Spellchecker Analysis for Behavioural Biometric of Typing Errors Scenario

Bartłomiej Marek and Wojciech Wodo
Faculty of Information and Communication Technology, Wroclaw University of Science and Technology, Wybrzeze Wyspianskiego 27, Wroclaw, Poland

Keywords: Spellchecker, Behavioural Biometrics, Typing Errors, User Model, Typing, Keystroking, Authentication.

Abstract: Unlike the typical approach using keystroke dynamics for user authentication and identification, we focus on a more inherent characteristic - the pattern of typing mistakes, which are not widely investigated in the literature. The paper presents initial research that enables the selection of an appropriate Python-based spellchecker for detection in behavioural biometrics systems based on static text characteristics: typing errors. Integrating a robust spellchecker into a biometric system based on static features such as errors made during typing can significantly enhance its effectiveness and user experience. The study evaluated seven tools and their combinations, amounting to forty-nine variants. The research is split into two phases. The first one used fewer sentences to filter satisfying the criteria tools, for which, in the second phase, the context was expanded to be able to choose the most appropriate one by using more sentences. The ultimate goal of the research is to create different user behavioural models for typing errors and test them in the verification and identification scenarios. We will apply the most promising spellcheckers based on the current investigation results.

1 INTRODUCTION

As the paper (Mróz-Gorgoń et al., 2022) presents, the market for biometric-based solutions is rapidly growing, especially for security-oriented applications. Finding an easy-to-use and cost-effective solution based on the fundamental behavioural trait - typing is very tempting. Keystroking is a natural way of interaction between humans and their systems; that is the way it is almost transparent for the user, and there is no need to apply any additional sensors to get the data (Gupta et al., 2023).

The development of a reliable and robust biometric system based on typing errors would be an interesting endeavour, given the gap in the application of static features of human typing. To the best of our knowledge, the only notable application of the continuous authentication system that uses typing error features was introduced in the (Parappuram et al., 2016), which presented a high accuracy but lacked the dataset, technical specifics and no continuation of research was found. Hence, we initiate work on a behavioural biometrics system based on typing errors by selecting the most suitable spellchecker in the context of the planned application.

The primary objective of this research is to survey various Python-based tools available for use in spelling error correction, a key component of a behavioural biometrics system based on static text characteristics: typing errors. The biometric system aims to identify and verify users by error characteristics extraction from data gathered in real-time (*online mode*) or uploaded text files (*offline mode*). In contrast to most of the keystroke approaches, a system based on misspells uses static features that may bring other benefits, including reducing dependence on devices and continuous monitoring (in case of *offline mode*), ensuring stability and consistency in user authentication and identification, proposing an alternative or reinforcing a dynamic approach.

To pinpoint characteristics derived from typing errors, one must flawlessly detect and identify any miswriting within the text. A crucial aspect of detecting misspells is accuracy and precision in predictions of *what the author meant* and *how it should be written* while not generating false-positive errors. A spellchecker enhances static biometric features by identifying and correcting typing mistakes, reducing noise and improving the performance of the biometric system, thereby ensuring a more secure and efficient authentication method.

This paper evaluates tools and their combinations,

Marek, B. and Wodo, W.
Spellchecker Analysis for Behavioural Biometric of Typing Errors Scenario.
DOI: 10.5220/0012789000003767

In *Proceedings of the 21st International Conference on Security and Cryptography (SECRYPT 2024)*, pages 748-757
ISBN: 978-989-758-709-2; ISSN: 2184-7711

emphasising correctness based on defined string comparison metrics that can be divided into four groups. Each metric is calculated by comparing the output of the test variant and a given (arbitrarily correct) sentence. A secondary but important aspect considered is operational reliability, e.g. the operation time.

This paper evaluates tools and their combinations with an emphasis on correctness based on defined string comparison metrics that can be divided into four groups. Each metric is calculated by comparing the output of the test variant and a given (arbitrarily correct) sentence. A secondary but important aspect considered is operational reliability, e.g. the operation time.

1.1 Related Work

The problem of error correction has already been extensively researched, focusing on a number of approaches. In (Näther, 2020) 14 spelling error correction tools from various applications and environments were benchmarked. The authors split the test into error-free sentences and sentences with various numbers of errors from multiple error classes (such as adding non-existing words, changing a word to a similar one or repeating some words. Tools were evaluated using, e.g. sequence accuracy. As a result of the research, the authors noticed that all tools are effective at correcting non-words, but sentences with other error types, especially those that require context understanding, are challenging.

The authors of (Bexte et al., 2022) introduced LESPELL, a benchmark corpus of spelling errors for evaluating spellchecking methods for learner language. It discusses the challenges of spellchecking learner language and introduces DKPro Spelling as a solution. The article uses different dictionaries and corpora to compare the performance of different spellchecking tools, such as Hunspell and LanguageTool. Reranking candidates based on language model probabilities improves correction performance for most corpora, but the results vary depending on the language and corpus (Bexte et al., 2022).

1.2 Contribution

A developed module enables tools and their combinations testing, using sentences from a dataset modified with the most typical typing errors (Damerau, 1964). The review aims to identify the most suitable spell checker based on the defined experimental environment and evaluation criteria assessing accuracy, robustness and reliability. The main objective of this work is to select a spell checker for use in a biometric system. Through experimentation and assessment, valuable insights are provided into the strengths and weaknesses of each tool and their combinations[1].

2 RESEARCH METHODOLOGY

The effectiveness of the selected implementations was assessed through empirical testing to select the most effective program or combination of typing error correction tools from the assortment contained within the Python libraries. The main initial assumption is to establish a controlled environment to test each of the chosen variants.

