Using a Pre-Trained Language Model for Context-Aware Error Detection and Correction in Persian language

Anonymous ACL submission

Abstract

This paper presents a Persian spell checker called Virastman, which aims to detect and correct non-word and real-word errors in a sentence. A state-of-the-art method based on sequence labeling with BERT detects real-word errors on a small artificially made dataset. An unsupervised model based on BERT is used for correcting errors by calculating the probability of each candidate in a sentence (including the detected word). A highly probable candidate word is selected as the correct word if some conditions are met based on two thresholds named α and β. Our experiments across six distinct test sets underscore our proposed methodology's notable superiority in detecting and correcting real-word and non-word errors compared to the baselines. More specifically, our approach demonstrates an average enhancement of 3.41% in error detection and an average substantial 15% in error correction when assessed using the $F_{0.5}$ metric, thus surpassing contemporary baselines, establishing our method as the state-of-the-art for error detection and correction.

1 Introduction

Spelling error correction has been the subject of numerous studies.(Hládek et al., 2020) Spell checkers can help people write text without any errors. Language learners can learn a language more successfully by identifying and correcting written mistakes. Spell checkers are also useful in many applications, namely as a post-processing step in speech recognition (Priya et al., 2022) and OCR (Hangaragi et al., 2023). Moreover, they are useful to have better results in search engines (Li, 2020).

Spelling errors are classified into two categories: non-word and real-word errors. Non-word errors involve words that are incorrect and do not exist in the language, while real-word errors encompass words that are part of the language but lack the appropriate meaning in the given context. Existing Persian spell checker tools perform well in detecting non-word errors but are not good at correcting them. It can be said that they do not have the ability to detect and correct real-word errors. Real-word mistakes are not scarce; in fact, they account for 25% to 40% of observed spelling mistakes (Mitton, 1987). For improving non-word error correction and real-word error detection and correction, we represent a Persian Spell Checker called *Viratsman*.

Persian is a low-resource language which lacks large body of data for using supervised methods spell checking methods (Hagiwara, 2021; Jayanthi et al., 2020). To address the data scarcity problem, we present a method for real-word error detection which works even on small artificially generated data. Furthermore, for error correction, we employ an unsupervised approach that leverages pre-existing language models.

Our contribution is summarized below: (1) Developed an unsupervised method for error correction based on existing language models. (2) Achieved the highest correction rate in all test sets in comparison to all other existing Persian spell checkers. (3) In a dataset where approximately 66.9% of the errors were real-word errors, notable improvements were achieved in the $F_{0.5}$ metric. Error detection was enhanced by approximately 35%, and error correction was improved by nearly 40% through the model that was developed. (4) Derived unique threshold values for a spell checker model by employing distinct datasets tailored to individual error categories.

The rest of the paper is structured as follows. A summary of relevant work is presented in Section 2. Models for spelling detection and correction are discussed in Section 3. Section 4 discusses experiments and findings. The paper is concluded in Section 5. Finally, Section 6 discusses limitations.

2 Related Work

A spell checker consists of two primary phases: error detection and correction. The most popular and simple way to detect non-word errors is using a dictionary (Faili et al., 2016; Hládek et al., 2020). Other methods for detecting all error types are deep-learning-based models such as encoder-decoder (Zaky and Romadhony, 2019; Dehghani and Faili, 2023) and sequential binary labeling models (Madi and Al-Khalifa, 2020; Liu et al., 2022; Zhang et al., 2020) which is used for detecting real-word errors in Virastman[1].

