A context-sensitive real-time Spell Checker with language adaptability

Prabhakar Gupta
Amazon
prabhgup@amazon.com

***Abstract*—We present a novel language adaptable spell checking system that detects spelling errors and suggests context-sensitive corrections in real-time. We show that our system can be extended to new languages with minimal language-specific processing. Available literature majorly discusses spell checkers for English but there are no publicly available systems that can be extended to work for other languages out of the box. Most of the systems do not work in real-time. We explain the process of generating a languages word dictionary and n-gram probability dictionaries using Wikipedia-articles data and manually curated video subtitles. We present the results of generating a list of suggestions for a misspelled word. We also propose three approaches to create noisy channel datasets of real-world typographic errors. Finally, we show the effectiveness of language adaptability of our proposed system by extending it to 24 languages.**

***Index Terms*—spell checker, auto-correct, n-grams, tokenizer, context-aware, real-time**

I. Introduction

Spell checker and correction is a well-known and well-researched problem in Natural Language Processing [1]–[4]. However, most state-of-the-art research has been done on spell checkers for English [5], [6]. While there has been research done on spell checkers for individual languages, but there has not been as extensive research in spell checkers that can be adapted to other languages. People have tried to make spell checkers for individual languages: Bengali [7], Czech [8], Danish [9], Dutch [10], Finnish [11], French [12], [13], German [14], [15], Greek [16], Hindi [17], [18], Indonesian [19], Marathi [20], Polish [21], Portuguese [22], Russian [23], [24], Spanish [25], Swedish [26], Tamil [27], Thai [28], etc. This is due to the fact languages are very different in nature and pose different challenges making it difficult to have one solution that work for all languages [29]. Many systems do not work in real-time cases. There are some rule-based spell checkers (like LanguageTool[1]) which try to capture grammar and spelling rules [30], [31]. This is not scalable and requires language expertise to add new rules. Another problem is evaluating the performance of the spell check system for each language due to lack of quality test data. Spelling errors are classified in two categories [32]: *non-word errors* where the word is unknown and *real-word errors* where the word itself is correct but used in a wrong form / context.

We present a context-sensitive real-time spell-checker system which can be adapted to any language. One of the biggest problems earlier was the absence of publicly available misspelling data for languages other than English, so we propose three approaches to create noisy channel datasets of real-world typographic errors. We use Wikipedia data for creating dictionaries and synthesizing test data. To compensate for the resource-scarcity of most languages we also use manually curated video subtitles since it provides information about how people communicate as shown in [33].

Our system outperforms industry-wide accepted English spell checkers (Hunspell and Aspell) and shows our performance on benchmark datasets for English. We present our performance on the synthetic dataset for 24 languages *viz.,* Bengali, Czech, Danish, Dutch, English, Finnish, French, German, Greek, Hebrew, Hindi, Indonesian, Italian, Marathi, Polish, Portuguese, Romanian, Russian, Spanish, Swedish, Tamil, Telugu, Thai and Turkish. We also compare our system to one of the most popular rule-based systems. We did not customize our spell checker to suit local variants or dialects of a language. For example — the spelling *"color"* is used in American English whereas spelling *"colour"* is preferred in other versions of English. Our system will not flag any of these spellings.

The paper makes the following contributions:

- We propose three different approaches to create typographic errors for any language which has never been done in a multilingual setting (all earlier approaches have either been very simple [17] or language-specific [20]).
- We show the system's time performance for each step in the process, proving it's real-time effectiveness.
- Our system outperforms existing rule-based and industry-wide accepted spell checking tools.
- We show that our system can be adapted to other languages with minimal effort — showing precision@k for $k \in 1, 3, 5, 10$ and mean reciprocal rank (MRR) for 24 languages.

The paper is divided into four sections. Section II explains the preprocessing steps and approaches to generate a ranked list of suggestions for any detected error. Section III presents different synthetic data-generation algorithms. Section IV describes the experiments and reports their results. Finally, section V concludes the paper and discusses future endeavours.

[1] www.languagetool.org

978-1-7281-6332-1/20/$31.00 ©2020 IEEE
DOI 10.1109/ICSC.2020.00023

II. Approach

Our system takes a sentence as input, tokenizes the sentence, identifies misspelled words (if any), generates a list of suggestions and ranks them to return the top k corrections. For ranking the suggestions, we use n-gram conditional probabilities. As a preprocessing step, we create word frequency dictionaries that will aid in the generation of n-gram conditional probabilities.

A. Preprocessing: Building n-gram dictionaries

We calculated unigram, bigram and trigram frequencies of tokens from the corpus. Using these frequencies, we calculated conditional probabilities expressed in the equation 1 where P are conditional probability and c is the count of the n-gram in the corpus. For unigrams, we calculate its probability of occurrence in the corpus.

$$P(w_i|w_{i-n+1}...w_{i-1}) = \frac{c(w_{i-n+1}...w_i)}{c(w_{i-n+1}...w_{i-1})} \quad (1)$$

We used Wikipedia dumps[2] along with manually curated video subtitles for all languages. We capped Wikipedia articles to 1 million and subtitle files to 10K. On average, each video subtitle file contains 688 subtitle blocks and each block contains 6.4 words [33]. We considered words of minimum length 2 with a frequency more than 5 times in the corpus. Similarly, only bigrams and trigrams where each token was known were considered.