2.1 Metrics

Two types of algorithms are applicable to compare strings: similarity, the bigger the value, the more similar strings are in reference to each other; and the distance (reverse similarity), opposite, the lower value, the more similar are strings in reference to each other. The classification of algorithm types, based on their characteristics, includes the following categories:

- Edit distance algorithms, which are determined by the number of editing operations applied to the given strings (Wu, 2021).
- Token-based methods, akin to set similarity algorithms, utilizing string tokens as their primary elements (Blurock, 2021a);
- Sequence-based algorithms which focus on common substrings or subsequences, where a 'sequence' is defined as a series of consecutive characters(Blurock, 2021a);
- Phonetic-based approaches are designed to index words according to their pronunciation.

Calculating string similarity and distance measures allows solving problems in e.g. correction or string recognition applications(Haque et al., 2022). However, they are very often costly to compute resources with uncertain performance due to the variety of potential errors.

2.1.1 Edit Based

The approach proposed by F.J. Damerau, one of the most popular and universally applied methods that relies on edit distance, posits that a significant majority of spelling errors are the result of a single miswriting.

[1]All datasets and codes are available in the git repository: https://github.com/BaarTeek123/python_spellcheckers_comparison

These errors can be corrected by applying one of the four categories:

- deletion that represents removing a character,
- insertion that stands for adding a character,
- replacement that implies a character change,
- transposition that represents swapping neighbouring characters,

thus expanding Levenshtein distance, including the first three from the above-mentioned (Damerau, 1964)(Levenshtein et al., 1966). Both metrics can be defined as the minimal amount of specified operations that must be executed to transform a string into a second one.

Another example of edit-based metrics is the Jaro-Winkler distance, which represents a modification of the Jaro distance developed by William E. Winkler. His change of the Jaro algorithm relies on using p prefix scale, which results in assigning higher ratings to strings that match from the start. This metric is normalized - a score of 0 means an exact match, and 1 indicates no resemblance. However, mathematically this algorithm cannot be considered as a metric due to not complying with triangle inequality (Jaro, 1989)(Cohen et al., 2003).

2.1.2 Token Methods

This category includes metrics based on sets of similarity algorithms (containing tokens in this case). To calculate such a metric, three main steps can be distinguished:

1. Definition of a set of tokens (strings),
2. Specification of the number of token occurrences in a text, for example, by counting the frequency of each token in the text,
3. Calculation of distance or similarity using relatively simple algorithms based on the computations from the previous phase (Blurock, 2021a).

An example measure is cosine similarity, which is the cosine of the angle between the vectors. The cosine similarity can take a value between -1 and 1, where 1 means that X, Y are the same, 0 indicates their orthogonality and -1 that they are opposite. However, it cannot be considered as an appropriate distance metric due to no fulfilment of the Schwarz inequality and the coincidence axiom (Singhal et al., 2001).

Another example of token methods metrics is the Sørensen-Dice coefficient that can be defined for given sets of discrete data: X, Y as:

$$DSC(X,Y) = \frac{2|X \cap Y|}{|X| + |Y|} (Sorensen, 1948) \quad (1)$$

The next instance token-based metric is the overlap coefficient (otherwise: Szymkiewicz-Simpson coefficient) that can be defined as:

$$overlap(X,Y) = \frac{|X \cap Y|}{min(|X|, |Y|)}, \quad (2)$$

where $min(|X|, |Y|)$ is the minimum of the number of tokens in X and the number of tokens in Y (Vijaymeena and Kavitha, 2016). Both the above are related to Jaccard index - having one of them, the other can be easily calculated (Dice, 1945)(Sorensen, 1948).

2.1.3 Sequence Based

The string can be compared using the longest common substring - with rising substring length, similarity increases(Blurock, 2021b). Another sequence-based metric is Ratcliff / Obershelp pattern recognition (otherwise: Gestalt Pattern Matching), which is assumed to be closer to human analysis (Cohen et al., 2003). It is defined as:

$$Gestalt\ similarity = \frac{2\,K_m}{|s_1| + |s_2|} (Blurock, 2021b) \quad (3)$$

, where: K_m is a number of matching characters that is calculated using recursive method to find the longest common substrings, $|s_1|$, $|s_2|$ are lengths of strings s_1, s_2 (Blurock, 2021b).

2.1.4 Phonetic Based

A separate category is methods using phonetics. Metrics from this group rely on evaluating the pronunciation of words. An exemplary algorithm is the Match Rating Approach developed by Western Airlines. In this approach, which performs well if the edit distance is less than two, word coding rules and comparative methods are used (Moore, 1977). For example, *Smyth* and *Smith* will be considered as matching.

The basis of the approach taken to measure the efficiency of misspelling correction is defined class *Distances* that attributes mostly based on the metric (Figure 1).

Another attribute of class Distance is a list of operations that keeps instances of *EditOperation* subclasses (Figure 1). Those error classes are defined according to Damerau's study (described in Section 2.1.1), expanding the context of each misspelling by providing additional information about miswriting. The base class *EditOperation* contains fields that enable identification of the type of string manipulation, related index, a preceding and a following letter. Derived classes specify each string operation using the following additional fields. As presented in

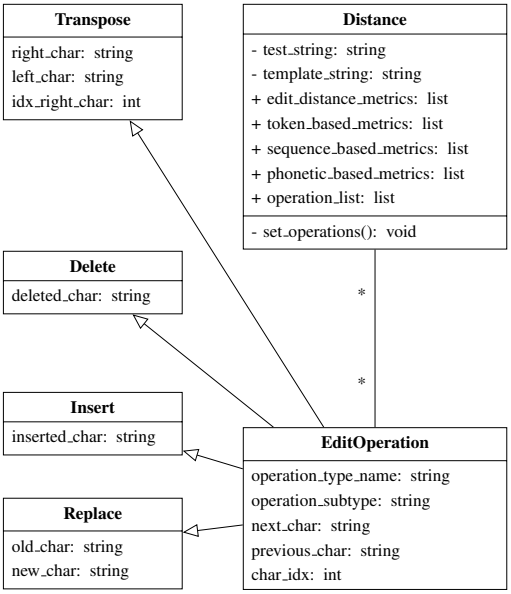

Figure 1: Distance class diagram.

