Platform for Studying Self-Repairing Auto-Corrections in Mobile Text Entry based on Brain Activity, Gaze, and Context

Felix Putze[1] **Tilman Ihrig**[1] **Tanja Schultz**[1] **Wolfgang Stuerzlinger**[2]
[1]Cognitive Systems Lab, University of Bremen, Bremen, Germany
[2]SIAT, Simon Fraser University, Vancouver, Canada
{felix.putze, ihrigtil, tanja.schultz}@uni-bremen.de w.s@sfu.ca

ABSTRACT
Auto-correction is a standard feature of mobile text entry. While the performance of state-of-the-art auto-correct methods is usually relatively high, any errors that occur are cumbersome to repair, interrupt the flow of text entry, and challenge the user's agency over the process. In this paper, we describe a system that aims to automatically identify and repair auto-correction errors. This system comprises a multi-modal classifier for detecting auto-correction errors from brain activity, eye gaze, and context information, as well as a strategy to repair such errors by replacing the erroneous correction or suggesting alternatives. We integrated both parts in a generic Android component and thus present a research platform for studying self-repairing end-to-end systems. To demonstrate its feasibility, we performed a user study to evaluate the classification performance and usability of our approach.

Author Keywords
Text entry; auto-correction; self-repair; eye gaze; EEG

CCS Concepts
•**Human-centered computing** → **Human computer interaction (HCI);**

INTRODUCTION
To increase the usability of systems, some modern user interfaces include components that record sensor data and/or user input and interpret it through statistical models or other machine learning techniques. One prominent example is the auto-correction module embedded in every mobile software keyboard. On most current mobile devices, auto-correction uses a statistical language model [18] to replace the input text sequence with the most likely sequence of letters or words. While auto-correct usually leads to improvements, it is, like any statistical model, prone to errors. The occurrence of such system-generated errors can annoy users, as such mistakes violate the user's expectation of text entry system behavior and can lead to embarrassing or costly results. Even the most sophisticated state-of-the-art auto-correction approaches have error rates around 5% [14]. Given that we can expect a substantial fraction of typed words to contain at least one wrong letter[1], we can assume a sufficiently high prevalence of auto-corrections – and therefore, auto-correction errors – during mobile text entry. While it is difficult to get auto-correction error rates from "in the wild" studies on text entry (as these do not capture the actually intended text), we can derive a lower boundary of 8.4% auto-correct error rate from Buschek et al. [8], who report this as the number of auto-corrections which were manually changed (which does not include auto-correction errors which were ignored or missed). Each such auto-correction error is cumbersome to repair (as it requires otherwise unnecessary operations), interrupts the flow of text entry (because "random-access" to the text is not a efficient operation for the user, auto-correction errors may need to be dealt with immediately), and challenges the user's agency over the process (as it results in uncontrollable, potentially undesired changes to the entered text). An approach to detect such errors automatically and (at least) attempts to repair them can thus increase the usability of a text entry system. Besides, a repair attempt can be considered as an implicit apology for a previously-made error and it is known that apologetic systems are perceived as more appealing and usable compared to non-apologetic ones [28].

When a user perceives erroneous and unexpected system behavior, this leads to cognitive and behavioral responses. If the system can detect these responses from sensor data and context information, it can use such knowledge to respond proactively to an error that occurred by attempting a self-repair. For example, brain activity measured through electroencephalography (EEG) can encode error potentials [34] which occur in such situations, eye gaze may respond with prolonged fixations to unexpected system behaviors [21], or the context provided by the language model of the auto-correction could hint at an increased likelihood of an auto-correction error to occur. Note that exploiting such cognitive responses to system errors requires the errors to be actually perceived by the user. We do not consider this an important limitation, as any measures which do not rely on the user noticing the error should have been implemented preemptively anyways.

[1]we estimate this fraction as 23%, based on a 5% probability for mistyping a single letter [3] and an average English word length of 5.

CHI'20, April 25–30, 2020, Honolulu, HI, USA

ISBN 978-1-4503-6708-0/20/04…$15.00
DOI: http://dx.doi.org/10.1145/3313831.3376815

Figure 1. Autocorrect failures can be embarrassing or costly (source: http://damnyouautocorrect.com).

In this paper, we build on previous work which demonstrated that it is possible to detect user reactions to incorrect auto-corrects from the EEG signal and context cues [31] well after the fact, i.e., in post-hoc analysis. However, studying the potential for automatic recognition alone is not sufficient for a self-repairing system. For this purpose, we also need to create a research platform that allows us to study an end-to-end interactive system, including a real-time recognition method, appropriate strategies to decide between different self-repair options, and a user interface to ultimately offer appropriate self-repair operations. This paper presents such a research platform and demonstrates its feasibility in a user study. In our study, we investigate an extended, real-time capable version of the approach by Putze et al. [31] to which we added eye-tracking as an information channel to identify if and when the user actually perceived an auto-correction. We added gaze information, because the original approach [31] simply assumed users to immediately perceive any auto-correction event, instead of accounting for trial-by-trial variation of the time it takes to attend the event. Such variation could for example be caused by different gaze targets (on keyboard vs. on the text panel) when the event occurs.

For creating an end-to-end interface for resolving auto-correction errors, we first replaced the previously employed simple edit-distance-based auto-correction method with a more realistic statistical language model, as implemented in most commercial text entry systems. Second, we designed an auto-correction strategy which chooses between different system actions to resolve a detected auto-correction error, such as autonomously swapping the initial correction with a second-best estimate or offering different, equally valid, replacement options to the user. This strategy is based on classification confidence and language model output. Third, we implemented this self-repair component as a real-time Android text entry module that can be integrated into arbitrary applications.

While related work studied physiological responses to erroneous behavior (e.g., [29, 39]), we present here to the best of our knowledge in an ecologically valid setting the first end-to-end system which exploits a real time recognition of such responses. Our main research objective is to establish that it is possible to impact interaction in a measurable way by adding self-repairing functionality, despite challenging conditions. To tackle this objective required us to extend the proof-of-concept detection approach proposed by Putze et al. [31] into a fully-fledged research platform for studying multimodal error responses and appropriate system responses to them. For this, we provide all necessary components for the end-to-end architecture, including the classification server and Android keyboard, as open source software at https://osf.io/wkxct. This will enable further research into self-correcting interfaces in text entry and other use cases in the future.

RELATED WORK

Text entry auto-correction uses a number of different methods to identify errors in the entered text and to propose likely corrections. All such approaches include information about the entered text, typically in the form of statistical language models [18]. Modern approaches often use such models to support both word completion and typing correction [7]. Other information sources for auto-correction have been explored as well: hand postures [16], key press timing [11], accelerometer data in mobile text entry [15], uncertainties touch positions [41], geometric pattern matching [24], or as a combination of spatial and language models [14]. Multi-modal input and output can improve auto-correction performance or usability, e.g., through auditory and tactile feedback during eyes-free input [38] or through voice and text-based repair [37].

