Some analytical observations from Table 5 are as follows:

- Step-1 shows the best performance across both categories of data (all tokens of the corpus and challenging tokens). This tool has the highest Recall (88.5 and 77.8), Precision (87.1 and 73.5), and F1 (87.5 and 75.2) scores. Especially for challenging tokens, its performance is significantly better than the other tools.
- Hazm also performs very well on all tokens of the corpus, with Recall (84.5), Precision (81.2), and F1 (82.5). This shows that Hazm is relatively accurate in processing non-challenging data, but it performs less well with challenging tokens (Recall: 57.1, Precision: 51.5, F1: 53.7).
- Parsivar also shows notable performance among the other tools. With Recall (80.9) and Precision (76.9) for all tokens, this tool demonstrates reasonable accuracy in tokenization. However, it faces a significant decrease in performance for challenging tokens (Recall: 51.3, Precision: 46.1, F1: 48.1).
- ParsBert and Dadma have lower performance compared to the top tools in both categories of data. Particularly for challenging tokens, both tools show low F1 scores (ParsBert: 37.9, Dadma: 31), indicating weaknesses in handling more complex linguistic challenges.
- The evaluation of the LLaMA3-8b language model shows that its performance in the tokenization task falls between the Dadma and Polyglot tokenizers, performing slightly better than Dadma but somewhat weaker than Polyglot.
- parsiNorm also shows moderate performance for processing all tokens (F1: 70.5), but like many other tools, it experiences a decline in accuracy when dealing with challenging tokens (F1: 34.6).
- Polyglot has similar performance to other mid-range tools, with an F1 score of 69.9 for all tokens and 34.5 for challenging tokens.

6. SUMMARY AND CONCLUSION

Based on the overall results obtained in this study, it appears that before developing a robust tokenizer for the Persian language, it is more crucial to prioritize the identification of challenges and issues that a tokenizer might encounter. The authors of this paper have made efforts to first identify such challenges and prepare the test data accordingly. In the next step, the most commonly used and accessible tools for the Persian language were tested and evaluated in this regard.

The results indicate that the Persian language requires more attention in the area of tokenization, and the development of a powerful and accessible tool should be prioritized.

During the review and preparation of this paper, efforts were made to install and test all the tools mentioned in the introduction section of this paper. Some of the problems we encountered are as follows:

- There was no reliable and specific source for downloading tokenizer resources.
- Some tools mentioned in research were not accessible.
- Installing some tools was challenging, and in some cases, various tools had to be installed.
- The tools were designed using different programming languages.
- Some tools did not have a normalizer.

- There was no consistent standard in the output of the tokenizers.

Therefore, the authors intend to focus on developing a language model-based tokenizer with easy access in future endeavors.

REFERENCES

[1] D. Jurafsky and J. H. Martin, Speech and Language Processing, 3rd ed. Pearson, 2023.

[2] Peng Qi, Y. Zhang, Y. Zhang, J. Bolton, and C. D. Manning, Stanza: A Python Natural Language Processing Toolkit for Many Human Languages, In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, 2020.

[3] S. Mohtaj, B. Roshanfekr, A. Zafarian, and H. Asghari, Parsivar: A language processing toolkit for persian. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018).

[4] R. Oji, S. Razavi, S. Abdi Dehsorkh, A. Hariri, H. Asheri, R. Hosseini, ParsiNorm: A Persian Toolkit for Speech Processing Normalization, 7th International Conference on Signal Processing and Intelligent Systems (ICSPIS), 2021.

[5] M. Shamsfard, H. Jafari, and M. Ilbeygi, STeP-1: A Set of Fundamental Tools for Persian Text Processing, Manuscript ACM, 2010.

[6] M. Manshadi, Farsi Verb Tokenizer, Available online: https://github.com/mehdi-manshadi/Farsi-Verb-Tokenizer, Accessed: 2024.

[7] R. Etezadi, M. Karrabi, N. Zare Maduyieh, M. B. Sajadi, and M. T. Pilehvar, DadmaTools: a Natural Language Processing Toolkit for the Persian Language, Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: System Demonstrations, pp 124 - 130, 2022

[8] M. Farahani, M. Gharachorloo, M. Manthouri, Leveraging ParsBERT and pretrained mT5 for persian abstractive text summarization, 26th International Computer Conference, Computer Society of Iran (CSICC), 2021, pp 1–6.

[9] M. Sharifi Atashgah, M. Bijankhan, Corpus-based analysis for Multi-Token Units in Persian, proceedings of the 3rd Workshop on Computational Approaches to Arabic Script-Based Languages [at] MT, Ottawa, Canada, 2009.

[10] M. Farhoodi, M. Mahmoudi, and M. Davoudi Shamsi, Creation of a tagged corpus for Persian tokenization considering computational linguistics considerations, Quarterly Journal of Signal and Data Processing, no. 3, issue 19, pp. 157-188, 2022 (In Persian)

[11] M. Qayoumi, A Language Model-Based Method for Tokenizing the Persian Corpus, Institute for Humanities and Cultural Studies, Vol. 14, No. 27, pp. 21-50, 2018 (In Persian)

[12] D. Kamali, B. Janfada, M. E. Shenasa, B. Minaei-Bidgoli, Evaluating Persian Tokenizers, arXiv: 2202.10879, 2022

[13] M. Qayoumi, Preprocessing and basic tools, Persian Text and Speech Processing, SAMT Publications, pp. 86-113, 2023 (In Persian)

[14] M. Mohseni, J. Ghofrani and H. Faili, Persianp: A persian Text Processing Toolbox, Computational Linguistics and Intelligent Text Processing, Vol. 9623, Springer, Cham.

[15] M. Seraji, B. Megyesi, and J. Nivre, A Basic Language Resource Kit for Persian. In Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12), pages 2245–2252, Istanbul, Turkey.

Mehrnoush Shamsfard is an Associate Professor at the Faculty of computer science and Engineering, Shahid Beheshti University. Her research interests include Natural Language Processing and Knowledge Engineering. She is the head of the Natural Language Processing and Knowledge Engineering Laboratory at Shahid Beheshti University.
Email: m-sham@sbu.ac.ir
ORCID: 0000-0002-7027-7529

Mostafa Karimi Manesh is a Ph.D. student in Artificial Intelligence at the Faculty of computer science and Engineering, Shahid Beheshti University. He is interested in Natural Language Processing and works on language models and their evaluation.
Email: m_karimimanesh@sbu.ac.ir
ORCID: 0009-0001-7410-3999