Figure 1, the fields include the old and new characters for the Replace operation, an additional field for the inserted character for the Insert operation, the right character, the left character, and the right character's index for the Transpose operation, and lastly, the deleted character field for the Delete operation.

3 EXPERIMENTS

3.1 Scope

Several tools available in various Python modules were selected for testing purposes:

- *TextBlob* (from *textblob* module) that allows to correct spelling (Loria, 2018),
- *SpellChecker* (from *spellchecker* module) which is Peter Norivig's spell checking algorithm implementation(Barrus, 2022)(Norvig, 2016),
- *Speller* (from *autocorrect* module) which is a spellchecker(Jonas McCallum, 2021),
- *GingerIt* (from *gingerit* module) enabling simultaneous correction of grammar and spelling mistakes (Kleinschmdit, 2021),
- *LanguageTool* (from *language_tool_python module* using local server), the program allows correcting both grammar and spelling mistakes (Morris, 2022),
- *TSpellCorrector* - from *jamspell* module that offers fast, multi-language and accurate spell checking (Ozinov, 2020),
- *GramFormer* from the Gramformer - a library that provides three distinct interfaces to using a set of techniques to find, highlight and fix grammar mistakes (PrithivirajDamodaran, 2021).

Each of them is configured following the documentation. Apart from a simple approach of passing a sentence to the correction as an argument for the above tools, also the approach assumes providing the output from one tool to another as an argument (Algorithm 1). Utilizing a combination of two spellcheckers may enhance error detection and increase accuracy by leveraging each tool's unique strengths in identifying and correcting errors. However, this approach introduces complexity in implementation and integration, possibly leading to computational overhead that could impact the efficiency and performance of the biometric system, particularly in real-time applications. Additionally, combining spellcheckers might lead to over-correction, where one tool introduces new errors, potentially resulting in less accurate output.

The research examines tools by:

- passing the test sentence as an argument to assess tendencies towards over-correction and the capability of detecting and correcting erroneous mistakes. This approach also evaluates the potential increase in false-positive error generation, which could amplify the challenge of filtering out noise characteristics.
- passing phrases containing a predetermined number of misspellings based on error classes using QWERTY keyboard layout characters. This method evaluates the tools' performance in a more realistic scenario of typing errors, simulating instances where the incorrect key is pressed due to its proximity to the intended key on the keyboard layout.
- passing phrases misspelled with a specific number of random letters. This approach aims to assess how effectively spellcheckers handle random errors that do not conform to any specific keyboard layout pattern.

In each scenario, the types and locations of the misspellings are randomly selected, adhering to predefined classes of errors.

The first of the above groups aims to evaluate testing tools in the aspect of generating false positive errors - trying to correct a sentence without any misspells. This type of error may be harmful to the biometric system due to adding unwanted noise to data that is used to create a user model, which may affect its characteristics. The second one is based on QWERTY keyboard layout characteristics - relatively common is mistakenly typing an adjacent key. For

Algorithm 1: Correct sentence by a tool.

Input: *ts* ▷ tested sentence
Output: *cs* ▷ corrected sentence
if isStandaloneTool **then**
  cs = *correct sent* ← *ts*)
else
  cs = *correct sent toolA* ← *correct sent toolB* ← *ts*
end

this purpose, each printable key was mapped based on the QWERTY layout as presented in Figure 2 (having regard to left, right, upper and bottom keys). The operation type, an adjacent key (from the map) is chosen randomly. The last considered category includes completely random errors - operation type, position, and character are aleatory.

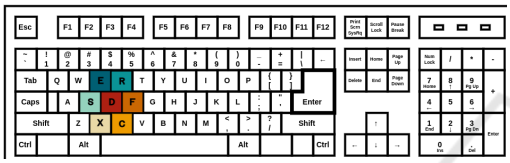

Figure 2: Key class instance example: character=*d*, left upper=*e*, right upper=*r*, left=*s*, right=*f*, left bottom=*x*, right bottom=*c*

Algorithm 2: Algorithm for creating misspells process.

Input: *s* ▷ sentence
op ▷ operation list
n ▷ number of errors
r ▷ randomly
Output: *missp_sent* ▷ *n*-times misspelled sentence

missp_sent ← s
while *n* > 0 **do**
  op = rand(len(op))
  if op = 0 **then**
    missp_sent ← replace(missp_sent, r)
  else if op = 1 **then**
    missp_sent ← delete(missp_sent, r)
  else if op = 2 **then**
    missp_sent ← insert(missp_sent, r)
  else
    missp_sent ← transpose(missp_sent, r)
  end
  n-=1
end

To systematically evaluate the effectiveness of correction tools and their combinations, the two-phase testing procedure has been designed - for each of them the process will be similar and includes comparing a template sentence with a tested one (that can be also a template sentence or a phrase with generated misspells according to Algorithm 2). In Phase 1 of the experiments, the primary objective is to identify the top *n* spellcheckers or combinations of two spellcheckers (49 in total) that will be assessed in the next stage. This approach enables to conduct of an efficient preliminary assessment of the individual spellcheckers and their combinations while obtaining meaningful insights into their performance across the three error types in a sentence provided as an argument according to Algorithm 3. In the second phase (Phase 2), after identifying the top *n* spellcheckers or combinations from the first phase, the amount of data used per variant will be increased. This growth in data will facilitate a more in-depth and comprehensive evaluation of the selected spellcheckers or combinations, allowing for assessing their effectiveness, robustness, and generalizability under more complex and realistic conditions.