Very little is known about text entry errors and how they are corrected in the wild, automatically or manually. Komninos et al. [23] performed a study on how users deal with typing errors and identified different strategies for interleaving reviewing and correcting. Buschek et al. [8] collected in-the-wild text entry data on mobile devices and reported statistics on auto-correction usage. They found that auto-correction is not popular among the participants as “it was often wrong”. Although their work identified the prevalence and impact of auto-correction errors, there are only few approaches which provide mechanisms to cope with them. Arif et al. [2] introduced the “Smart-Restorable Backspace”, which allows users to deal with auto-correction errors with very few operations. This technique was highly accepted by users and lead to quick manual resolutions of auto-correction errors. Quinn et al. [32] analyzed different strategies for presenting auto-correction suggestions. They showed that while the number of keyboard interactions could be reduced by confidence-based selection of auto-corrections, the baseline cost of attending to auto-corrections and deciding on their resolution was still present. Alharbi et al.’s work on WiseType [1] investigated better presentation methods for auto-corrections and suggestions through appropriate highlighting. They also supported fast manual handling of auto-correction errors through swiping.

To resolve auto-correct errors automatically, it is important to detect them first. A lot of work has pursued the classification of neural responses to system malfunctions, for example, increased workload and neural responses, as investigated by

Hirshfield et al. [19]. To detect specific instances of erroneous system behavior, several researchers have used EEG to identify error potentials from brain activity. For example, Förster et al. [13] used classification of error potentials during operation of a gesture recognition system to improve its performance through online adaptation of the gesture recognizer to incorrectly classified trials. Vi and Subramanian [39] proposed a recognizer for error potentials based on a consumer-level EEG device. They performed person-dependent classification of error potentials, using a test set of 80 Flanker trials and achieved a classification accuracy of about 0.7. Using a simulation of error potential classification with different error rates, they also showed that already a non-perfect detection rate between 0.65 and 0.8 enhanced spatial selection through better detection of user errors in flick-gestures on a touch surface. The authors analyzed accuracy improvements by allowing manual corrections when an error potential was detected, but did not analyze costs or other usability aspects. Putze et al. [30, 29] showed that error potentials can be detected in gesture-based user interfaces and used them to provide a self-correcting input mechanism. The authors used a model-based simulation to investigate the effect of different correction strategies and validated its predictions in a user study. In further research [31], the authors also showed that error potentials can be used together with context features to detect auto-correct errors in tablet-based text entry after the fact.

The above body of research also discusses a few strategies for handling errors in processing user input, such as replacing erroneous input interpretations with the second-best alternative [26] or asking the user for re-entering some input [35].

Yet, EEG data is not the only source for information about perceived system errors. Kalaganis et al. [22] showed that the combination of EEG and gaze-based data resulted in a more robust detection of errors than the individual modalities. Banovic et al. [6] demonstrated that, to avoid the costs of correcting mistakes, user typing behaviors change in anticipation of text entry errors. This implies that user behaviors during typing and error handling can also act as a feature source to detect auto-correction errors.

PLATFORM FOR SELF-REPAIRING AUTO-CORRECTION

Our new system for self-repairing auto-correction consists of several components which work together to support mobile text entry: The *text entry component* is an Android-compatible keyboard application and acts as a user interface to present auto-corrections and their repair based on a statistical language model. The *classifier* activates when an auto-correction was performed and processes incoming EEG, gaze, and context information to detect whether the auto-correction was (likely) successful or erroneous. The *repair strategy* resolves detected auto-correction errors by providing alternatives. All components are embedded in an *online system*. Figure 2 illustrates the interplay of these components.

Text Entry Component

We used a custom keyboard application to enable users to enter text. This component uses a standard German mobile keyboard layout where all keys necessary for the experiment are available in the standard layout with a single button press. We had to create our own component, as standard Android auto-correction methods do not provide all data necessary for our work, see below. Such data also cannot be sent "through" the standard Android interface for spelling correction services.

To get access to all likelihood scores associated with the provided corrections, we implemented a custom spell-check service, based on state-of-art auto-correction methods: we employ an n-gram based language model to generate corrections with the highest likelihood among all candidate words with an edit distance smaller than a given threshold. Because the candidate search had to be done in real time on a tablet, the edit distance threshold was set to 4, meaning that words with an edit distance of larger than 4 to any word in the vocabulary of the used language model were not corrected. Search of all candidate words with the given edit distance was done via SymSpell [2] using a vocabulary of the 10,000 most common German words. Each candidate was then given a likelihood score by a custom trigram language model, trained on German phrases from the Leipzig Corpora Collection [17], using only phrases containing the 10,000 most common German words. The model was trained with the SRILM toolkit [36] using Kneser-Ney smoothing. The auto-correction then always selected the candidate with the highest likelihood score.

An important aspect of analyzing user responses to erroneous system behavior is to ensure that the behavior in question is actually perceived by the user. To maximize the probability of users perceiving an auto-correction, our keyboard employed four mechanisms: First, after the participant started typing, the target phrase to type was removed to prevent participants from gazing at the text presentation (and thus not be able to see potential corrections). Second, the auto-corrected word was replaced within the text field. Third, a sound was played for each correction. Fourth, and motivated by previous work [31], the replacement was shown as a notification on the keyboard over the letter which was pressed last, as well as the space bar (see Figure 3).

This last mechanism was introduced to make sure users noticed auto-corrections even when they focused their gaze on the keyboard instead of the typed text. The notification was displayed at multiple locations to maximize the probability that it appeared at a location at which the user was already looking at. In contrast to solutions with a notification in a fixed location, e.g., in the center of the screen, our design allows users to naturally control their eye gaze.

Classification Setup

To detect auto-correction errors during text entry, we formulate a two-class classification problem. We extract windows of EEG and eye tracking data aligned with the presentation of an auto-correction in the user interface. The alignment is performed by detecting eye gaze fixations, which indicate that a presented auto-correction is actually perceived. The `noError` class is assigned to windows with corrections which yielded the expected word, while the `error` class is assigned to windows corresponding to wrong corrections.