One issue we encountered while building these dictionaries using such a huge corpus was its size. For English, the number of unique unigrams was approx. $2.2M$, bigrams was $50M$ and trigrams was $166M$. If we store these files as uncompressed Python Counters, these files end up being $44MB$, $1.8GB$ and $6.4GB$ respectively. To reduce the size, we compressed these files using a word-level Trie with hashing. We created a hash map for all the words in the dictionary (unigram token frequency) assigning a unique integer id to each word. Using each word's id, we created a trie-like structure where each node represented one id and its children represented n-grams starting with that node's value. The Trie ensured that the operation to lookup an n-gram was bounded in *O(1)* and reduced the size of files by 66% on an average. For English, the hashmap was $14MB$, unigram probabilities' file was $8.7MB$, bigram was $615MB$ and trigram was $2.5GB$.

B. Tokenization

There are a number of solutions available for creating a tokenizer for multiple languages. Some solutions (like [34]–[36]), try to use publicly available data to train tokenizers, whereas some solutions (like Europarl preprocessing tools [37]) are rule-based. Both approaches are not extensible and typically are not real-time.

For a language, we create list of supported characters using writing systems information[3] and Language recognition charts[4]. We included uppercase and lowercase characters (if applicable) and numbers in that writing system, ignoring all punctuation. Any character which doesn't belong to this list is implied as a foreign character to that language and will be tokenized as a separate token. Using regex rule, we extract all continuous sequences of characters in the supported list.

C. Error Detection

We kept our error-search strictly to non-words errors; for every token in a sentence, we checked for its occurrence in the dictionary. However, to make the system more efficient, we only considered misspelled tokens of length greater than 2. On manual analysis of Wikipedia misspellings dataset for English, we discovered misspelling of length 1 and 2 do not make sense and hence computing suggestions and ranking them is not logical.

D. Generating candidate suggestions

Given an unknown token, we generated a list of all known words within an edit distance (ED) of 2, calling them candidate suggestions. We present the edit distance distribution of publicly available datasets for English in figure 3. Two intuitive approaches to generate the list of suggestions that work fairly well on a small-size dataset are checking edit-distance of incorrect spelling with all words in the dictionary and second, generating a list of all words in edit-distance 2 of incorrect spelling[5]. The obvious problem with the first approach is with the size of the corpus which is typically in the range of hundreds of thousands and with the second approach is the size of the word because for longer words there can be thousands of suggestions and building a list of such words is also time-consuming.

We considered four approaches — Trie data structure, Burkhard-Keller Tree (BK Tree) [38], Directed Acyclic Word Graphs (DAWGs) [39] and Symmetric Delete algorithm (SDA)[6]. In table I, we represent the performance of algorithms for edit distance 2 without adding results for BK trees because its performance was in the range of a couple of seconds. We used Wikipedia misspelling dataset[7] to create a list of 2062 unique misspellings of lengths varying from 3 to 16 which were not present in our English dictionary. For each algorithm, we extracted the list of suggestions in edit distance of 1 and 2 for each token in the dataset.

E. Ranking suggestions

Using SDA, we generate a list of candidates that are to be ranked in order of relevance in the given context. Authors of [40], demonstrate the effectiveness of n-grams for English to auto-correct real-word errors and unknown word errors. However, they use high-order n-grams in isolation. We propose a weighted sum of unigrams, bigrams, and trigrams to rank the suggestions. Authors in [41], use character embeddings to

[2]Wikimedia Downloads: https://dumps.wikimedia.org

[3]https://en.wikipedia.org/wiki/List$_o f_l anguages_b y_w riting_s ystem$

[4]https://en.wikipedia.org/wiki/Wikipedia:Language$_r ecognition_c hart$

[5]https://norvig.com/spell-correct.html

[6]https://github.com/wolfgarbe/SymSpell

[7]https://en.wikipedia.org/wiki/Wikipedia:Lists$_o f_c ommon_m isspellings$

TABLE I
AVERAGE TIME TAKEN BY SUGGESTION GENERATION ALGORITHMS (EDIT DISTANCE = 2) (IN MILLISECOND)

Token	Trie	DAWGs	SDA
3	170.50	180.98	112.31
4	175.04	178.78	52.97
5	220.44	225.10	25.44
6	254.57	259.54	7.44
7	287.19	291.99	4.59
8	315.78	321.58	2.58
9	351.19	356.76	1.91
10	379.99	386.04	1.26
11	412.02	419.55	1.18
12	436.54	443.85	1.06
13	473.45	480.26	1.16
14	508.08	515.04	0.97
15	548.04	553.49	0.66
16	580.44	584.99	0.37

generate embeddings for each misspelling for clinical free-text and then similar to [42], rank based on contextual similarity score.