By adopting this two-phase approach with varying amounts of data, a more efficient and thorough assessment of the spellcheckers is ensured, ultimately contributing to the identification of the most effective and reliable solutions for correcting typing errors in the context of a behavioural biometric system that relies on typing errors.

Algorithm 3: Algorithm for testing process.

Input: *sl* ▷ list of template sentences
nel ▷ list of amount of errors
vl ▷ list of tool variants
Output: *d_fpl* ▷ list of Distance(sl, sl) for each v in vl
d_kel ▷ list of Distance(sl, kbes) for each v in vl
d_rel ▷ list of Distance(sl, rbes) for each v in vl

foreach *t* ∈ *sl* **do**
  foreach n∈ *nel* **do**
    kbes = t ← *n* key based errors
    rbes = t ← *n* random errors
    foreach *v* ∈ *vl* **do**
      if n = 0 **then**
        t_corr = v ← t
        d_fpl[v] ← Distance(t, t_corr)
      kbes_corr = v ← kbes
      rbes_corr = v ← rbes
      d_kel[v] ← Distance(t, kbes_corr)
      d_rel[v] ← Distance(t, rbes_corr)
    end
  end
end

3.2 Phase 1

In the initial part of the experiments, a smaller dataset for each variant is used, considering the large number of variants being evaluated (49 in total). The performance of individual spellcheckers and their combinations was assessed across the three distinct testing methods: over-correction analysis, QWERTY-based typing errors, and random letter misspelling detection. In this nested tool architecture, $Tool\_outside(Tool\_inside)$, represents a sequential processing pipeline, where the output from $Tool\_inside$ is fed into $Tool\_outside$, and the performance is evaluated based on the latter's output

The preliminary stage used a comprehensive compilation of H.P. Lovecraft's publicly available works, sourced from the GitHub repository. As most of Lovecraft's creations are in the public domain, they are not restricted by copyright law. The dataset preserves the original text without significant modifications, which may necessitate further processing for analytical purposes (Gawarecki, 2017) (Nate Smith, 2013). As mentioned in the previous section (Section 3.1) the evaluation process encompasses three categories: false positives, QWERTY layout error-based, and random errors, where QWERTY and random divided into three groups depending on the n - number of errors in a sentence:

1. $n \in [1,3)$ (around 350 sentences per variant);
2. $n \in [3,6)$ (around 500 sentences per variant);
3. $n \geq 6$ (around 170 sentences per variant);

These subcategories will be referred to as QE_i for QWERTY and RE_i for random errors, where $i \in \{1,2,3\}$ and FP for non-error sentence (around 175 sentences per variant). The performance of the error correction variant is evaluated by comparing their outputs to the original template sentences for each subcategory - for which accuracy (A_m) is calculated for each metric:

$$A_m = \frac{Number\ of\ correct\ corrections}{Total\ number\ of\ correction}$$

, where m is a metric that is grouped into four categories (M_i): edit-based, phonetic-based, token-based, and sequence-based. Each of the metrics is considered equally important in Phase 1. It is reflected in the total score for each subcategory s_j which is a sum of accuracy in each metric:

$$Score = \sum_{mc \in \{M_i\}} \sum_{j=1}^{|mc|} m_j \ \ \forall i \in 1,2,3,4$$

, where M is a string comparison metrics, i group of metrics. For each subcategory individually, the top n values were assigned a $Points$ (denoted by the symbol p), and all other values received a score of 0.

Finally, a weighted sum was calculated using category-specific weights w to determine the total score for each error correction variant:

$$Total = \sum_{j=0}^{|S|} w_j s_j$$

,where $S = \{FP, RE_i, QE_i\}$, for $i \in \{1,2,3\}$

This evaluation process in Phase 1 allows for accurately assessing the performance of each spellchecker and combination across multiple categories and subcategories. However, selecting appropriate weights w for each category is essential in the evaluation process, as it significantly impacts the overall assessment of spellcheckers and their combinations. Real-world relevance and error frequency should be considered when determining weights for a balanced and meaningful evaluation. Furthermore, the choice of top n values for assigning points p may affect the selection of candidates advancing to Phase 2.

3.3 Phase 2

During Phase 2 of the experiments, the 15 identified spellcheckers and their combinations proceeded to an advanced evaluation, adhering to the approach employed in Phase 1 and described in Section 3.2. The dataset under consideration consists of approximately 1000 false positive sentences (for each variant), QWERTY layout error-based, and random errors, which both can be divided into three categories based on the variable n, which represents the number of errors in a sentence:

1. $n \in [1,3)$ (around 2000 sentences per variant);
2. $n \in [3,6)$ (around 3000 sentences per variant);
3. $n \geq 6$ (around 2000 sentences per variant);

that were created using the corpus of sentences previously utilized in the initial phase of research (Section 3.2).

The dataset under consideration extends the corpus of sentences previously utilized in the initial phase of research. Other sentences from the set as in Phase 1 (section 3.2) were used as the source dataset to create sentences with false positives, QWERTY layout error-based, and random errors, which QWERTY and random were divided into three categories based on the variable n, which represents the number of errors in a sentence.

However, in this phase, the points (P) will be assigned only to the top 5. To facilitate a comprehensive and in-depth analysis of variants' performance, a

more extensive dataset was employed for each of the spellcheckers or combinations. It allowed for a more accurate assessment of the spellcheckers' capabilities in handling misspells, consequently enabling them to make an informed and most appropriate decision in the context of biometrics behavioural system of typing errors. Moreover, in the case of choosing between the best spellcheckers, the analysis also contains time aspects and may include careful analysis of each part of performed experiments.