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

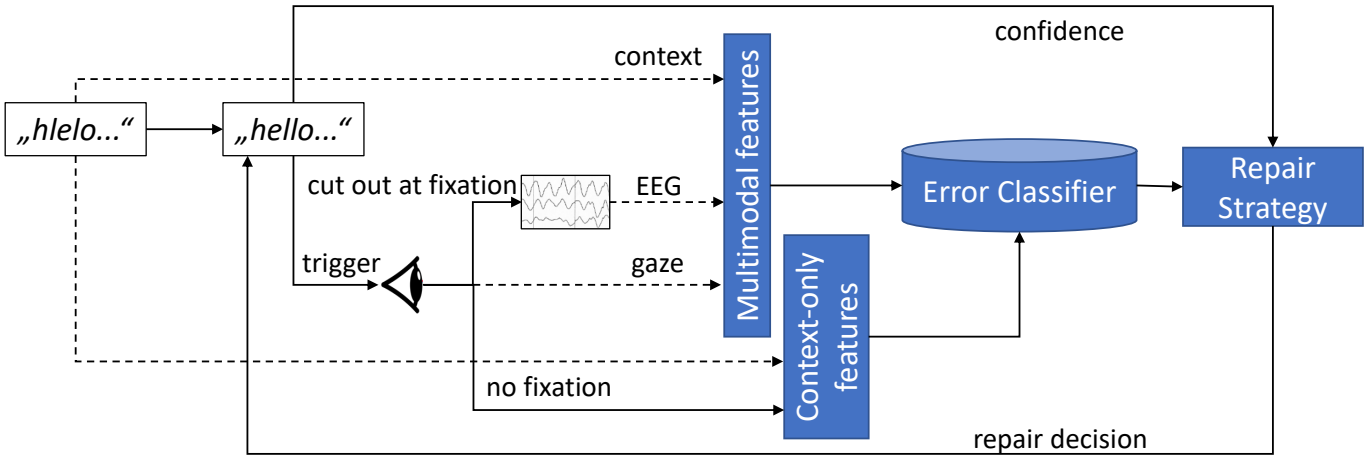

Figure 2. Flowchart of the auto-correct self-repair process.

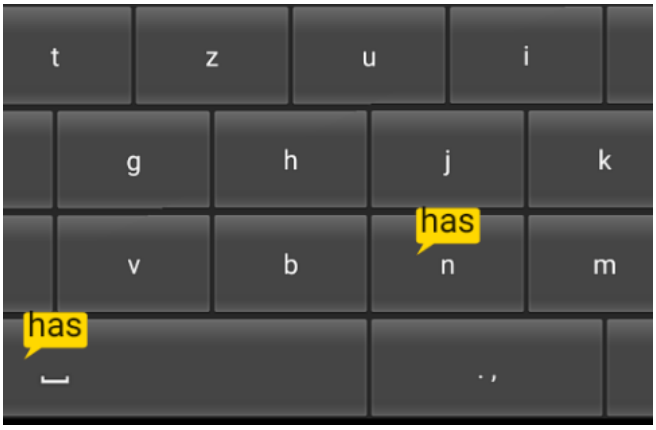

Figure 3. Screenshot of the Android keyboard showing auto-corrections as used by Putze et al. [31] and this study. To increase the likelihood users see the auto-corrections, we display them on the last pressed key and space bar.

We trained the classifier in a person-dependent manner, i.e., an individual model was trained for each user. For this purpose, the classifier uses features from three different information sources: EEG, gaze, and context. The EEG features are designed to capture error potentials which occur when an auto-correction error is perceived. Gaze features capture similar responses in the visual processing of such errors, for example through longer fixation times in the presence of an auto-correction error. Finally, the context features encode information about the status of text entry itself at the time the correction occurred. Person-dependent training is necessary as person-independent models of error potentials are still an open research question [40].

Beyond acting as a source for features, gaze data has another important role in the classification process: There is substantial variance in the latency between the occurrence of an auto-correction in the interface and the point in time where it is actually perceived. Putze et al. [31] simply extracted the classification window right after the presentation of the auto-correction. Here, we improve on this method and exploit the fact that a physiological marker for the perception of the correction is a fixation at one of the locations where the correction is shown. Therefore, our approach tries to identify the fixation following the auto-correction event as the onset of the classification window. For this, a target fixation needs to be at least 200 ms long, have a distance of at most $0.2x$ to the closest correction notification (where both the width and height of the input device are defined as x), and start before typing is resumed. If multiple fixations fulfill these criteria, the first one is selected. We then extract the target window with a duration of 1 s starting from the fixation onset. See Figure 4 for a visualization of fixations in response to an auto-correction.

The identified classification window is relevant to extract the EEG and gaze features. However, if no window can be identified that fulfills the criteria listed above, no such features can be extracted. Then, the classifier resorts to a fallback mechanism which only relies on the context data; context features are defined independently of the auto-correction event itself and thus do not require a precisely extracted classification window.

For the actual classification, we first extract a joint feature vector from EEG, gaze, and context data. In the following, we describe how these features are derived:

EEG Features: The feature extraction from EEG is similar to previous work [31]: We first adjust the data to have zero mean. Then, we segment the classification window into smaller parts of 50 ms length. Subsequently, we bandpass-filter the EEG data from the electrodes at the positions Fz and FCz between 1 Hz and 10 Hz using a fifth-order Butterworth filter. For each segment, we calculate the signal mean and stack these as a feature vector. Additionally, we calculate the power spectrum using the Welch method for each window.

Gaze Features: From the eye tracking data we derive a number of features. Using the fixation detection and saccade detection algorithms from PyGazeAnalyser [12], we extract fix-

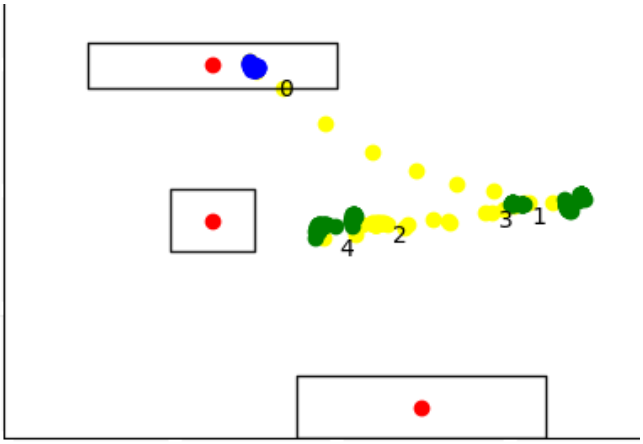

Figure 4. Exemplary visualization of gaze data for 2 seconds after a correct correction. The red dots with their respective boxes represent the points where corrections can be visually noticed, namely (from top to bottom): The text entry field, the last pressed button before the correction, and the space bar. Green dots are gaze points within a fixation. Blue dots are gaze points within a 'relevant' fixation. The numbers represent the chronological order of each fixation. Figure is cropped to the relevant segment of the tablet surface.

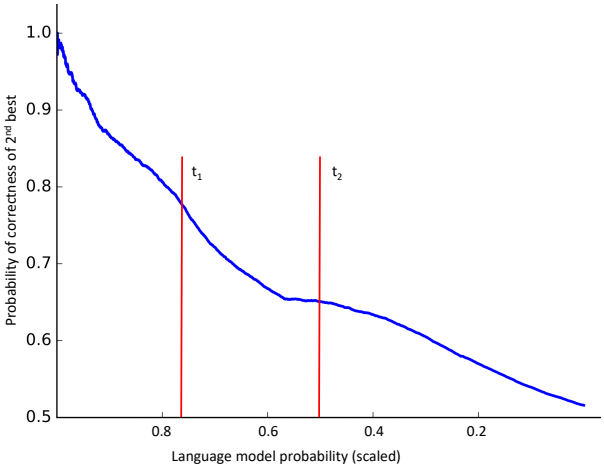

Figure 5. Cumulative accuracy of 2nd-best candidate (for $n = 6$), ordered by descending confidence (scaled language model probability) of the 2nd-best candidates. Note that the vertical axis starts at 0.5, i.e., random chance.

ations and saccades. For both event types, we calculate the number, relative total duration, and mean length. Additionally, we calculate mean confidence (estimated by the Pupil Labs recorder) for all samples in the window and the total gaze travel distance.