We create a context score (S) for each suggestion and rank on decreasing order of that score, returning top k suggestions. Context score is weighted sum of unigram context score (S_1), bigram context score (S_2) and trigram context score (S_3) defined by equation 2. This score is calculated for each suggestion by replacing the token x_i with the suggestion. For n-grams where any token is unknown, the count is considered to be 0.

$$S_n = W_n \sum_{j=0}^{n-1} \frac{c(x_{i+j-n+1}^{i+j})}{c(x_{i+j-n+1}^{i+j-1})} = W_n \sum_{j=0}^{n-1} P(x_i | x_{i+j-n+1}^{i+j-1}) \quad (2)$$

where:
i = index of misspelled token
W_n = the weight for n^{th}-gram's score
$c(x_i^j)$ = occurrence frequency of sequence $(w_i \dots w_j)$
P = conditional probability.

III. SYNTHESIZING SPELLING ERRORS

The biggest challenge in the evaluation of spell checkers was the quality test dataset. Most of the publicly available datasets are for English [43]. We propose three strategies to introduce typographical errors in the correct words to represent noisy channels. We select all the sentences, where we did not find any spelling error and introduced exactly one error per sentence.

A. Randomized Characters

From a sentence, we pick one word at random and make one of the three edits: insertion, deletion or substitution with a random character from that language's supported character list. Since it is a completely randomized strategy, the incorrect words created are not very "realistic". For example — in English for edit distance 2, word *"moving"* was changed to *"moviAX"*, *"your"* to *"mouk"*, *"chest"* to *"chxwt"*. We repeated the process for edit distance 1 (introducing only one error) and edit distance 2 (introducing two errors) and create a dataset for 20,000 sentences each.

B. Characters Swap

On analyzing common misspellings for English [43], we discovered the majority of edit-distance 2 errors are a swap of two adjacent characters. For example — *"grow"* is misspelled as *"gorw"*, *"grief"* as *"greif"*. One swap implies edit distance of two, we created a dataset of 20,000 samples for such cases.

C. Character Bigrams

Introducing errors randomly produces *unrealistic* words. To create more realistic errors, we decided to use character bigram information. From all the words in the dictionary for a language, we calculate occurrence probabilities for character bigrams. For a given word, we select a character bigram randomly and replace the second character in a selected bigram with a possible substitute from pre-computed character bigram probabilities. This way, we were able to generate words that were more plausible. For example — in English for edit distance 1, word *"heels"* was changed to *"heely"*, *"triangle"* to *"triajgle"*, *"knee"* to *"kyee"*. On shallow manual analysis of generated words, most of the words look quite realistic. For English, some of the words generated are representative of keyboard-strokes error (errors that occur due to mistakenly pressing a near-by key on keyboard/input device). For example, we generated some samples like — *"Allow"* to *"Alkow"*, *"right"* to *"riggt"*, *"flow"* to *"foow"* and *"Stand"* to *"Stabd"*. We generated a sample of 40,000 sentences each for edit distance 1 and edit distance 2.

IV. EXPERIMENTS AND RESULTS

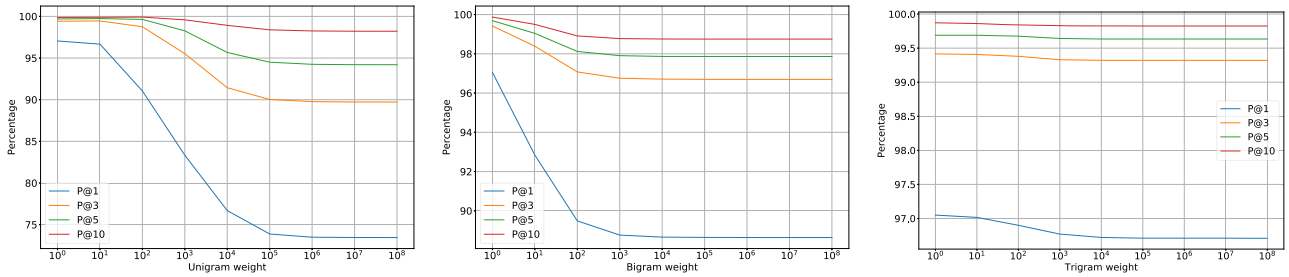
A. Synthetic Data evaluation

For each language, we created a dataset of 140,000 sentences with one misspelling each. The best performances for each language are reported in table II. We present Precision@k[8] for $k \in 1, 3, 5, 10$ and mean reciprocal rank (MRR). The system performs well on the synthetic dataset with a minimum of 80% *P@1* and 98% *P@10*.

The system can do each sub-step in real-time; the average time taken to perform for each sub-step is reported in table III. All the sentences used for this analysis had exactly one error according to our system. Detection time is the average time-weighted over the number of tokens in query sentence, suggestion time is weighted over misspelling character length and ranking time is weighted over the length of suggestions generated.