3.4 Results and Analysis

3.4.1 Results of Phase 1

To minimize the potential impact of choosing top n values and weights on the final results, as described in Section 3.1, the following strategy was used:

- Assign points corresponding to the place (15 points for the 1^{st} and 1 point for the 15^{th}) or more scaled point system for the top values and the rest rewarding the best to give a more detailed illustration of the performance variations between the tested variants.
- Assign equal weights to each subcategory S_i or prioritize subcategories based on their real-world importance - for the intended use as a part of a biometric system, the greatest importance is found in the correctness of correcting a few errors (RE_1, EQ_1)
- Penalties for below-median performance: introduce penalties for metrics that score below the median for a specific metric in a subcategory S_i. For example, a penalty of -0.5 could be applied to encourage better overall performance across all metrics.

Based on the aforementioned approach, a series of tests were conducted to evaluate the performance of various spellcheckers and their combinations. Following the assessment from Table 1, a classification was generated (Table 2), which allowed for the identification of the top-performing tools.

A summary of conducted experiments with different configurations is shown in the Table 3 :

- T_1 - $p_i \in P_1$, $w_i \in W$ with penalties,
- T_2 - $p_i \in P_2$, $w_i \in W$ with penalties,
- T_3 - $p_i \in P_1$, $w_i \in W$ without penalties,
- T_4 - $p_i \in P_2$, $w_i \in W$ without penalties,

, where $P_1 = \{30, 25, 22, 18, 15, 14...1\}$, $P_2 = \{15..1\}$, $W = \{0.8, 1, 1, 0.8, 0.8, 0.5, 0.5\}$. This consistent performance highlights the robustness and reliability of these selected tools in addressing various error correction challenges within biometric systems. The weights reward a lower number of errors due to the intended use - the full spelling corrections in sentences with more misspells are still more incidental events which in the case of the usage as a part of a biometric system is less important than providing meaningful and reliable correction process even for less number of misspells.

Notably, 15 tools were selected for further analysis, as they consistently ranked within the top 20 across all tested scenarios, irrespective of the specific weights, points, and penalties adopted (marked as green in Table 3).

Table 1: Score table for TOP 20 with the most default parameters (weights & points without any penalties).

Function\Subcategory	FP	QE1	RE1	QE2	RE2	QE3	RE3
SpellChecker(Gingerit)	4.91	3.58	3.58	1.73	1.85	0.47	0.62
Autocorrect(GramFormer)	5.49	3.41	3.12	1.76	1.53	0.48	0.76
SpellChecker(GramFormer)	4.95	3.23	3.23	1.56	1.60	0.41	0.62
Gingerit(GramFormer)	5.49	4.17	3.26	1.65	1.17	0.22	0.30
SpellChecker(LanguageTool)	5.19	3.10	3.34	1.30	1.43	0.41	0.51
GramFormer(Gingerit)	5.48	3.90	3.30	1.49	1.14	0.25	0.20
GramFormer(LanguageTool)	5.55	3.61	3.17	1.32	1.12	0.26	0.25
GramFormer(Autocorrect)	5.57	3.53	3.04	1.30	1.15	0.34	0.29
Gingerit(SpellChecker)	4.84	3.40	2.84	1.40	1.17	0.35	0.31
Gingerit(LanguageTool)	5.78	3.79	2.64	1.51	0.83	0.23	0.19
TextBlob(GramFormer)	4.40	2.57	2.45	1.28	1.27	0.39	0.54
Gingerit(Autocorrect)	6.28	3.71	2.38	1.28	0.73	0.31	0.16
GramFormer(SpellChecker)	4.77	3.08	3.09	1.25	1.23	0.34	0.45
LanguageTool(GramFormer)	5.48	3.10	2.98	1.46	1.13	0.23	0.46
Gingerit	6.45	3.71	2.28	1.13	0.64	0.20	0.14
Autocorrect(SpellChecker)	4.99	2.61	2.63	1.24	1.21	0.46	0.50
Jamspell(Gingerit)	6.44	3.69	2.27	1.13	0.64	0.20	0.14
Gingerit(Jamspell)	6.44	3.68	2.28	1.13	0.64	0.20	0.14
SpellChecker(Autocorrect)	4.97	2.58	2.79	1.08	1.19	0.36	0.46
Autocorrect(Gingerit)	6.28.	3.26	2.21	1.27	0.76	0.36	0.19

Table 2: Points table with Total for TOP 20 with the most default parameters (weights & points without any penalties).

Function\Subcategory	FP	QE1	RE1	QE2	RE2	QE3	RE3	Total
SpellChecker(Gingerit)	0	7	15	14	15	14	14	**59**
Autocorrect(GramFormer)	0	5	9	15	13	15	15	**51**
SpellChecker(GramFormer)	0	2	11	12	14	11	13	**46**
Gingerit(GramFormer)	0	15	12	13	6	0	0	**42**
SpellChecker(LanguageTool)	0	0	14	6	12	10	11	**39**
GramFormer(Gingerit)	0	14	13	10	4	0	0	**38**
GramFormer(LanguageTool)	0	8	10	7	2	0	0	**25**
GramFormer(Autocorrect)	0	6	7	5	5	5	0	**24**
Gingerit(SpellChecker)	0	4	3	8	7	6	0	**22**
Gingerit(LanguageTool)	0	13	0	11	0	0	0	**22**
TextBlob(GramFormer)	0	0	0	3	11	9	12	**22**
Gingerit(Autocorrect)	8	12	0	4	0	0	0	**22**
GramFormer(SpellChecker)	0	0	8	0	10	4	7	**22**
LanguageTool(GramFormer)	0	0	6	9	3	0	9	**20**
Gingerit	11	11	0	0	0	0	0	**20**
Autocorrect(SpellChecker)	0	0	0	0	9	13	10	**19**
Jamspell(Gingerit)	10	10	0	0	0	0	0	**18**
Gingerit(Jamspell)	9	9	0	0	0	0	0	**16**
SpellChecker(Autocorrect)	0	0	1	0	8	7	8	**15**
Autocorrect(Gingerit)	7	3	0	2	0	8	0	**14**