Recent large language models, like BERT and GPTs, perform well on various tasks, including mistake correction. They can calculate word or sentence probability, which is useful for error correction. A 5-gram language model is utilized in (Bryant and Briscoe, 2018) for correcting non-word and grammatical errors, and their model was a benchmark for many other language model error correction models. The advantage of using a language model for error correcting is that it is an unsupervised model and does not need annotated data. Their model determines the word that can be replaced with this incorrect word by computing the logarithm of the sum of the sequence of words that contains an error. The candidate word is substituted if one of the logarithms of the probability of the words is higher than the original word, and this difference is bigger than a threshold. They reported the performance of their model based on several thresholds rather than calculating the best value for the threshold, and it is advised that this be done in the future. Grammarly (Alikaniotis and Raheja, 2019) does the same as (Bryant and Briscoe, 2018), but in correcting grammatical errors, they used GPT and GPT2 as a language model. For calculating the probability of a sentence, they replace each word in a sentence from left to right with a `[MASK]` token and then calculate the sum of the logarithm of words in a sentence. They used 0,2,4,6,8 as a value for the threshold and selected the best one for the threshold. In our work, we fined-tuned a model for calculating the threshold, and each error type has a different value for the threshold. BERT is used as a language model for correcting non-word and real-word errors in (Hu et al., 2020). A dictionary detects non-word errors, but it considers that the real-word errors are detected and then tries to correct the error. This paper used two methodologies to rank the suggestions. The first one ranks the BERT suggestion word and then selects words with low edit distance. The latter made a suggestion list based on edit distance and then ranked the suggestions. We used the second method for ranking the suggestions.

[1] https://virastman.ir/

3 Virastman Spell Checker

This paper addresses two distinct error categories within the Persian language, namely non-word real-word errors. In order to optimize the speed and simplicity of our spell checker while accommodating the intricacies of the Persian language, we have devised a streamlined five-step pipeline. This pipeline consists of pre-processing, error detection, suggestion word generation, error correction, and post-processing stages. It is worth noting that the pre-processing and post-processing phases exhibit a uniform mechanism for both non-word and real-word errors, while the remaining components employ distinct models and processes. The pipeline of Virastman is shown in Figure 1. In this paper and within the Virastman tool, the initial focus is on non-word errors, followed by the consideration of real-word errors.

3.1 Pre-processing

The pre-processing step does not need complex models with deep learning; it can be done based on some rules. The first rule used in this paper is called *generalization*, a Unicode normalization that converts numbers, Persian, and English alphabet to the standard Persian form. For this purpose, we used the generalization part of ParsiNorm (Oji et al., 2021a) Python library. The second rule is correcting *zero-width non-joiner* (ZWNJ). Persian is a morphologically rich language(Khallash et al., 2013) and hence, a word can have multiple prefixes and suffixes; some of them are connected

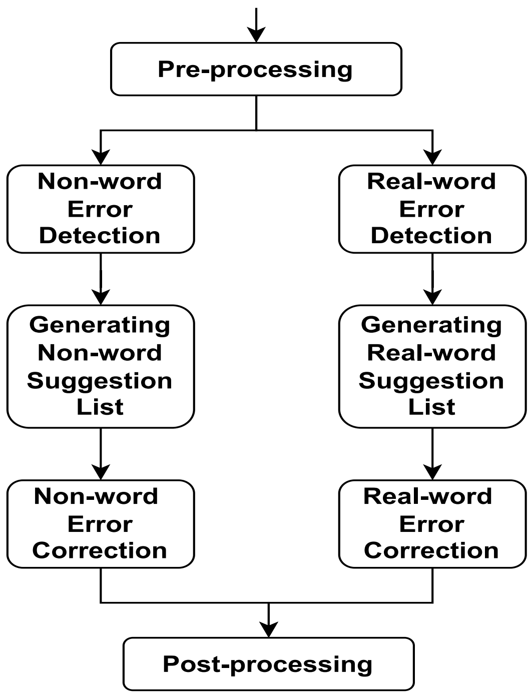

Figure 1: Pipeline of Virastman Spell Checker

to the main word with or without a space character or a ZWNJ character. Concatenation of prefixes and suffixes to the main word depends on a word's meaning and its part of speech. Due to the aforementioned morphological complexity of Persian, one cannot make a list of all possible variations. We start from a reference for the correct form of words, namely Virastaran[2], and create rules for words with suffixes and prefixes based on their part of speech in a sentence.

3.2 Non-word error detection

Detecting non-word errors is a straightforward process, primarily because these errors involve words that do not exist in the language's dictionary. Thus, the key to error detection lies in having an extensive language dictionary that encompasses all word forms. For instance, if a word is a verb, the dictionary must include all variations of verb tenses. A dictionary consists of unigrams and should be generated from the texts written in a particular language. To create a detection dictionary encompassing all the language's words, we employed the Persian raw text corpus[3], which is a collection of extensive 80GB text files, and from it, we extracted the unigrams.