Table IV presents the system's performance on each error generation algorithm. We included only *P@1* and *P@10* to show trend on all languages. "Random Character" and "Character Bigrams" includes data for edit distance 1 and 2 whereas "Characters Swap" includes data for edit distance 2. Table V presents the system's performance individually on edit distance 1 and 2. We included only *P@1*, *P@3* and *P@10* to show trend on all languages.

[8]Percentage of cases where expected output was in top k results



(a) Variation of unigram weight (W_1) (b) Variation of bigram weight (W_2) (c) Variation of trigram weight (W_3)

Fig. 1. Importance of n-grams weights towards system accuracy

TABLE II
SYNTHETIC DATA PERFORMANCE RESULTS

Language	P@1	P@3	P@5	P@10	MRR
Bengali	91.3	97.8	98.9	99.7	94.7
Czech	95.8	98.7	99.3	99.6	97.4
Danish	85.8	95.2	97.3	98.8	90.9
Dutch	86.8	95.0	97.0	98.7	91.3
English	97.1	99.4	99.7	99.9	98.3
Finnish	97.8	99.6	99.8	99.9	98.7
French	86.5	95.7	97.5	98.8	91.4
German	87.6	96.2	97.9	99.1	92.1
Greek	85.0	95.0	96.9	98.4	90.3
Hebrew	94.0	98.3	99.1	99.6	96.2
Hindi	82.2	93.7	96.3	98.3	88.4
Indonesian	95.0	99.0	99.5	99.8	97.0
Italian	89.9	97.3	98.5	99.4	93.8
Marathi	93.0	98.2	99.1	99.7	95.7
Polish	95.7	99.2	99.6	99.9	97.4
Portuguese	86.7	96.3	97.9	99.1	91.7
Romanian	95.5	98.8	99.3	99.7	97.2
Russian	94.9	98.7	99.3	99.7	96.9
Spanish	85.9	95.4	97.2	98.6	90.9
Swedish	88.9	96.4	98.0	99.1	92.9
Tamil	98.1	99.7	99.9	99.9	98.9
Telugu	97.1	99.7	99.9	99.9	98.4
Thai	98.7	99.7	99.8	99.9	99.2
Turkish	97.1	99.5	99.8	99.9	98.3

TABLE III
SYNTHETIC DATA TIME PERFORMANCE RESULTS
DETECTION TIME IN μS, SUGGESTION AND RANKING TIME IN MS

Language	Detection Time	Suggestion Time		Ranking Time
		ED=1	ED=2	
Bengali	7.20	0.48	14.85	1.14
Czech	7.81	0.75	26.67	2.34
Danish	7.28	0.67	23.70	1.96
Dutch	10.80	0.81	30.44	2.40
English	7.27	0.79	39.36	2.35
Finnish	8.53	0.46	15.55	1.05
French	7.19	0.82	32.02	2.69
German	8.65	0.85	41.18	2.63
Greek	7.63	0.86	25.40	1.87
Hebrew	22.35	1.01	49.91	2.18
Hindi	8.50	0.60	18.51	1.72
Indonesian	12.00	0.49	20.75	1.22
Italian	6.92	0.72	29.02	2.17
Marathi	7.16	0.43	10.68	0.97
Polish	6.44	0.64	24.15	1.74
Portuguese	7.14	0.66	28.92	2.20
Romanian	10.26	0.63	18.83	1.79
Russian	6.79	0.68	22.56	1.72
Spanish	7.19	0.75	31.00	2.41
Swedish	7.76	0.83	32.17	2.57
Tamil	11.34	0.23	4.83	0.31
Telugu	6.31	0.29	7.50	0.54
Thai	11.60	0.66	18.75	1.33
Turkish	7.40	0.49	17.42	1.23

We experimented with the importance of each n-gram. Figure 1 presents the results for this experiment. We kept two weights constant varying one weight to compare the performance. For example to determine unigram weight (W_1) importance, we set bigram weight (W_2) and trigram (W_3) to 1, varying W_1 ($10^i, i \in [0, 8]$). As shown in figure 1(a) and figure 1(b), if unigram or trigram are given more importance, the performance of system worsens. Figure 1(c) shows removing lower order n-grams and giving more importance to the only trigram also decreases performance. Therefore, finding the right balance between each weight is crucial for the system's best performance.

B. Comparison with LanguageTool

We compared the performance of our system with one of the most popular rule-based systems, LanguageTool (LT) for 11 languages *viz.,* Danish, Dutch, French, German, Greek, Polish, Portuguese, Romanian, Russian, Spanish and Swedish. As shown in figure 2, LT doesn't detect any error in many cases. For example — for German, it did not detect any error in 42% sentences and for 25% (8% (No Match) + 17% (Detected more than one error)), it detected more than one error in a sentence out of which in 8% sentences, the error detected by our system was not detected by LT. Only in 33% of sentences LT detected exactly one error which was the same as detected by our system. Results for Portuguese seem very skewed which can be due to the fact Portuguese has two major versions, Brazilian Portuguese (pt-BR) and European Portuguese (pt-PT); LT has a different set of rules for both versions whereas dataset used was a mix of both.