3.4.2 Results of Phase 2

The results of Phase 2 experiments provide important light on how well the top 15 spellchecker versions performed. The results from this phase provide a more thorough assessment of the spellcheckers' capabilities because the research was expanded and

Table 3: Summary of spellcheckers variants evaluation using different approaches (values of points, weights, and penalties). The brackets include the rank position for that particular configuration, based on which an average position (column *Avg P*) was calculated.

Function	T_1	T_2	T_3	T_4	Avg P
SpellChecker(Gingerit)	98 (1)	51,7 (1)	112 (1)	59,2 (1)	1
Autocorrect(GramFormer)	87,6 (2)	47,9 (2)	97,6 (2)	51,4 (2)	2
Gingerit(GramFormer)	62,4 (3)	36,6 (3)	75,2 (4)	42,2 (4)	3,5
GramFormer(Gingerit)	59 (4)	35,8 (4)	64,6 (6)	38,2 (5)	4,75
SpellChecker(GramFormer)	43 (8)	31,6 (7)	75,9 (3)	45,8 (3)	5,25
SpellChecker(LanguageTool)	57,4 (6)	31,8 (6)	62,5 (5)	38,9 (6)	5,75
GramFormer(Autocorrect)	57,5 (5)	31,9 (5)	48,1 (8)	23,5 (7)	6,25
Gingerit(LanguageTool)	42,6 (9)	24,4 (9)	34 (10)	21,8 (14)	10,5
GramFormer(LanguageTool)	29 (16)	19 (12)	42,8 (7)	25,2 (9)	11
Autocorrect(SpellChecker)	46,6 (7)	27,6 (8)	29 (16)	18,7 (18)	12,25
Gingerit(Autocorrect)	32 (13)	20,4 (11)	34,2 (12)	21,6 (13)	12,25
Gingerit(SpellChecker)	31,6 (14)	10 (23)	46,6 (9)	22 (8)	13,5
Gingerit	30 (15)	20,8 (10)	30 (15)	19,8 (17)	14,25
GramFormer(SpellChecker)	27,8 (17)	14 (21)	35,1 (13)	21,5 (12)	15,75
Autocorrect	36,6 (10)	18,5 (14)	24 (21)	12,2 (20)	16,25
Autocorrect(Gingerit)	34,1 (11)	15,5 (19)	32,8 (20)	14,2 (15)	16,25
Jamspell(Gingerit)	24,6 (21)	19 (12)	24,6 (17)	18 (19)	17,25
Gingerit(Jamspell)	27 (18)	16,2 (16)	24 (18)	16,2 (20)	18
SpellChecker(Autocorrect)	25,1 (19)	15,5 (19)	31,7 (19)	14,9 (16)	18,25
Autocorrect(TextBlob)	33,4 (12)	16,3 (15)	15,4 (25)	7,6 (25)	19,25
Jamspell(Autocorrect)	25,1 (19)	15,9 (17)	21,1 (23)	10,9 (23)	20,5
Jamspell	24 (22)	12 (22)	24 (22)	12 (20)	21,5

the amount of data collected increased. In Phase 2, the experiments and assessment method from Phase 1 were repeated (it only varied in that the number of points was limited to the top 5). As is shown in Table 4 and in Figure 3, four variants are distinguished by the best results: Autocorrect(GramFormer), SpellChecker(Gingerit), GramFormer(Autocorrect) and SpellChecker(GramFormer). However, upon delving deeper into the results of Phase 2, it can be observed that GramFormer(Autocorrect) is sensitive to penalties. That suggests being more prone to making more errors that exceed the median value, which is confirmed by analysis of the results of individual experiments.

Autocorrect(GramFormer), compared to other variants from the best four, exhibits several notable advantages also seen in other experiments. Firstly, in comparison to the other top 4 variants, Autocorrect(Gramformer):

- generates the least over-correction (slightly worst in this respect is GramFormer(Autocorrect)). Both of these variants are better than the rest of the top 4 ;
- stands out significantly in correcting a few misspells (QE_1, RE_1);
- is marginally worse than SpellChecker(GramFormer) in correction sentences from $QE2$, but both of them are more efficient than the others;
- is much worse than SpellChecker(Gingerit) in correction sentences with the highest numbers of misspells (QE_3, RE_3), but similar to the rest.

This high level of accuracy across subcategories contributes to its overall robustness and adaptability.