Three annotators then annotate all of the words in this dictionary. The final decision on whether the word is correct is based on the majority vote of the annotators. One important note is that the dictionary of a language is not static and can be changed. Due to the dynamic nature of language and the fact that Virastman is used by multiple users who occasionally contribute new words that are detected as mistakes, annotators look into any new words that Virastman users add, and if they are determined to be correct words, these annotated words are then added to the existing dictionary.

3.3 Non-word suggestion list

After detecting errors, we need a list of suggested words to select the correct word to be replaced with the wrong word. This list contains words with one edit distance away from the wrong word in deletion (removing one character), insertion (adding one character), and spacing (removing space between two words or adding space between two concatenated words) and two edit distances away using transposition (changing place of two adjacent characters) and substitution (replacing one character instead of another character). The suggested list contains words that cover 80 to 90 percent of errors in a large corpus (Pollock and Zamora, 1984).

3.4 Non-word error correction

For correcting errors, the first step is ranking the suggestion list and then replacing the first ranked suggested word with the wrong word. To do this, we need a language model which considers words around the wrong word and gives the sentence probability. We employ an unsupervised method, utilizing BERT as a language model, to arrange the suggestions in order of their likelihood. A sentence is comprised of a sequence of words $X =< x_1, x_2, ..., x_n >$ and after the detection of wrong words, the wrong word is replaced by `[MASK]` token and the probability of suggested words are calculated. If a sentence has multiple wrong words, only one word at a time is considered wrong and corrected. This process continues until no incorrect word remain. Note that addressing all errors at once is possible; however, it is

[2] https://emla.virastaran.net/

[3] https://github.com/persiannlp/persian-raw-text

more complex in terms of runtime complexity (Alikaniotis and Raheja, 2019).

The idea behind using language models is that if there is an error in a sentence, that sentence has a lower probability in comparison to the sentence with all correct words (Bryant and Briscoe, 2018). To find the likelihood of a sentence, we mask each word in a sentence and then compute the summation of the log probability of the next word in the sentence. (Wang and Cho, 2019) has shown that the computed value is a good approximation of the likelihood of a sentence. If a sentence is long, the probability of the sentence becomes large. On the other hand, if the sentence is short, the probability of the sentence becomes small, so in order to prevent selecting two-part words (when two words are concatenated and the correct form can be the separated form), the sum of the logarithm of probability is normalized based on the sentence length. More concretely, the sentence likelihood is calculated as follows.

$$\frac{1}{n}\sum_{i=1}^{n}\log P(W_i) \tag{1}$$

Certain correct words, such named entities, may not be in the detection dictionary, and they are labeled as wrong words. By adding the wrong word to the suggestion list, the probability of original word in the sentence is the highest value, and the sentence remains unchanged. This technique reduces the probability of converting correct to wrong words (i.e., false positives).

A word with the highest probability is replaced with the wrong word, when the following conditions are met.

1- Probability of a sentence is greater than a certain threshold. Consider that $P(C)$ is the logarithm of the probability of a candidate sentence. When $P(C) > \alpha$, this candidate sentence can be the correct form of the sentence. α is a hyperparameter. In the context of sentences containing non-word errors, this parameter is determined empirically through an iterative procedure, selecting the optimal value by achieving a trade-off between the precision and recall of the correction among approximately 2000 values. This calculation is based on a dataset in which every sentence is known to contain at least one erroneous word, and this dataset is gathered from Virstman logs. This dataset contains 5k sentences. The best value is $\alpha = -25$. The advantage of using this condition is that when a sentence contains unfamiliar words, the probability that this sentence is converted to an incorrect sentence is reduced.

2- The difference between the probability of the first and second words is greater than a certain threshold. $P(W^1)$ is the logarithm of the probability of the first-ranked candidate, and $P(W^2)$ is the logarithm of the probability of the second-ranked candidate. If $P(W^1) - P(W^2) > \beta$, the first candidate is considered as the correct word and is replaced by the wrong word. To attain the optimal value for β, the identical process employed in the previous condition with the 5K dataset is applied, with the parameter α being set to a fixed value of -25. The best β value is found as 0.33. One significant benefit of employing this condition is that it allows a sentence to retain its original structure unless a superior word replacement is identified. Consequently, this condition aids in mitigating the likelihood of inadvertently substituting the correct word with an incorrect one.