Context Features: As support for the above-mentioned psychophysiological features, we derive a number or context features, which encode information about user behavior and the state of the language model and thus *indirectly* also the likelihood of an auto-correction error. Our context features are: Typing speed during the auto-corrected word, length of the auto-corrected word, time until typing continues, the number of suggestions with minimal edit distance, the absolute language model score of the replacing word, and the relative language model score (relative to the scores of the other candidate words). An additional binary feature, which indicates whether a relevant fixation was identified, is only used as feature for the fallback context classifier. The context features capture the likelihood of a wrong correction and the likelihood that a wrong correction is actually noticed.

To generate the final feature vector, all features are concatenated and normalized by removing the median and scaling according to the interquartile range. For classification, we employ a boosted regression tree (using XGBoost [10]) with 1000 base classifiers and a learning rate set to 0.01. For the context classifier, which is trained on a much smaller feature set, we employ Linear Discriminant Analysis with an analytically derived shrinkage coefficient [25].

Resolution Strategies

In this section, we present our new strategy for self-repair of auto-corrections, which is more robust to mistakes of the classifier and the ambiguity of the underlying language model.

The central idea of this strategy is to use contextual information from the auto-correction, e.g., confidence of the 1st-best and 2nd-best candidate, to decide on how to much to trust an `error` result of the classifier, where the confidence of the nth candidate is its n-gram likelihood normalized by the likelihood mass of all candidates. Depending on this contextual information, our strategy can either

- automatically replace the initial correction with another candidate,
- show a pop-up menu to let the user choose an alternative, or
- do not react and rely on the user to correct the potential error.

To explain our new repair strategy, we formalize the auto-correction process as follows: To replace the last word v_3 in a sequence $v_1v_2v_3$, we generate an n-best list of candidates $w_1, \ldots, w_N$ by retrieving all dictionary entries with minimal edit distance and ranking them according to their n-gram probability $p(v_1v_2w_i)$ from highest to lowest. We then use the best candidate w_1 to replace the current word.

Using this notation, we formalize our repair strategy, which combines two mechanisms: Automatic replacement of the best candidate w_1 with the second-best w_2 or presenting a transient pop-up menu near the text entry field with alternatives for the user to choose from. Yet, even if we actively replace with the second-best candidate, we still also show the transient pop-up menu with additional alternatives to support quick manual repair, as necessary. If the user continues typing, the transient pop-up disappears automatically without further changes to the input text. However, to avoid unnecessary distraction or backward steps, the two mechanisms are not employed for *every* error.

As the transient pop-up menu does not automatically modify the actual text without user intervention, it is less intrusive

Algorithm 1 Auto-correct Self-Repair

```
Calculate confidence c_1st for w_1.
c_1st = ngram(w_1) − ngram(w_2)
normalize c_1st by ∑_{w∈C} ngram(w)
if c_1st > t_1st then
    return no repair
else
    l = classification result
    if l == noError then
        return no repair
    else
        c_2nd = ngram(w_2) − ngram(w_3)
        normalize c_2nd by ∑_{w∈C\w_1} ngram(w)
        if c_2nd > t_2nd then
            replace w_1 by w_2                ▷ Direct Replacement
            open pop-up(w_1, w_3, w_4, w_5)   ▷ Repl. Pop-up
            return
        else
            open pop-up (w_1, w_2, w_3, w_4)  ▷ Full Pop-up
            return
        end if
    end if
end if
```

than a direct word replacement and can thus be employed more freely. Still, the transient popup menu creates a visual distraction which could be avoided. Thus, we use two thresholds on the confidence scores of the initial correction and the replacements to determine if the current instance is a good opportunity for automatic self-repair. After all, a central goal of our repair strategy is to avoid the detrimental effects of false alarms from the auto-correction error classifier. The pseudo-code shown in Algorithm 1 illustrates the logic for our repair strategy, which is employed after an auto-correction error was detected.

To prevent the repair strategy from replacing auto-corrections too aggressively, we use two thresholds, t_{1st} and t_{2nd} on the confidence. Based on pilots, we heuristically selected the values for t_{1st} and t_{2nd} to optimize the trade-off between mitigation of false alarms and leveraging the successful detection of auto-correction errors. As an example for words of length six, Figure 5 shows how with decreasing confidence of the 2nd-best candidate its cumulative likelihood of being correct decreases from near-certainty to a coin toss. This shows that our chosen confidence is a valid parameter to base the repair decision on. We set the thresholds t_{1st} and t_{2nd} to 0.75 and 0.5, respectively. These values were chosen based on pilot tests and a simulation of repair costs for different system configurations.

It should be noted that a user does not have to wait to continue typing until the self-repair has been triggered. The self-repair will occur in whatever state the text entry system is when the result is returned and the user can then respond to it, if necessary.

Online System

We implemented an end-to-end system which is able to perform self-repair in real-time. This setup consists of a Python server which collects the data and performs the classification. It is connected to its data sources and the text entry application via the LabStreamingLayer[3] (LSL) middleware, which enables us to switch between different types of EEG devices, for example the BrainVision actiCHamp and the wireless OpenBCI[4]. Text entry is handled by a custom Android app which employs the self-repairing auto-correct for text entry. Both Android apps can also connect to the classification server via LSL. When the server is notified of an auto-correction, it retrieves the relevant EEG and context data from its buffer and performs classification. Classification models are trained before each session from data recorded during a training segment with predefined sentences.

Upon detection of an auto-correction error by the classification server, the system then informs the text entry app about it. The app then follows the repair strategy described in Algorithm 1 to provide pro-active self-repair. This setup is very generic and can be employed on any Android device. See Figure 6 for a screenshot of the application.

Figure 6. Screenshot of the user interface for the text entry with automatic self-repair, showing the user making a text entry error (a), an auto-correction error (b), an automatic self-repair with a pop-up (c) and the result after a finished self-repair (d). Example in English for illustration purposes.

DATA COLLECTION

Data was collected from sixteen participants. Participants were students and employees at the University of Bremen with ages between 21 and 55 years (mean 27.6, standard deviation 11.0), four of whom were female. All participants gave their

[3] https://github.com/sccn/labstreaminglayer
[4] https://openbci.com/

Figure 7. User typing on Android phone using self-repairing auto-correct in the experimental setup with EEG cap and eye tracker.

written consent to the data collection. The study was approved by the local ethics committee.

The data of the first ten participants was used to evaluate the classification component of the system. Participants entered a total of 220 sentences. One sentence at a time was presented at the top of the application.