TABLE IV
SYNTHETIC DATA PERFORMANCE ON THREE ERROR GENERATION ALGORITHM

Language	Random Character		Characters Swap		Character Bigrams	
	P@1	P@10	P@1	P@10	P@1	P@10
Bengali	91.24	99.49	82.58	99.17	93.69	99.87
Czech	94.04	99.26	91.56	99.15	97.80	99.91
Danish	84.61	98.44	71.81	97.16	90.10	99.44
Dutch	85.33	98.45	72.80	96.68	91.16	99.31
English	97.26	99.90	93.22	99.70	98.05	99.88
Finnish	97.74	99.86	94.51	99.69	98.68	99.97
French	84.33	98.48	72.57	97.22	91.17	99.41
German	86.87	98.88	73.92	97.55	91.45	99.51
Greek	82.55	97.80	71.93	96.91	90.29	99.39
Hebrew	94.18	99.67	88.49	99.20	95.41	99.71
Hindi	81.61	97.64	67.73	96.20	86.27	99.17
Indonesian	94.74	99.84	89.04	99.56	96.75	99.91
Italian	88.87	99.14	78.77	98.27	93.40	99.78
Marathi	92.39	99.49	85.15	99.03	95.45	99.91
Polish	94.92	99.74	90.28	99.71	97.45	99.95
Portuguese	86.42	98.90	71.74	97.69	90.79	99.56
Romanian	94.93	99.58	90.81	99.25	97.12	99.85
Russian	93.29	99.50	89.00	99.24	97.20	99.94
Spanish	84.54	98.21	71.35	96.65	90.40	99.25
Swedish	87.20	98.87	76.94	97.65	92.83	99.66
Tamil	98.12	99.99	96.92	99.99	99.28	99.99
Telugu	97.32	99.99	93.94	99.99	97.90	99.99
Thai	97.99	99.76	97.24	99.45	98.86	99.99
Turkish	97.05	99.88	93.20	99.82	98.26	99.97

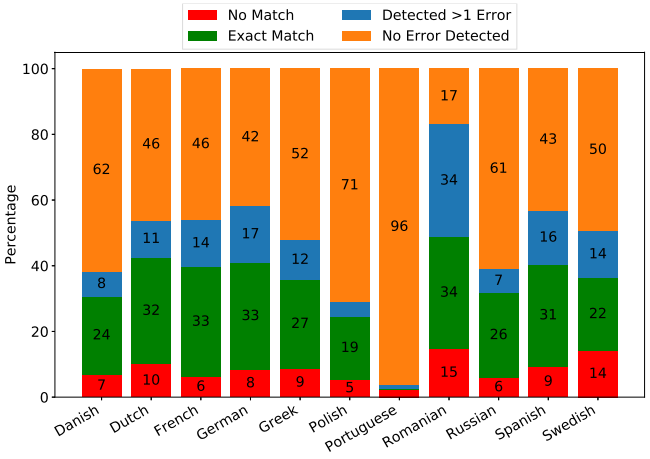


Fig. 2. Performance Comparison with LT for 11 languages

C. Public Datasets results

We used four publicly available datasets for English — **birkbeck**: contains errors from Birkbeck spelling error corpus[9], **hollbrook**: contains spelling errors extracted from passages in book, *English for the Rejected*, **aspell**: errors collected to test GNU Aspell[10] [44], **wikipedia**: most common spelling errors on Wikipedia. Each dataset had a list of misspelling and the corresponding correction. We ignored all the entries which had more than one tokens. We extracted 5,987 unique correct words and 31,589 misspellings. Figure 3(a) shows the distribution of edit distance between misspelling and its correction. Figure 3(b) shows the same distribution excluding **birkbeck** dataset leaving 2,081 unique words and 2,725 misspellings.