If none of the specified conditions materialize or only one condition is met, the spell checker presents the top three ranked items to the user. In such a situation, the user retains the autonomy to select the most appropriate word. By evaluating test sets mentioned in Section 4, it has been determined that 98 percent of the time, the correct word can be found within the initial three ranked words.

3.5 Real-word error detection

Real-word errors, while potentially present in dictionaries, often lack appropriate meanings within the given context, rendering them more challenging to identify. Consequently, relying solely on a dictionary is not a viable approach for their detection. A sequential binary labeling model is used to detect this kind of error. The input sentence is tokenized to words $X = (x_1, x_2, ..., x_n)$ and then a contextualized embedding is calculated using ParsBERT (Farahani et al., 2021), which is BERT-based model for Persian. Next, embeddings are passed through a dense layer followed by a softmax layer.

The output of the model is $L = (l_1, l_2, ..., l_n)$, where l_i denotes the correctness of the token i. Label 1 shows that the i^{th} word is incorrect and has real-word error, while label 0 shows that the i^{th} word is correct.

For training the mentioned real-word error detector model a synthetically generated data is used. This synthetic data generation process involved the extraction of clean data from the aforementioned 80GB corpus. To ensure data cleanliness, we specifically selected sentences in which all constituent words were found within the detection dictionary.

The foundation of this data generation effort was the creation of a confusion matrix, utilizing the detection dictionary, to identify word pairs (w_i, w_j) with an edit distance of one or two between them. In each such pair, w_i is assigned as the correct word, while w_j is assigned as the incorrect word. On average, this confusion matrix comprised approximately 36 words for each word w_i. Subsequently, in the process of data synthesis, whenever a word w_i was encountered in a clean sentence, it was systematically replaced with word w_j. As a result, roughly 15% of the words in each sentence were substituted with incorrect alternatives.

Every pair (w_i, w_j) is employed in a minimum of 10 distinct sentences. This minimum requirement of ten substitutions is crucial, as having a smaller number of (w_i, w_j) pairs would hinder the model's training process. This is because, with fewer pairs, certain substitutions may result in sentences that remain correct and even convey identical meanings. T When the substitution process is repeated, the change in the meaning of sentences is guaranteed. For instance, the correct sentence «پرهیز از کژی و کاستی در میان پژوهش» is changed to «پرهیز از کژی و کاستی در بیان پژوهش» while both sentences are correct, a single substitution example is not enough to properly train the model. 80% of the synthetically generated data is used for training, while the rest is kept for validation. We used the dataset published in (Mirzababaei et al., 2013) as a test set.

3.6 Real-word suggestion list

Similar to non-word suggestion list in real-word errors, we cover insertion, deletion, substitution, and transportation errors. In addition, we cover homophone and spoonerism errors. Homophone words have the same pronunciation but different spelling. Spoonerism happens when words with more than two syllables have their first characters or syllables transposed. The pair of words («یارانه» , «رایانه») is an example of spoonerism.

Suggested words in real-word errors based on insertion, deletion, substitution, and transposition differ from non-word suggestion errors. The difference is that all of the generated words are not considered as suggested words, and some of them are removed from the list. For removing words, some rules based on part of speech are considered. Consider the example «کره‌ی زمین گرد است.» (Translation: "The earth is round."), this sentence has a positive verb «است» which cannot be changed to a negative verb «نیست» (In Persian, adding the character «ن» to the start of the verb, makes it negative.) By having the negative word in the suggestion list, the spelling of the sentence is correct, but it is not factually correct. One way to overcome this problem is to remove these words from the suggestion list.

3.7 Real-word error correction.

The model of real-word error detection is similar to non-word error corrector, and the only difference is the two conditions that are used for auto-correcting errors.