Presented sentences for training and the main study were selected from the German Open Subtitles corpus [33]. The sentences were filtered and normalized so that all presented sentences were restricted to contain only English lower case letters and only words from the dictionary of the language model. This was done to avoid confounds due to switches between different keyboards or modes and to avoid any associated errors, as well as to avoid un-repairable errors. As the presented sentence disappeared after the first character, participants were asked to remember the sentence and then type it as they would normally do.

To keep participants from typing too slowly and deliberately, an alarm was triggered if a sentence took longer than 20 seconds to type. During all text entry submissions from a participant, we did not want to enforce correctness or completeness of the phrase, to avoid breaking the typing flow of the participants and to provide an "easy out" if a participant forgot the rest of a sentence. To avoid later ambiguities in the alignment of the entered phrases and the originally prompted text during the training of the classifier, we only used segments of data up to the first mismatch between entered and expected text and disregarded the rest.

The experiment was conducted on a Google Pixel C tablet (Android 8.1), which was positioned slightly below eye height. The high placement of the tablet was chosen to encourage users to keep their eyes reasonably open (in contrast to half-closed eyes when looking down to a device in their lap), which improved eye tracking accuracy substantially. Markers were placed around the tablet to enable the outward looking camera of the eye tracking system to register the gaze direction accurately, even if the participants moved their heads. Sentences were presented using a modified version of the TEMA Software [9]. Participants typed 10 sentences in a row, progressing to the next sentence via the Enter key, and were free to start the next set of 10 sentences at their leisure. At the start of every set, participants had to look at the middle of the tablet and needed to have the tablet centered in their field of view for the first sentence to appear. This was done to ensure the best possible calibration of the eye tracking component throughout the experiment.

To record enough training data we needed to induce a sufficient number of auto-corrections. Thus we set the keyboard to replace 5% of typed letters with neighboring keys [4], except for the space bar. An error rate of 5% is similar to a normal typing error rate in mobile text entry [3]. While this measure may create a slight confusion when noticed, it enabled us to keep the time required for the experiment within reasonable bounds. As auto-corrections take place only after the completion of a word (i.e., during times when no letter replacements were performed), potential responses to detected keyboard errors could not be confused with responses to auto-correction errors.

The data of the final six participants was used to evaluate the complete end-to-end interface for self-repairing auto-correction. They first entered 150 sentences to enable us to train the classifier to perform optimally for each user. During this phase, participants were asked to manually correct mistakes of the auto-correction. We also calculated statistics on manual repair of auto-correction errors from this data. After training the classifier, participants then entered 50 additional sentences, this time with activated self-repair. After the experiment was completed, participants filled a questionnaire regarding the usability of the system. The fixed ordering of the two conditions (with and without self-repair) was necessary to keep the total duration of the experiment under control. As the text entry interface closely followed the appearance and behavior of well-known standard components (i.e., the default Android keyboard), we can assume that participants were familiar with it and did not exhibit a strong learning effect.

For data acquisition, participants were equipped with a BrainProducts actiCAP with 32 active EEG electrodes. The electrodes were organized in a standard 32 channel actiCAP layout following the international 10-20 system. P2 was used as reference electrode. Impedance was kept below $16k\Omega$. The measurements were amplified by the actiCHamp amplifier and recorded via the PyCorder software. To synchronize the correction events with the EEG signal, the custom keyboard included a color-switching box to which a light sensor of the recording setup was attached. Note that while 32 electrodes were recorded during the experiments, only two electrodes (plus the reference electrode) were actually used for analysis. Thus, the EEG setup could be (much) simplified in actual application.

Participants received financial compensation in the form of the equivalent of 20 USD. All participants gave their written consent to participate and to have their data evaluated.

EVALUATION

To evaluate our research platform, we first analyzed the classification performance when detecting auto-correction errors. For this purpose, we use the first 150 sentences of a session for training of the model and the remaining 50 sentences for testing. Maintaining the temporal order of the original recording (compared to a cross-validation) allows us to draw more robust conclusions for the online system. Due to the imbalance of classes, i.e., as the auto-correction is giving the right result more often than it is failing, we needed a relatively high number of 150 training sentences.

We performed this specific evaluation by replaying the recorded session to the actual online system from the recorded LSL streams. This allows for an exact reproduction of the online conditions. For each sentence, we evaluated all occurring auto-correction events until the entered text (after auto-correction, automatic repair, and manual repair was applied) diverged from the prompted sentence.

As performance metric, we report the unweighted averaged F1 score. Table 1 summarizes the results. These results indicate that the performance is significantly higher than the random baseline ($p < 10^{-5}$ for a one-sided paired t-test), showing that our model is able to successfully detect auto-correction errors. Furthermore, we see that the addition of the context fallback mechanism (slightly) improves the classification performance and that using the fixation-related windows (instead of fixed correction-related ones) increases classification performance by 7% relative, which is a significant difference ($p = 0.02$ for a one-sided paired t-test), which makes this a substantial improvement over previous work. See Table 1, where "All features, no fixation windows" corresponds to the data from Putze et al. [31] which used no eye-tracking (instead, classification windows were always extracted immediately following the auto-correction event).

Mode	F_1	$F_{0.5}$	F_2	SD
All features	0.65	0.68	0.62	0.1
All features + context fallback	**0.67**	0.71	0.64	0.06
All features, no fixation windows [31]	0.61	0.65	0.57	0.1
Baseline	0.41	0.39	0.47	0.02

Table 1. Classification performance for different feature combinations (SD = standard deviation). We report F_β scores for $\beta \in \{1, 0.5, 2\}$. The baseline is calculated for only predicting the majority class.

For evaluation of the auto-correction strategy, we compared the average required correction time for manual and automatic repair of auto-correction errors. Figure 8 shows a significant difference in correction time ($p = 0.02$ for one-sided paired t-test). In case a re-correction of the automatic result is necessary, the repair time is still not significantly higher than a manual repair (actually still slightly lower), i.e., there is no extra overhead when our method does not succeed. Looking at the operation count metric in Figure 9, we can see that the automatic repair also significantly reduces the number of necessary manual operations ($p = 0.02$ for one-sided paired t-test, for pop-up selection and to account for situations in which no

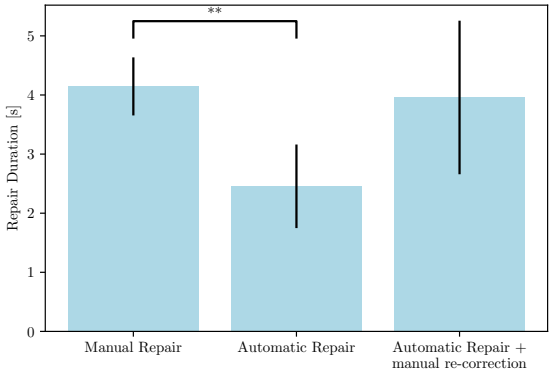

Figure 8. Average duration to repair an auto-correction error manually or automatically.