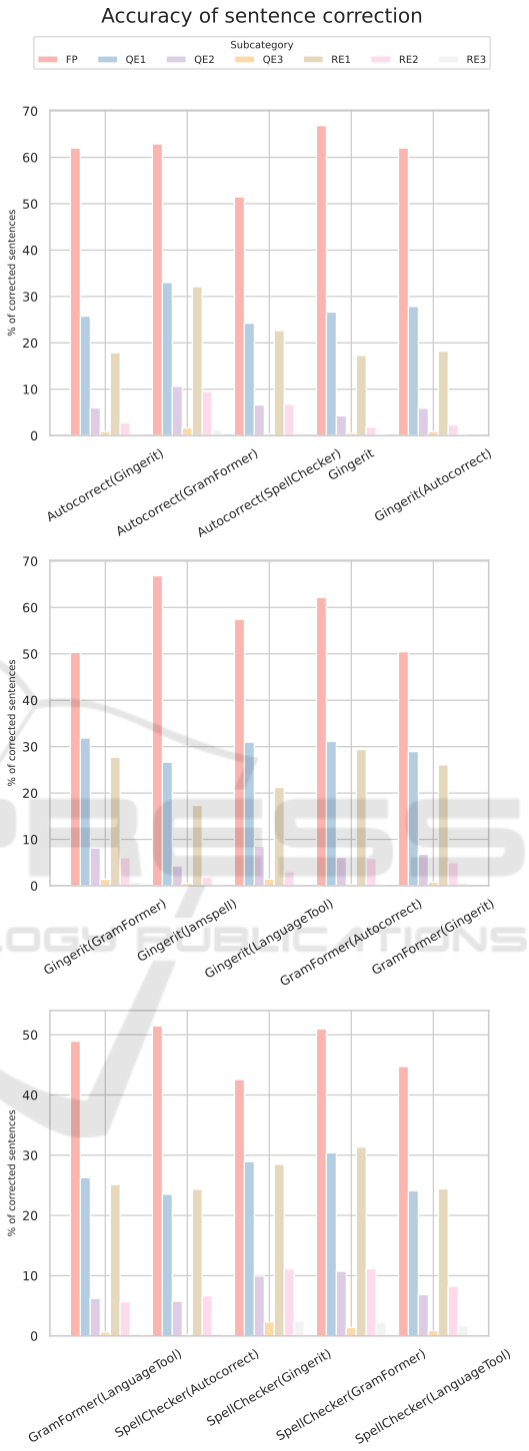

Figure 3: Accuracy of sentence correction.

Moreover, this spellchecker variant does not generate as many errors that are over the median value. This indicates that it can effectively handle spelling errors, assuring consistent performance and providing trustworthy results in a variety of spelling con-

texts. Also, as it can be seen in Table 5, the average time of the correction process performed by Autocorrect(GramFormer) is a little worse than SpellChecker(GramFormer), but both of them are, on average, faster than others variants.

Table 4: Summary of spellcheckers chosen best variants evaluation using same configurations and approaches as in Phase 1 differs only in sets of Points (P_1 and P_2) that were fit to top 5 - $P_1 = \{30, 25, 22, 18, 15\}$, $P_2 = \{5..1\}$

Function	T_1	T_2	T_3	T_4	Avg P
Autocorrect(GramFormer)	149,1 (1)	19,5 (1)	126,2 (1)	19,5 (1)	1
SpellChecker(Gingerit)	70,6 (3)	12,1 (3)	83,1 (3)	12,1 (3)	2,75
GramFormer(Autocorrect)	75,9 (2)	6,6 (4)	54,4 (4)	6,6 (4)	3,25
SpellChecker(GramFormer)	48,4 (5)	17,5 (2)	115,5 (2)	17,5 (2)	3,75
Gingerit(LanguageTool)	45 (6)	5,1 (6)	43,9 (6)	5,1 (6)	5,75
Autocorrect(SpellChecker)	59,5 (4)	2,8 (10)	30 (7)	2,8 (10)	6,25
Gingerit(GramFormer)	37 (8)	5,8 (5)	52 (5)	5,8 (5)	6,5
Gingerit	39 (7)	4 (7)	24 (9)	4 (7)	7,5
Gingerit(Jamspell)	20 (11)	3,2 (8)	20 (10)	3,2 (8)	9,5
Autocorrect(Gingerit)	21 (9)	0,8 (11)	12 (11)	0,8 (11)	10,25
SpellChecker(LanguageTool)	0 (13)	3,1 (9)	25,4 (8)	3,1 (9)	10,75
SpellChecker(Autocorrect)	21 (9)	0,5 (12)	7,5 (12)	0,5 (12)	10,75
Gingerit(Autocorrect)	7,5 (12)	0 (13)	0 (13)	0 (13)	12,5
GramFormer(Gingerit)	0 (13)	0 (13)	0 (13)	0 (13)	13
GramFormer(LanguageTool)	0 (13)	0 (13)	0 (13)	0 (13)	13

The time aspect is not the primary focus of the research, but in the aspect of usage as an integral part of a biometric system of typing errors, the time efficiency of these spellcheckers is still important, due to a direct impact on a user experience and overall system performance (especially in *Online* mode).

Table 5: Mean times in Phase 2.

Function	Mean Time [s]
Autocorrect(SpellChecker)	0,0597
SpellChecker(Autocorrect)	0,0816
SpellChecker(LanguageTool)	0,096
SpellChecker(GramFormer)	0,471
Autocorrect(GramFormer)	0,4862
Autocorrect(Gingerit)	0,5364
SpellChecker(Gingerit)	0,5443
GramFormer(Autocorrect)	0,581
Gingerit(Jamspell)	0,5945
GramFormer(LanguageTool)	0,603
Gingerit	0,6217
Gingerit(LanguageTool)	0,6233
Gingerit(Autocorrect)	0,6315
Gingerit(GramFormer)	1,1093
GramFormer(Gingerit)	1,116

4 CONCLUSIONS AND FUTURE WORKS

In the comprehensive analysis of various spellchecker variants within the context of a biometric behavioural system based on typing errors, the approach aimed to identify an effective spellchecker solution or their combination that also ensures robustness and solidity. Phase 1 enabled filtering out some variants, leaving the most promising ones for further analysis. The adopted approach and criteria revealed the most promising variants in combinations of different spellcheckers rather than standalone tools. It has been confirmed in Phase 2 in which each of the variants was tested on a larger sample of data. On this basis, four spellcheckers were identified. Despite the advantages of each of them, upon a deeper comparison, one combination of spellcheckers, *Autocorrect (Gramformer)*, emerged as the most solid and robust option among them.