1- Probability of a sentence is greater than a threshold. If $P(C) > \alpha$, the candidate word might be replaced by the incorrect word, just like with non-word mistakes. A 10k subset of dataset used for real-word error detection ensuring that each pair of (w_i, w_j) is utilized no more than twice is used to calculate alpha. The optimal result for alpha is -7. The reason behind this value is that if the confidence level of an error is very high, auto-correction must be performed; hence, the barrier for this error type is much stricter.

2- The difference between the probability of the first and original words is greater than a threshold. The logarithm of the probability of the top-ranked candidate is represented by the $P(W^1)$, and the logarithm of the probability of the original word is represented by $P(W^O)$. When $P(W^1) - P(W^O) > \beta$, auto-correction takes place. We have two additional hyperparameters, word length, and sentence length; depending on their various values, β is determined. The mentioned 10k dataset is used for determining the value of β. Table 1 displays the precise value of β depending on the two mentioned hyperparameters.

Conditions	β
Sentence length less than 6	4
Word length equal to 1	20
Word length equal to 2	3
Word length equal to 3	2.2
Other	2

Table 1: Hyperparameters and model details of real-word error detection.

3.8 Post-processing

In the final step, punctuations are placed in their correct position, and the same as in the pre-processing step, ZWNJ errors are corrected. Finally, we have gathered a list of wrong words and their corresponding correct words from (Oji et al., 2021b) and suggest or auto-correct them.

4 Experimental Results

4.1 Dataset

We used six test set to compare our methodology with other spell checkers. Some of the datasets are modified because they do not consider Hamza, Tanvin, or ZWNJ as a type of error. The modified version is uploaded in a github repository.[4] Zarebin[5], Nevise news content [5], Nevise news title [5], Shargh [5], PerSpellData (Oji et al., 2021b) and real-word error (Mirzababaei et al., 2013) are test sets that contain both non-word and real-word data but they have different rates. As shown in Table 2 selected test sets have a variety of sentence lengths and error rates that cover different scenarios that can happen in a sentence. All of the test sets also have errors that real users make, and they are not fake errors.

4.2 Baselines

We used the following methodologies as our baselines for comparison.

[4]https://github.com/rominaoji/modified-spellchecker-testset

[5]https://github.com/Dadmatech/Persian-spell-checkers-comparison

Nevise[6] is a deep learning model trained on 30M parallel datasets. Nevise has 2 versions, and the second version[7] is used. **Paknevis**[8] is a Persian AI-based spell checker that suggests 4 words as the correct candidate, and we consider the first one as the correct word. **Google API** on Google Docs is used as another spell checker. The errors are fixed using the default settings. **Farsiyar**[9] is an AI-based spell checker that suggests different words as correct word, and the first ranked word is chosen as the correct word. **Xfspell** (Hagiwara, 2021) is a character level transformer based spell checker that is trained on the Persian parallel dataset of PerspellData (Oji et al., 2021b). To improve the performance of Xfspell, the backtranslation method is used for generating artificial noisy data. Xfspell github[10] code is used for training this spell checker.

4.3 Experiment Setting

As evaluation metrics, we used precision, recall, $F_{0.5}$, are used for both detection and correction, and correct to the wrong ratio is used for correction. We used $F_{0.5}$ since precision is more important than recall in the spell checker task because it is important not to change the correct word to the wrong word.

BertForTokenClassification of PyTorch library is used for implementing real-word error detection, and hyperparameters are fine-tuned. Hyperparameters and details are shown in Table 3 .

4.4 Results

Table 4 represents the experimental results of five spell checkers and Virastman on the six datasets. In all of the test sets for all of the metrics of precision, recall, and $F_{0.5}$ Visratman error correction has the best performance, and that is because of covering real-word, hamza, tanvin, and ZWNJ or word-boundary errors. Based on Table 2, 66.19% of errors are real-word errors, and the performance of Virastman on real-word errors is more significant, which

[6]https://neviise.ir/

[7]https://neviise.ir/service.html

[8]https://chrome.google.com/webstore/detail/paknevis-ai-based-persian/pklcojlgnoaahjchjbiilfgjehinajmd?hl=fa

[9]https://text-mining.ir/landing/virastar]