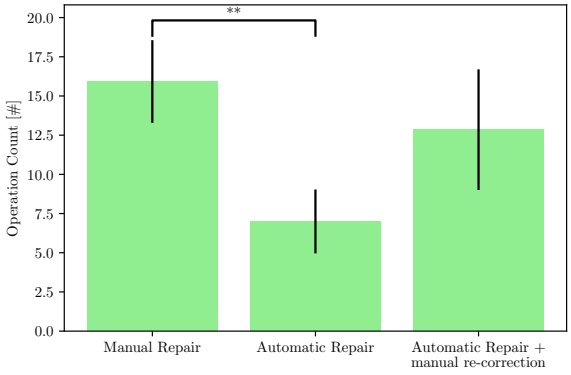

Figure 9. Average number of manual operations to repair an auto-correction error manually or automatically.

self-repair was issued due to high confidence of the original correction). The number of manual operations for the automatic correction is likely overestimated as it also counts key presses which occurred before the self-repair was issued. If we look at situations where manual re-correction was necessary, the number of operations for our new system was still less, but not significantly so.

Table 2 illustrates the performance of the different components of the auto-correction and the self-repair component. Here we compare a manual repair (by user only, through manual keyboard operations), automatic repair (through the self-correcting interface after a detected error; only correctly detected and repaired auto-correction errors), and automatic repair+manual re-correction (through the self-correcting interface after a detected error and with a potential follow-up of manual re-correction, if the repair subsystem did not work correctly). The table shows that the auto-correction exhibits an accuracy of 73.5%, indicating that while its performance is below what one might expect from a "perfect" auto-correction

mechanism, it performs reasonably well. It should be considered that the user prompts were chosen from a different corpus than the data for the language model, which makes auto-correction and its repair harder compared to a more homogeneous, but less realistic matching condition.

The result also implies that the classification task to detect such errors is imbalanced. In 40% of all cases, a direct replacement lead to an automatic repair of an erroneous auto-correction (requiring no manual intervention by the user), and in 77% of the remaining cases, an additional pop-up contained the correct replacement (requiring only a single manual operation). A similar performance was achieved when a pop-up was generated immediately.

Manual re-correction after a failed repair attempt occurred in 26.7% of all cases (absolute number of occurrences: 53 (recorrection) vs. 145 (no re-correction) on average across all sessions).

Method	Correctness [%]	
	Before	After*
Initial correction (1st-best cand.)	73	-
Direct replacement	-	40
Replacement pop-up	-	77
Direct replacement & Replacement pop-up	-	86
Full pop-up	-	75

Table 2. Correctness of several individual correction methods, both *before* the first auto-correct is executed and, as a result to different repair attempts, *after* a failed auto-correct (*relative to the total number of failed auto-corrects). A "pop-up after a direct replacement" does not contain the word that directly replaces the text.

Beyond the objective evaluation, we also looked at the subjective assessment by the participants. Table 3 reports on the reception of the text entry system. While users acknowledged that they frequently made mistakes, they reported only medium agreement to the statements that the keyboard was malfunctioning and that it was sometimes changing letters. Additionally, memorizing the given phrases was no challenge for most users. These two results indicate that our experimental manipulation did not impede text entry and users could still type reasonably swiftly. Participants rated auto-corrections as plausible, which shows that our implemented text entry component behaves comparable to existing commercial systems. The fact that most people did not look at the text entry field supports our decision to present the auto-corrections at multiple places. Participants indicated strongly that they paid attention to the performed corrections.

Table 4 shows the comparison of average user responses for manual and automatic repair of auto-correction errors. Results show that the self-repair approach is perceived as significantly more "intelligent" by the users ($p = 0.01$), i.e., they acknowledge that the system is (sufficiently) context-dependent. We also see a tendency to perceive the self-repair more quickly, and that compared to manual correction it makes the process of dealing with auto-correct errors less annoying. On the other hand, the automatic repair is also perceived as less reliable (as

Statement	Agreement
I frequently made mistakes while I was typing.	5.25 (1.48)
I could type swiftly on the keyboard.	5.13 (1.32)
I thought the keyboard was malfunctioning.	3.44 (1.97)
I could usually remember the sentences well.	5.84 (1.14)
When an auto-correction was performed, I visually checked whether the new word was correct.	6.06 (0.83)
I could anticipate when an auto-correction would happen.	4.94 (0.97)
I mostly looked at the keyboard while typing.	6.50 (0.71)
I ignored the auto-corrections.	2.19 (1.13)
I noticed that sometimes wrong letters would appear.	3.94 (1.89)
The auto-corrections were usually plausible.	5.81 (0.73)
I could anticipate when an auto-correction would be correct.	4.31 (1.53)
I mostly looked at the text field while typing.	2.31 (1.21)

Table 3. Typing behaviours questionnaire, using a 7-point Likert scale (from strong disagreement=1 to strong agreement=7). Standard deviation in brackets. Cells in light gray support the validity of the evaluation. Cells in dark gray are potential challenges.

it can introduce a new source of errors) and more confusing. While not all these effects are significant, we see a strong correlation (absolute average Pearson's ρ of 0.61) between classification performance and positive assessment of the self-repairing variant. This shows us that a well-performing self-repair can improve user satisfaction during text entry noticeably.

DISCUSSION & CONCLUSION

In this paper, we presented a platform for research on recognition and reaction to multimodal error responses in the context of self-repairing auto-corrections and evaluated it in a user study. This platform brings together a 1) multimodal, real-time classifier for the detection of error responses during text entry, 2) a multi-state self-repairing strategy which takes confidence and context into account, 3) an integration of both components into generic Android components, which allows the investigation in arbitrary applications.

Our study demonstrates that the system successfully detects a large percentage of incorrect auto-corrections and either automatically fixes them or enables users to fix them through a pop-up menu, with a manual correction fallback. We further showed that a pro-active self-repair strategy is able to significantly reduce the required time for repairing erroneous auto-corrections as well as the number of required user operations. Users also perceived the behavior of the self-repairing auto-correct to be significantly more "intelligent" than the auto-correction without self-repair.

Statement	Manual	Automatic
Auto-correct mistakes could be repaired reliably.	5.86 (0.99)	5.25 (1.15)
Auto-correct mistakes could be repaired quickly.	5.00 (1.41)	5.31 (1.26)
I felt comfortable using the system.	5.69 (1.21)	5.31 (0.77)
I regarded the system's behaviour as intelligent.	4.31 (1.45)	5.5 (0.87)
I found the system confusing.	1.94 (0.83)	2.86 (1.05)
I had no control over the system's behaviour.	2.81 (1.63)	3.06 (1.43)
Repairing auto-correct mistakes was annoying.	4.38 (1.22)	3.94 (1.39)

Table 4. Questionnaire results for the auto-correct repair mechanisms, comparing manual and automatic correction. Participants used a 7-point Likert-Scale (from strong disagreement=1 to strong agreement=7). Standard deviation in parentheses.