In each phase, to minimize potential biases in the results, the experiments were conducted using different weights, and scoring methods and divided the evaluation into subcategories. It allowed assessing thoroughly the performance of the spellchecker combinations, ensuring the conclusions drawn were reliable and robust spellchecking solutions for biometric behavioural analysis of typing errors.

However, the arbitrary selection of points, weights, and penalties may lead to biases and the risk that complex factors affecting the spellcheckers' performance might be overlooked or oversimplified. This could cause some spellcheckers to be unfairly penalized or rewarded. The approach might not fully take into account the dynamic and multifaceted nature of the biometric behaviour of the typing error system. Real-world scenarios could be more complex, and the chosen method might not adequately model all relevant factors.

In addition to evaluating spellcheckers, future research might examine the usefulness of analytical tools in correcting grammatical errors. Assessing their effectiveness in correcting grammar mistakes and enhancing the overall quality of text input will provide a more comprehensive understanding of the tools' capabilities and is reflected in tool robustness and the ability to correct such errors can impact biometric systems based on typing errors, e.g. a user may not use correct conjugation in Present Simple sentences by not adding verb's ending.

Another avenue for future work involves implementing and testing the most robust spellchecker identified in this study within a behavioural biometrics system based on typing errors. By incorporating the spellchecker into the system, its real-world performance and its impact on the accuracy and reliability of the biometric authentication process can be evaluated. To further validate the selected spellchecker's effectiveness and integration into the behavioural biometrics system, it is essential to conduct experiments on a representative group of users. This will help determine the system's performance across diverse user profiles, including variations in typing styles and language proficiency.

By addressing these future work directions, researchers can continue to refine and optimize the performance of the behavioural biometrics system that relies on typing errors, ultimately contributing to

the development of more secure, accurate, and user-friendly biometrics solutions.

REFERENCES

Barrus, T. (2022). Spellchecker. https://pyspellchecker.readthedocs.io/en/latest/.

Bexte, M., Laarmann-Quante, R., Horbach, A., and Zesch, T. (2022). LeSpell - a multi-lingual benchmark corpus of spelling errors to develop spellchecking methods for learner language. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 697–706. European Language Resources Association.

Blurock, E. (2021a). String Similarity Metrics – Token Metohods. https://www.baeldung.com/cs/string-similarity-token-methods.

Blurock, E. (2021b). String Similarity Metrics –Sequence Based. https://www.baeldung.com/cs/string-similarity-sequence-based.

Cohen, W., Ravikumar, P., and Fienberg, S. (2003). A comparison of string metrics for matching names and records. In *Kdd workshop on data cleaning and object consolidation*, volume 3, pages 73–78.

Damerau, F. J. (1964). A technique for computer detection and correction of spelling errors.

Dice, L. R. (1945). Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302.

Gawarecki, M. (2017). hp-lovecraft. https://data.world/mattgawarecki/hp-lovecraft.

Gupta, S., Maple, C., Crispo, B., Raja, K., Yautsiukhin, A., and Martinelli, F. (2023). A survey of human-computer interaction (hci) & natural habits-based behavioural biometric modalities for user recognition schemes. *Pattern Recognition*, 139:109453.

Haque, S., Eberhart, Z., Bansal, A., and McMillan, C. (2022). Semantic similarity metrics for evaluating source code summarization. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, pages 36–47.

Jaro, M. A. (1989). Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420.

Jonas McCallum, F. S. (2021). autocorrect. https://pypi.org/project/autocorrect/.

Kleinschmdit, T. (2021). Gingerit. https://pypi.org/project/gingerit/.

Levenshtein, V. I. et al. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10.

Loria, S. (2018). Textblob. https://pypi.org/project/textblob/0.15.1/.

Moore, G. B. (1977). *Accessing individual records from personal data files using non-unique identifiers*, volume 13. US Department of Commerce, National Bureau of Standards.

Morris, J. (2022). language-tool-python. https://pypi.org/project/language-tool-python/.

Mróz-Gorgoń, B., Wodo, W., Andrych, A., Caban-Piaskowska, K., and Kozyra, C. (2022). Biometrics innovation and payment sector perception. *Sustainability*, 14(15).

Nate Smith, R. F. (2013). lovecraftcorpus. https://github.com/vilmibm/lovecraftcorpus.

Näther, M. (2020). An in-depth comparison of 14 spelling correction tools on a common benchmark. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 1849–1857, Marseille, France. European Language Resources Association.

Norvig, P. (2007-2016). How to write a spelling corrector. https://norvig.com/spell-correct.html.

Ozinov, F. (2020). jamspell. https://pypi.org/project/jamspell/.

Parappuram, A., Nidhina, T. R., and Gopal, G. N. (2016). Continuous user identity verification using typing error classifications. In *2016 International Conference on Computing, Communication and Automation (ICCCA)*, pages 1194–1198.

PrithivirajDamodaran (2021). Gramformer. https://github.com/PrithivirajDamodaran/Gramformer.

Singhal, A. et al. (2001). Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4).

Sorensen, T. A. (1948). A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on danish commons. *Biol. Skar.*, 5:1–34.

Vijaymeena, M. and Kavitha, K. (2016). A survey on similarity measures in text mining. *Machine Learning and Applications: An International Journal*, 3(2).

Wu, G. (2021). String Similarity Metrics – Edit Distance. https://www.baeldung.com/cs/string-similarity-edit-distance.