[10]https://github.com/mhagiwara/xfspell

Test set	Sentence Numbers	Average of Words	Total Errors	Average of Errors	Non-word Errors		Real-word Errors		space or ZWNJ Errors	
					No.	%	No.	%	No.	%
Shargh	223	8.56	451	2.01	422	93.77	29	6.23	304	67.40
PerSpellData	1127	12.90	1362	1.20	1337	98.16	25	1.48	247	18.13
Zarebin	1033	3.53	1470	1.42	1304	88.70	166	11.30	20	1.36
Nevise News Content	451	26.96	2586	5.73	2216	85.69	370	14.31	983	38.01
Nevise News Title	19,421	11.09	16,751	1.28	14572	86.99	2179	13.01	5731	34.21
Real-word	1100	16.16	1470	1.43	533	33.81	1043	66.19	408	25.88

Table 2: test set sentence and errors statistics

Hyperparameter	**Value**
Optimizer	AdamW
Loss Function	Cross-Entropy
Learning Rate	$2e-5$
Epsilon	$1e-3$
Epoch	2
Batch Size	16
Scheduler	Linear

Table 3: Hyperparameters and model details of real-word error detection.

means other spell checkers do not have a good performance on correcting real-word errors.

On the other hand, Virastman has a higher value in recall of error correction, which means that it detects more errors than other spell checkers; however, in most of them, the precision is lower than Nevise, which means that some of the words that detected as errors are correct. This can also be seen in the correct-to-wrong rate, in which, most of the time, Virastman has a higher percentage than Nevise. Investigation showed that this happens because the dictionary for detecting errors does not cover all the forms of words and needs to be extended. All in all, even if Viratsman detects more words as errors, it has better performance in the detection of errors since it corrects more errors than other spell checkers.

4.5 Threshold Impact on Virastman

As mentioned in Section 3.4, we correct the word if we have high confidence that it is wrong, and this is done by considering thresholds. For this experiment, we removed the real-word error detection and correction of Virastman, and we applied other parts of the Virastman to the Shargh dataset with and without thresholds. As illustrated in Table 5, the model without threshold the correct words that are regarded as wrong is increased, and this increases the recall and reduces the precision, while in spell checker, precision is more crucial than recall.

4.6 Impact of adding detected word to suggestion list

In Section 3.4, it was stated that adding the detected word to the suggestion list reduces the ratio of converting the correct to the wrong word. To investigate this, we use the non-word Virastman with the same parameters two times with and without adding the detected word to the suggestions list. As shown in Table 5, by adding the detected word to the suggestion list, recall reduces (0.44% in detection and 0.88% in error correction), but precision, which is more important, increased significantly (2.92% for detection and 1.12% for correction). The ratio of converting the correct to the wrong word was also reduced by 0.85%.

5 Conclusion

This paper notes a sequence labeling model for real-word error detection that needs less data than other methods, such as using a machine translation as a spell checker. An unsupervised model based on the sentence and word probability is used for non-word and real-word error correction. Experimental results show a significant improvement in the output of Visratman in comparison with other baselines.