[9]http://ota.ox.ac.uk/
[10]http://aspell.net/

TABLE V
SYNTHETIC DATA PERFORMANCE ON DIFFERENT EDIT DISTANCE OF ERRORS

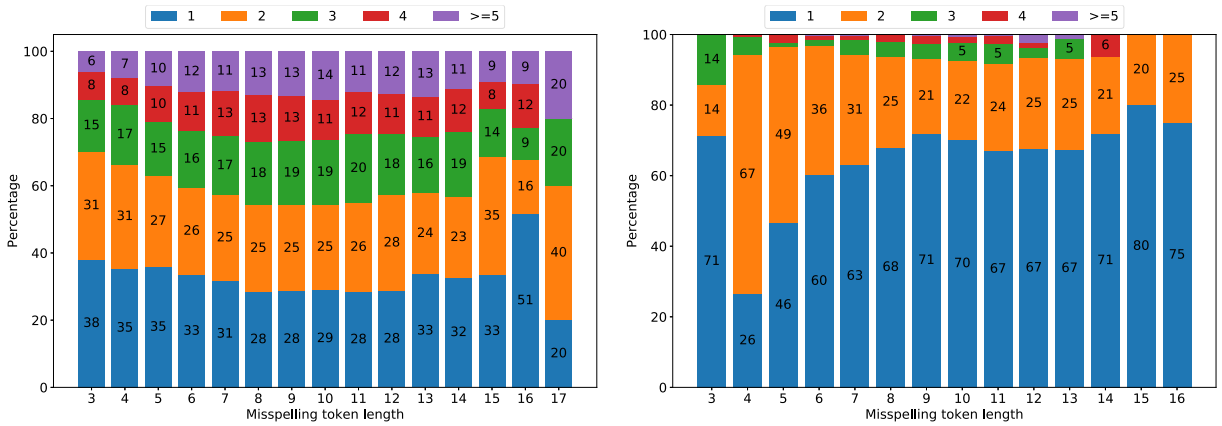
Language	ED=1	ED=2	
	P@1	P@1	P@3
Bengali	97.5	86.6	96.3
Czech	98.9	93.0	97.6
Danish	95.9	78.3	91.8
Dutch	96.2	79.8	91.5
English	99.3	95.4	99.0
Finnish	99.4	96.5	99.3
French	95.6	79.7	92.7
German	96.6	80.9	93.4
Greek	95.0	76.1	91.0
Hebrew	97.6	90.2	96.9
Hindi	93.1	73.7	89.3
Indonesian	98.7	92.1	98.2
Italian	95.8	84.6	95.4
Marathi	96.3	89.5	96.8
Polish	96.9	93.2	98.6
Portuguese	95.9	79.9	93.6
Romanian	98.7	93.2	97.9
Russian	97.6	92.3	97.9
Spanish	95.2	79.0	92.1
Swedish	96.9	82.8	93.9
Tamil	97.1	98.2	99.8
Telugu	96.0	95.7	99.4
Thai	97.0	97.8	99.5
Turkish	98.6	95.5	99.2

birkbeck dataset is the biggest out of four but the quality of this dataset is questionable. As explained by the dataset owners, the dataset is created using poor resources. From figure 3(b), our assumption of most of the common misspelling being in maximum edit-distance of 2 is correct.

TABLE VI
PUBLIC DATASET COMPARISON RESULTS

	P@1	P@3	P@5	P@10
Aspell	60.82	80.81	87.26	91.35
Hunspell	61.34	77.86	83.47	87.04
Ours	68.99	83.43	87.03	90.16

We use every correct and incorrect token in this dataset to check if they are present and absent in our dictionary respectively to prove if our detection system can detect correctness/incorrectness of tokens efficiently. The detection system was able to detect 99.13% of correct tokens and 88.37% of incorrect tokens accurately. The percentage of incorrect token detection is comparatively low is because there are many tokens in the dataset which were correct but were added in misspelling dataset — *"flower"*, *"representative"*, *"mysteries"*, etc. Some correct words in the dataset which were detected incorrectly were also noise due to the fact some words start with a capital letter but in the dataset, they were in lowercase — *"beverley"*, *"philippines"*, *"wednesday"* etc. Comparison of most popular spell checkers for English



(a) (b) without **birbeck**

Fig. 3. Edit distance distribution for Public English Datasets

(GNU Aspell and Hunspell[11]) on this data is presented in table VI. Since these tools only work on the word-error level, we used only unigram probabilities for ranking. Our system outperformed both the systems.

D. False Positive evaluation

For a spell checker system, false positives are when spelling error is detected but there was none. We experimented with a mix of three public datasets — **OpenSubtitles dataset** [45], **OPUS Books dataset** [46] and **OPUS Tatoeba dataset** [46] to generate a dataset with minimum 15,000 words for each of 24 languages. Since these datasets are human-curated, we can safely assume every token should be detected as a *known* word.

As shown in table VII, most of the words for each language were detected as known but still there was a minor percentage of words that were detected as errors. For English, the most frequent errors in the complete corpus were either proper nouns or foreign language words — *"Pencroft"*, *"Oblonsky"*, *"Spilett"*, *"Meaulnes"* and *"taient"*. This proves the effectiveness of the system against false positives.

TABLE VII
FALSE POSITIVE EXPERIMENT RESULTS
NUMBER OF WORDS IN THOUSANDS AND PERCENTAGE ACCURACY

Language	# words (in k)	Accuracy
Bengali	457	97.05
Czech	37	97.90
Danish	103	98.95
Dutch	1048	95.80
English	4982	98.52
Finnish	43	92.02
French	3244	98.25
German	1283	97.43
Greek	43	97.79
Hebrew	505	97.85
Hindi	38	98.85
Indonesian	84	98.18
Italian	719	97.88
Marathi	84	94.76
Polish	34	95.78
Portuguese	26	99.56
Romanian	35	97.79
Russian	384	97.06
Spanish	2057	98.61
Swedish	66	97.67
Tamil	21	92.79
Telugu	18	96.60
Thai	68	73.69
Turkish	794	97.69

V. CONCLUSION

We presented a novel context-sensitive spell checker system that works in real-time. Most of the available literature majorly discuss spell checkers for English and sometimes for some European (like German, French) and Indian languages (like Hindi, Marathi), but there are no publicly available systems (non-rule based) which can work for all languages.