Our results show that detection of erroneous system behavior in a realistic, complex text entry task is feasible, which goes well beyond most previous work that applies the detection of error potentials to user interfaces. We showed how the combination of multiple modalities, namely EEG, gaze, and context information can contribute to a more robust system. This improvement is not only due to the creation of a larger feature set, but also through the more accurate determination of the classification window together with an appropriate fallback mechanism. The proposed classification method as well as the confidence-based strategy for selecting between different repair options can be transferred to other human-computer interfaces where erroneous system behavior can be expected, such as speech recognition or recommender systems.

Now that we have demonstrated that our research platform can handle one specific auto-correction method, we are planning to extend it in many different directions. For example, we could easily replace the text entry method (e.g., using a swiping method or even spoken text entry), perform a comparison of different modalities for detection of error responses, or evaluate different self-repair strategies. We already investigated an always-on variant of the auto-correction through a simulation and a pilot experiment. While the simulation results showed relatively good performance, pilot users found it confusing. Another possibility would be to study error responses not only to individual words but also to text messages that were sent and then immediately regretted. Another alternative approach to handling detected auto-correction errors would be not maximize automation through self-repair, but instead use the "error self-awareness" of the system for building trust with the user through transparency and feedback, following the suggestions of Hoff and Bashir [20].

To enable others to build on the presented system, the software components of our work are all available at Open Science Framework. The modular architecture enables the modification of individual components to adapt the platform to many different use cases.

Our new classifier requires only two EEG electrodes (plus reference and ground electrodes) and an affordable mobile eye tracker, i.e., there is substantial room for improvement in terms of further miniaturization and increased comfort. Recent years have shown remarkable improvements in mobile eye tracking from tablet-based cameras [5] and consumer-grade EEG systems (such as the NeuroSky or Muse headsets). For future research, such developments might enable us to actually take the research platform into the field, as one limitation of the presented work is that while the technology is mobile-ready (e.g., through use of wireless EEG recording devices), we did not actually evaluate mobile use of the system. After all, it is likely that during mobile text entry users will make more mistakes. We also expect to more frequently see situations where an auto-correction error is not attended to and thus not detected by our current approach. Moving to a smaller screen would also add new challenges, such as for the localization of eye gaze; Müller [27] showed that target differentiation from eye gaze on a smart phone screen is possible, but might come with reduced accuracy compared to larger tablet screens.

Our research platform also enables the future exploration of the potential of self-repairing auto-corrections using only the available context information or other modalities, such as fNIRS, skin conductance response, or facial features.

Another limitation of our work is that while our core auto-correction implementation uses a state-of-the-art statistical language model, its performance might not perfectly match that of the highly optimized auto-correction methods embedded in today's mobile devices and their operating systems. Yet, we cannot access these operating system components, as they do not provide sufficient access to all the information required by our current approach. We plan to explore this topic further through collaborations. This limitation also holds for the manual repair to which we compare to and which could be improved for a more realistic comparison to state-of-the-art text entry. This could for example happen by adding the alternative replacements used in the pop-ups of the automatic repair to every auto-correction to allow more efficient manual repair as well. For the presented study, the simple manual auto-correction allowed us to successfully demonstrate measurable impact through detecting and reacting to users' error responses. Future research with our presented platform will reveal whether the currently implemented repair strategy will be able to improve performance over the best available sophisticated text entry support systems.

ACKNOWLEDGEMENTS

This work was done within the project DINCO "Detection of Interaction Competencies and Obstacles". We thank the German Research Foundation (DFG) for funding this DINCO project under the reference number 316930318 and the Canadian NSERC Discovery program.

REFERENCES

[1] Ohoud Alharbi, Ahmed Sabbir Arif, Wolfgang Stuerzlinger, Mark D Dunlop, and Andreas Komninos. 2019. WiseType: A Tablet Keyboard with Color-Coded

Visualization and Various Editing Options for Error Correction. *Graphics Interface 2019* (2019).

[2] Ahmed Sabbir Arif, Sunjun Kim, Wolfgang Stuerzlinger, Geehyuk Lee, and Ali Mazalek. 2016. Evaluation of a Smart-Restorable Backspace Technique to Facilitate Text Entry Error Correction. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 5151–5162. DOI: http://dx.doi.org/10.1145/2858036.2858407

[3] Ahmed Sabbir Arif and Wolfgang Stuerzlinger. 2009. Analysis of text entry performance metrics. In *Science and Technology for Humanity (TIC-STH), 2009 IEEE Toronto International Conference*. 100–105.

[4] Ahmed Sabbir Arif and Wolfgang Stuerzlinger. 2010. Predicting the Cost of Error Correction in Character-based Text Entry Technologies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 5–14.

[5] Mihai Bâce, Sander Staal, and Andreas Bulling. 2019. Accurate and Robust Eye Contact Detection During Everyday Mobile Device Interactions. *arXiv preprint arXiv:1907.11115* (2019).

[6] Nikola Banovic, Varun Rao, Abinaya Saravanan, Anind K. Dey, and Jennifer Mankoff. 2017. Quantifying Aversion to Costly Typing Errors in Expert Mobile Text Entry. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 4229–4241. DOI: http://dx.doi.org/10.1145/3025453.3025695

[7] Xiaojun Bi, Tom Ouyang, and Shumin Zhai. 2014. Both Complete and Correct?: Multi-objective Optimization of Touchscreen Keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 2297–2306.

[8] Daniel Buschek, Benjamin Bisinger, and Florian Alt. 2018. ResearchIME: A Mobile Keyboard Application for Studying Free Typing Behaviour in the Wild. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, 255:1–255:14. DOI: http://dx.doi.org/10.1145/3173574.3173829

[9] Steven J. Castellucci and I. Scott MacKenzie. 2011. Gathering Text Entry Metrics on Android Devices. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems (CHI EA '11)*. ACM, New York, NY, USA, 1507–1512.

[10] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 785–794. DOI: http://dx.doi.org/10.1145/2939672.2939785

[11] James Clawson, Kent Lyons, Alex Rudnick, Robert A. Iannucci, Jr., and Thad Starner. 2008. Automatic Whiteout++: Correcting mini-QWERTY Typing Errors Using Keypress Timing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. ACM, New York, NY, USA, 573–582.

[12] Edwin S Dalmaijer, Sebastiaan Mathôt, and Stefan Van der Stigchel. 2014. PyGaze: An open-source, cross-platform toolbox for minimal-effort programming of eyetracking experiments. *Behavior research methods* 46, 4 (2014), 913–921.

[13] Kilian Förster, Andrea Biasiucci, Ricardo Chavarriaga, Jose del R Millan, Daniel Roggen, and Gerhard Tröster. 2010. On the Use of Brain Decoded Signals for Online User Adaptive Gesture Recognition Systems. In *Pervasive Computing*. Number 6030 in Lecture Notes in Computer Science. Springer Berlin Heidelberg, 427–444.

[14] Andrew Fowler, Kurt Partridge, Ciprian Chelba, Xiaojun Bi, Tom Ouyang, and Shumin Zhai. 2015. Effects of Language Modeling and Its Personalization on Touchscreen Typing Performance. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 649–658.