Test set	Spell Checkers	Correct to Wrong Rate	Error Detection			Error Correction		
			Recall	Precision	$F_{0.5}$	Recall	Precision	$F_{0.5}$
Shargh	Virastman	0.64	**91.13**	97.86	**96.44**	**85.81**	**92.12**	**90.08**
	Nevise	**0.28**	78.94	**98.89**	94.13	65.41	81.94	78.00
	Paknevis	3.54	78.71	87.64	85.70	61.86	68.89	67.36
	Google	0.77	40.13	94.27	74.24	33.48	78.65	61.94
	FarsiYar	1.96	72.17	93.26	88.11	52.39	67.70	63.96
	XFspell	1.49	52.76	91.89	80.03	37.47	65.25	56.82
PerSpell Data	Virastman	0.40	**93.61**	96.00	**95.52**	**90.09**	**92.39**	**91.91**
	Nevise	**0.33**	79.59	**96.10**	92.27	72.17	87.15	83.68
	Paknevis	0.03	84.36	72.86	74.90	72.47	62.59	64.34
	Google	0.84	69.75	89.62	84.79	66.81	85.85	81.22
	FarsiYar	0.91	84.80	90.59	89.37	77.02	82.27	81.16
	XFspell	1.28	82.97	86.99	86.16	74.82	78.44	77.69
Zarebin	Virastman	**0.06**	**92.86**	**99.92**	**98.43**	**90.95**	**97.88**	**96.41**
	Nevise	**0.06**	90.07	**99.92**	97.78	82.38	91.40	89.44
	Paknevis	2.85	88.84	96.45	94.83	81.70	88.70	87.21
	Google	0.30	91.97	99.63	98.00	90.20	97.72	96.12
	FarsiYar	0.42	86.33	99.45	96.52	74.83	86.21	83.67
	XFspell	2.26	87.55	97.13	95.05	78.16	86.72	84.87
Nevise Content	Virastman	0.50	**84.34**	98.10	**95.01**	**80.47**	**93.61**	**90.65**
	Nevise	**0.27**	75.14	**98.83**	92.97	64.31	84.59	79.57
	Paknevis	2.93	74.44	88.63	85.38	61.06	72.70	70.03
	Google	1.08	68.63	95.12	88.30	61.43	85.15	79.05
	FarsiYar	2.30	67.21	89.96	84.26	49.38	66.10	61.91
	XFspell	1.24	75.17	94.88	90.15	68.41	86.33	82.03
Nevise Title	Virastman	0.48	**87.87**	96.58	**94.70**	**84.42**	**92.78**	**90.98**
	Nevise	**0.30**	73.62	**97.38**	91.47	64.26	85.01	79.85
	Paknevis	2.78	74.84	80.58	79.36	61.44	66.15	65.15
	Google	0.56	54.93	93.79	82.16	50.99	87.07	76.27
	FarsiYar	1.28	68.63	86.08	81.91	54.12	67.88	64.60
	XFspell	1.21	70.69	89.70	85.12	67.69	85.89	81.51
Real-word Errors	Virastman	**0.01**	**88.71**	**97.56**	**95.65**	**87.17**	**95.88**	**94.00**
	Nevise	0.52	28.43	84.53	60.61	25.06	74.53	53.45
	Paknevis	3.30	19.8	37.64	31.89	15.42	29.31	24.84
	Google	0.80	35.66	81.69	64.93	33.19	76.02	60.42
	FarsiYar	1.14	12.55	52.66	32.14	10.91	45.74	27.92
	XFspell	0.77	49.68	86.62	75.41	48.54	84.63	73.67

Table 4: Performance of different spell checkers on different test sets.

Spell Checker Errors	Correct to Wrong Rate	Error Detection			Error Correction		
		Recall	Precision	$F_{0.5}$	Recall	Precision	$F_{0.5}$
Non-Word Virastman with threshold and detected word	**0.21**	86.03	**99.22**	**96.28**	80.27	92.58	89.82
Non-word Virastman without thresholds	0.35	86.25	98.72	95.95	80.93	**92.64**	**90.03**
Non-word Virastman without detected word	1.06	**86.47**	96.30	94.16	**81.15**	90.36	88.36

Table 5: Effect of thresholds in Virastman non-word error detection and correction

6 Limitations

Virastman error detection in the part of non-word errors detects correct words as wrong words, and a larger dictionary is needed to reduce the ratio of converting correct words to wrong words, and this can happen by investigating and collecting the words added to the dictionary by Virastman users. One other weakness of Virastman is that in the correction phase, it cannot distinguish the character «ا» and «آ» and this is because of the tokenizer of ParsBERT. Training a new tokenizer can solve this problem.

References

Dimitrios Alikaniotis and Vipul Raheja. 2019. The unreasonable effectiveness of transformer language models in grammatical error correction. *arXiv preprint arXiv:1906.01733*.

Christopher Bryant and Ted Briscoe. 2018. Language model based grammatical error correction without annotated training data. In *Proceedings of the thirteenth workshop on innovative use of NLP for building educational applications*, pages 247–253.

Mohammad Dehghani and Heshaam Faili. 2023. Persian typographical error type detection using many-to-many deep neural networks on algorithmically-generated misspellings. *arXiv preprint arXiv:2305.11731*.