Our proposed system outperformed industry-wide accepted spell checkers (GNU Aspell and Hunspell) and rule-based spell checkers (LanguageTool). First, we proposed three different approaches to create typographic errors for any language which has not been done earlier in a multilingual setting. Second, we divide our proposed system into 5 steps — Preprocessing; tokenization; error detection; candidate suggestion generation; and suggestion ranking. We used n-gram conditional probability dictionaries to understand the context to rank suggestions and present top suggestions.

We showed the adaptability of our system to 24 languages using precision@k for $k \in 1, 3, 5, 10$ and mean reciprocal rank (MRR). The system performs at a minimum of 80% *P@1* and 98% *P@10* on synthetic dataset. We showed the robustness of our system to false-positives. In the future, we can further increase the support for real-word errors and compound word errors.

[11]http://hunspell.github.io/

REFERENCES

[1] C. Whitelaw, B. Hutchinson, G. Chung, and G. Ellis, "Using the web for language independent spellchecking and autocorrection," in *EMNLP*, 2009.

[2] Q. Chen, M. Li, and M. Zhou, "Improving query spelling correction using web search results," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007.

[3] J. Gao, X. Li, D. Micol, C. Quirk, and X. Sun, "A large scale ranker-based system for search query spelling correction," in *Proceedings of the 23rd International Conference on Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 358–366.

[4] P. Gupta, M. Sharma, K. Pitale, and K. Kumar, "Problems with automating translation of movie/tv show subtitles," *ArXiv*, vol. abs/1909.05362, 2019.

[5] M. Choudhury, M. Thomas, A. Mukherjee, A. Basu, and N. Ganguly, "How difficult is it to develop a perfect spell-checker? a cross-linguistic analysis through complex network approach," 2007.

[6] M. D. Dunlop and J. Levine, "Multidimensional pareto optimization of touchscreen keyboards for speed, familiarity and improved spell checking," in *CHI*, 2012.

[7] M. Islam, M. Uddin, M. Khan *et al.*, "A light weight stemmer for bengali and its use in spelling checker," 2007.

[8] M. Richter, P. Straňák, and A. Rosen, "Korektor–a system for contextual spell-checking and diacritics completion," *Proceedings of COLING 2012: Posters*, pp. 1019–1028, 2012.

[9] E. Bick, "A constraint grammar based spellchecker for danish with a special focus on dyslexics," *A Man of Measure: Festschrift in Honour of Fred Karlsson on his 60th Birthday. Special Supplement to SKY Jounal of Linguistics*, vol. 19, pp. 387–396, 2006.

[10] A. M. Bosman, S. de Graaff, and M. A. Gijsel, "Double dutch: The dutch spelling system and learning to spell in dutch," in *Handbook of orthography and literacy*. Routledge, 2013, pp. 149–164.

[11] T. Pirinen, K. Lindén *et al.*, "Finite-state spell-checking with weighted language and error models," in *Proceedings of LREC 2010 Workshop on creation and use of basic lexical resources for less-resourced languages*, 2010.

[12] M. Starlander and A. Popescu-Belis, "Corpus-based evaluation of a french spelling and grammar checker." in *LREC*, 2002.

[13] T. Fontenelle, "Developing a lexicon for a new french spell-checker," 2006.

[14] A. Rimrott and T. Heift, "Evaluating automatic detection of misspellings in german," *Language Learning & Technology*, vol. 12, no. 3, pp. 73–92, 2008.

[15] G. Kodydek, "A word analysis system for german hyphenation, full text search, and spell checking, with regard to the latest reform of german orthography," in *TSD*, 2000.

[16] G. Petasis, V. Karkaletsis, D. Farmakiotou, G. Samaritakis, I. Androutsopoulos, and C. Spyropoulos, "A greek morphological lexicon and its exploitation by a greek controlled language checker," in *Proceedings of the 8th Panhellenic Conference on Informatics*, 2001, pp. 8–10.

[17] P. Etoori, M. Chinnakotla, and R. Mamidi, "Automatic spelling correction for resource-scarce languages using deep learning," in *Proceedings of ACL 2018, Student Research Workshop*, 2018, pp. 146–152.

[18] S. Kabra and R. Agarwal, "Auto spell suggestion for high quality speech synthesis in hindi," *CoRR*, vol. abs/1402.3648, 2014.

[19] M. Y. Soleh and A. Purwarianti, "A non word error spell checker for indonesian using morphologically analyzer and hmm," in *Proceedings of the 2011 International Conference on Electrical Engineering and Informatics*. IEEE, 2011, pp. 1–6.

[20] V. Dixit, S. Dethe, and R. K. Joshi, "Design and implementation of a morphology-based spellchecker for marathi, and indian language," *ARCHIVES OF CONTROL SCIENCE*, vol. 15, no. 3, p. 301, 2005.

[21] R. Grundkiewicz, "Automatic extraction of polish language errors from text edition history," in *International Conference on Text, Speech and Dialogue*. Springer, 2013, pp. 129–136.