[15] Mayank Goel, Leah Findlater, and Jacob Wobbrock. 2012. WalkType: Using Accelerometer Data to Accomodate Situational Impairments in Mobile Touch Screen Text Entry. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 2687–2696.

[16] Mayank Goel, Alex Jansen, Travis Mandel, Shwetak N. Patel, and Jacob O. Wobbrock. 2013. ContextType: Using Hand Posture Information to Improve Mobile Touch Screen Text Entry. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 2795–2798.

[17] Dirk Goldhahn, Thomas Eckart, and Uwe Quasthoff. 2012. Building Large Monolingual Dictionaries at the Leipzig Corpora Collection: From 100 to 200 Languages.. In *LREC*, Vol. 29. 31–43.

[18] Joshua Goodman, Gina Venolia, Keith Steury, and Chauncey Parker. 2002. Language Modeling for Soft Keyboards. In *Proceedings of the 7th International Conference on Intelligent User Interfaces (IUI '02)*. ACM, New York, NY, USA, 194–195.

[19] Leanne M. Hirshfield, Philip Bobko, Alex Barelka, Stuart H. Hirshfield, Mathew T. Farrington, Spencer Gulbronson, and Diane Paverman. 2014. Using Noninvasive Brain Measurement to Explore the Psychological Effects of Computer Malfunctions on Users During Human-computer Interactions. *Adv. in Hum.-Comp. Int.* 2014 (Jan. 2014), 2:2–2:2. DOI: http://dx.doi.org/10.1155/2014/101038

[20] Kevin Anthony Hoff and Masooda Bashir. 2015. Trust in automation: Integrating empirical evidence on factors that influence trust. *Human factors* 57, 3 (2015), 407–434.

[21] Gernot Horstmann and Arvid Herwig. 2015. Surprise attracts the eyes and binds the gaze. *Psychonomic Bulletin & Review* 22, 3 (June 2015), 743–749. DOI: http://dx.doi.org/10.3758/s13423-014-0723-1

[22] Fotis P. Kalaganis, Elisavet Chatzilari, Spiros Nikolopoulos, Ioannis Kompatsiaris, and Nikos A. Laskaris. 2018. An error-aware gaze-based keyboard by means of a hybrid BCI system. *Scientific Reports* 8, 1 (Sept. 2018), 13176. DOI: http://dx.doi.org/10.1038/s41598-018-31425-2

[23] Andreas Komninos, Mark Dunlop, Kyriakos Katsaris, and John Garofalakis. 2018. A Glimpse of Mobile Text Entry Errors and Corrective Behaviour in the Wild. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct (MobileHCI '18)*. ACM, New York, NY, USA, 221–228. DOI: http://dx.doi.org/10.1145/3236112.3236143

[24] Per-Ola Kristensson and Shumin Zhai. 2005. Relaxing Stylus Typing Precision by Geometric Pattern Matching. In *Proceedings of the 10th International Conference on Intelligent User Interfaces (IUI '05)*. ACM, New York, NY, USA, 151–158.

[25] Olivier Ledoit and Michael Wolf. 2004. A well-conditioned estimator for large-dimensional covariance matrices. *Journal of multivariate analysis* 88, 2 (2004), 365–411.

[26] Perrin Margaux, Maby Emmanuel, Daligault Sébastien, Bertrand Olivier, and Mattout Jérémie. 2012. Objective and Subjective Evaluation of Online Error Correction During P300-based Spelling. *Adv. in Hum.-Comp. Int.* 2012 (2012).

[27] Stefanie Mueller. 2019. Inferring target locations from gaze data: A smartphone study. In *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*. 1–4.

[28] S. Joon Park, Craig M. MacDonald, and Michael Khoo. 2012. Do You Care if a Computer Says Sorry?: User Experience Design Through Affective Messages. In *Proceedings of the Designing Interactive Systems Conference (DIS '12)*. ACM, New York, NY, USA, 731–740. DOI: http://dx.doi.org/10.1145/2317956.2318067

[29] Felix Putze, Christoph Amma, and Tanja Schultz. 2015. Design and Evaluation of a Self-Correcting Gesture Interface Based on Error Potentials from EEG. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 3375–3384.

[30] Felix Putze, Dominic Heger, and Tanja Schultz. 2013. Reliable subject-adapted recognition of EEG error potentials using limited calibration data. In *6th International Conference on Neural Engineering*. San Diego, USA.

[31] Felix Putze, Maik Schünemann, Tanja Schultz, and Wolfgang Stuerzlinger. 2017. Automatic classification of auto-correction errors in predictive text entry based on EEG and context information. In *Proceedings of the 19th ACM International Conference on Multimodal Interaction*. ACM, 137–145.

[32] Philip Quinn and Shumin Zhai. 2016. A Cost-Benefit Study of Text Entry Suggestion Interaction. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 83–88. DOI: http://dx.doi.org/10.1145/2858036.2858305

[33] Germán Sanchis-Trilles and Luis A. Leiva. 2014. A Systematic Comparison of 3 Phrase Sampling Methods for Text Entry Experiments in 10 Languages. In *Proceedings of the international conference on Human-computer interaction with mobile devices and services (MobileHCI)*.

[34] Gerwin Schalk, Jonathan R Wolpaw, Dennis J McFarland, and Gert Pfurtscheller. 2000. EEG-based communication: presence of an error potential. *Clinical Neurophysiology* 111, 12 (2000), 2138–2144.

[35] Martin Spüler, Michael Bensch, Sonja Kleih, Wolfgang Rosenstiel, Martin Bogdan, and Andrea Kübler. 2012. Online use of error-related potentials in healthy users and people with severe motor impairment increases performance of a P300-BCI. *Clinical neurophysiology: official journal of the International Federation of Clinical Neurophysiology* 123, 7 (2012), 1328–1337.

[36] Andreas Stolcke. 2002. SRILM-an extensible language modeling toolkit. In *Seventh international conference on spoken language processing*.

[37] Bernhard Suhm, Brad Myers, and Alex Waibel. 2001. Multimodal Error Correction for Speech User Interfaces. *ACM Trans. Comput.-Hum. Interact.* 8, 1 (2001), 60–98.

[38] Hussain Tinwala and I. Scott MacKenzie. 2010. Eyes-free Text Entry with Error Correction on Touchscreen Mobile Devices. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries (NordiCHI '10)*. ACM, New York, NY, USA, 511–520.

[39] Chi Vi and Sriram Subramanian. 2012. Detecting error-related negativity for interaction design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. New York, USA.

[40] Martin Voelker, Sofie Berberich, Ecaterina Andreev, Lukas DJ Fiederer, Wolfram Burgard, and Tonio Ball. 2017. Between-subject transfer learning for classification of error-related signals in high-density EEG. In *The First Biannual Neuroadaptive Technology Conference*, Vol. 81. 47.

[41] Daryl Weir, Henning Pohl, Simon Rogers, Keith Vertanen, and Per Ola Kristensson. 2014. Uncertain Text Entry on Mobile Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 2307–2316.