Heshaam Faili, Nava Ehsan, Mortaza Montazery, and Mohammad Taher Pilehvar. 2016. Vafa spell-checker for detecting spelling, grammatical, and real-word errors of persian language. *Digital Scholarship in the Humanities*, 31(1):95–117.

Mehrdad Farahani, Mohammad Gharachorloo, Marzieh Farahani, and Mohammad Manthouri. 2021. Parsbert: Transformer-based model for persian language understanding. *Neural Processing Letters*, 53:3831–3847.

Masato Hagiwara. 2021. *Real-world natural language processing: practical applications with deep learning*. Simon and Schuster.

Shivalila Hangaragi, Peeta Basa Pati, and N Neelima. 2023. Accuracy comparison of neural models for spelling correction in handwriting ocr data. In *Proceedings of Fourth International Conference on Communication, Computing and Electronics Systems: ICCCES 2022*, pages 229–239. Springer.

Daniel Hládek, Ján Staš, and Matúš Pleva. 2020. Survey of automatic spelling correction. *Electronics*, 9(10):1670.

Yifei Hu, Xiaonan Jing, Youlim Ko, and Julia Taylor Rayz. 2020. Misspelling correction with pre-trained contextual language model. In *2020 ieee 19th international conference on cognitive informatics & cognitive computing (icci* cc)*, pages 144–149. IEEE.

Sai Muralidhar Jayanthi, Danish Pruthi, and Graham Neubig. 2020. Neuspell: A neural spelling correction toolkit. *arXiv preprint arXiv:2010.11085*.

Mojtaba Khallash, Ali Hadian Cefidekhanie, and Behrouz Minaei-Bidgoli. 2013. An empirical study on the effect of morphological and lexical features in persian dependency parsing. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 97–107.

Yanen Li. 2020. Query spelling correction. *Query Understanding for Search Engines*, pages 103–127.

Shulin Liu, Shengkang Song, Tianchi Yue, Tao Yang, Huihui Cai, TingHao Yu, and Shengli Sun. 2022. Craspell: A contextual typo robust approach to improve chinese spelling correction. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 3008–3018.

Nora Madi and Hend Al-Khalifa. 2020. Error detection for arabic text using neural sequence labeling. *Applied Sciences*, 10(15):5279.

Behzad Mirzababaei, Heshaam Faili, and Nava Ehsan. 2013. Discourse-aware statistical machine translation as a context-sensitive spell checker. In *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013*, pages 475–482.

Roger Mitton. 1987. Spelling checkers, spelling correctors and the misspellings of poor spellers. *Information processing & management*, 23(5):495–505.

Romina Oji, Seyedeh Fatemeh Razavi, Sajjad Abdi Dehsorkh, Alireza Hariri, Hadi Asheri, and Reshad Hosseini. 2021a. Parsinorm: A persian toolkit for speech processing normalization. In *2021 7th International Conference on Signal Processing and Intelligent Systems (ICSPIS)*, pages 1–5. IEEE.

Romina Oji, Nasrin Taghizadeh, and Heshaam Faili. 2021b. Perspelldata: An exhaustive parallel spell dataset for persian. In *Proceedings of the Second International Workshop on NLP Solutions for Under Resourced Languages (NSURL 2021) co-located with ICNLSP 2021*, pages 8–14.

Joseph J Pollock and Antonio Zamora. 1984. Automatic spelling correction in scientific and scholarly text. *Communications of the ACM*, 27(4):358–368.

Shunmuga Priya, D Karthika Renuka, and L Ashok Kumar. 2022. Towards improving speech recognition model with post-processing spell correction using bert. *Journal of Intelligent & Fuzzy Systems*, 43(4):4873–4882.

Alex Wang and Kyunghyun Cho. 2019. Bert has a mouth, and it must speak: Bert as a markov random field language model. *arXiv preprint arXiv:1902.04094*.

Damar Zaky and Ade Romadhony. 2019. An lstm-based spell checker for indonesian text. In *2019 international conference of advanced informatics: concepts, theory and applications (ICAICTA)*, pages 1–6. IEEE.

Shaohua Zhang, Haoran Huang, Jicong Liu, and Hang Li. 2020. Spelling error correction with soft-masked bert. *arXiv preprint arXiv:2005.07421*.