[22] R. T. Martins, R. Hasegawa, M. d. G. V. Nunes, G. Montilha, and O. N. De Oliveira, "Linguistic issues in the development of regra: A grammar checker for brazilian portuguese," *Natural Language Engineering*, vol. 4, no. 4, pp. 287–307, 1998.

[23] A. Sorokin and T. Shavrina, "Automatic spelling correction for russian social media texts," in *Proceedings of the International Conference Dialog(Moscow*, 2016, pp. 688–701.

[24] A. Sorokin, "Spelling correction for morphologically rich language: a case study of russian," in *Proceedings of the 6th Workshop on Balto-Slavic Natural Language Processing*, 2017, pp. 45–53.

[25] F. R. Bustamante and E. L. Díaz, "Spelling error patterns in spanish for word processing applications." in *LREC*. Citeseer, 2006, pp. 93–98.

[26] V. Kann, R. Domeij, J. Hollman, and M. Tillenius, "Implementation aspects and applications of a spelling correction algorithm," *Text as a Linguistic Paradigm: Levels, Constituents, Constructs. Festschrift in honour of Ludek Hrebicek*, vol. 60, pp. 108–123, 2001.

[27] T. Dhanabalan, R. Parthasarathi, and T. Geetha, "Tamil spell checker," in *Sixth Tamil Internet 2003 Conference, Chennai, Tamilnadu, India*, 2003.

[28] T. Karoonboonyanan, V. Sornlertlamvanich, and S. Meknavin, "A thai soundex system for spelling correction," in *Proceeding of the National Language Processing Pacific Rim Symposium*, 1997, pp. 633–636.

[29] A. Helfrich and B. Music, "Design and evaluation of grammar checkers in multiple languages," in *COLING*, 2000.

[30] D. Naber *et al.*, *A rule-based style and grammar checker*. Citeseer, 2003.

[31] M. Miłkowski, "Developing an open-source, rule-based proofreading tool," *Software: Practice and Experience*, vol. 40, no. 7, pp. 543–566, 2010.

[32] K. Kukich, "Techniques for automatically correcting words in text," *ACM Comput. Surv.*, vol. 24, pp. 377–439, 1992.

[33] P. Gupta, S. Shekhawat, and K. Kumar, "Unsupervised quality estimation without reference corpus for subtitle machine translation using word embeddings," in *2019 IEEE 13th International Conference on Semantic Computing (ICSC)*, Jan 2019, pp. 32–38.

[34] E. Moreau and C. Vogel, "Multilingual word segmentation: Training many language-specific tokenizers smoothly thanks to the universal dependencies corpus," in *LREC*, 2018.

[35] B. Snyder and R. Barzilay, "Unsupervised multilingual learning for morphological segmentation," in *ACL*, 2008.

[36] P.-C. Chang, M. Galley, and C. D. Manning, "Optimizing chinese word segmentation for machine translation performance," in *WMT@ACL*, 2008.

[37] P. Koehn, "Europarl: A parallel corpus for statistical machine translation," 2005.

[38] W. A. Burkhard and R. M. Keller, "Some approaches to best-match file searching," *Commun. ACM*, vol. 16, pp. 230–236, 1973.

[39] M. Balík, "Implementation of directed acyclic word graph," *Kybernetika*, vol. 38, pp. 91–103, 2002.

[40] A. Carlson and I. Fette, "Memory-based context-sensitive spelling correction at web scale," *Sixth International Conference on Machine Learning and Applications (ICMLA 2007)*, pp. 166–171, 2007.

[41] P. Fivez, S. Suster, and W. Daelemans, "Unsupervised context-sensitive spelling correction of clinical free-text with word and character n-gram embeddings," in *BioNLP*, 2017.

[42] H. Kilicoglu, M. Fiszman, K. Roberts, and D. Demner-Fushman, "An ensemble method for spelling correction in consumer health questions," *AMIA ... Annual Symposium proceedings. AMIA Symposium*, vol. 2015, pp. 727–36, 2015.

[43] R. Grundkiewicz and M. Junczys-Dowmunt, "The wiked error corpus: A corpus of corrective wikipedia edits and its application to grammatical error correction," in *Advances in Natural Language Processing – Lecture Notes in Computer Science*, A. Przepirkowski and M. Ogrodniczuk, Eds., vol. 8686. Springer, 2014, pp. 478–490. [Online]. Available: http://emjotde.github.io/publications/pdf/mjd.poltal2014.draft.pdf

[44] S. Deorowicz and M. Ciura, "Correcting spelling errors by modelling their causes," 2005.

[45] P. Lison and J. Tiedemann, "Opensubtitles2016: Extracting large parallel corpora from movie and tv subtitles," in *LREC*, 2016.

[46] J. Tiedemann, "Parallel data, tools and interfaces in opus," in *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, N. C. C. Chair), K. Choukri, T. Declerck, M. U. Dogan, B. Maegaard, J. Mariani, J. Odijk, and S. Piperidis, Eds. Istanbul, Turkey: European Language Resources Association (ELRA), may 2012.