Interactive Error Correction

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik des
Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Stefan Georg Constantin

aus Karlsruhe

Tag der mündlichen Prüfung: 03. Juli 2025
1. Referent: Prof. Dr. Alex Waibel
2. Referent: Prof. Dr. Josef van Genabith

Abstract

Natural language processing systems that have speech as input modality are used more and more by people in their day-to-day life. They are integrated, for example, in smart assistants on smartphones or in robot vacuum cleaners. Although these systems often have an impressive performance, they do make errors. Therefore, the integration of a component that enables users to correct errors per speech is essential.

Existing error correction systems are too limited to be used to correct all errors via speech. Most rely on redictation or inflexible correction patterns. The correct version of erroneous parts must be in the correction utterance. That is impossible for words that are not known by the automatic speech recognition system like foreign names. The alternative spelling is slow, annoying, and difficult for users with a spelling weakness.
There is one first work in the literature to offer an unconstrained error correction, but it is too slow and inaccurate. In this thesis, a fast and accurate enough unconstrained error correction component is proposed to overcome the limits of existing systems. The development of it proceeded in multiple steps.

First, different approaches to correct wrong entities were developed and compared to each other. A Telegram bot was developed to collect human created error corrections for testing. The quality of the approaches is evaluated in the following manner: All sentences that could be completely solved are counted. A correct solution contains the error-free corrections and the extraction of the reparandum and repair pairs. The counted number is divided by the total number of sentences and is the accuracy. An approach combining a fine-tuned sequence labeling model and a fine-tuned sequence-to-sequence model (104 714 training sentences were used to train both models) achieves an accuracy of 77.81 % on the test data.

To be able to correct errors beyond text, a multimodal error correction model was developed. Wrongly selected entities can be corrected by the combination of speech and pointing at entities. It relies on a region proposal component which outputs regions of interest (RoIs) and a pointing line generation component that is used to intersect the pointing line with the outputted RoIs to filter RoIs that are on the pointing line. These RoIs are passed to the multimodal error correction component. The multimodal error correction component outputs the RoI where the corrected entity is predicted and the name of the entity used in the utterance that led to the wrong prediction. In cases where a correct prediction is possible because the wanted entity is in a passed RoI, after one try 78.63 % and by using the interactiveness and up to three tries 88.46 % of wrongly selected entities were successfully corrected and the correct entity names were predicted.

The error correction was extended to all types of errors. Again, data were collected by one extended web application and one from scratch developed web application. A fine-tuned model could correct 63.14 % of all segments correctly. The Levensthein distance and the backtracing of its calculation are used to extract the changes. The from scratch developed web application can be used to run the system as real-world application.

An unconstrained error correction component using a large language model (LLM) with a custom prompt that describes the task and provides examples was developed to have not only an error correction that is bound on the data that were used to fine-tune the model. On the same dataset used for evaluating the fine-tuned model, the LLM is with an accuracy of 65.32 % only slightly better, real world tests with the web application showed a remarkable better quality on corrections that differ more from the corrections contained in the training dataset of the fine-tuned model. To test the system in a real-word scenario, it was integrated in the Lecture Translator framework and evaluated with 15 participants and distinct 24 cutouts of earning conference calls of S&P 500 companies and ACL conference talks. Although on average only 61.11 % of the errors of a given cutout could be corrected by the participants on the first try, after up to three tries on average 88.35 % of the errors of a cutout could be corrected and at the end of the correction tries on average 98.91 % of the errors of a cutout could be corrected. On average 2.39 tries (median 2.00) were necessary to correct an error of a cutout.

Zusammenfassung

Sprachverarbeitende Systeme, die gesprochene Sprache als Eingabemodalität haben, werden von immer mehr Menschen in ihrem alltäglichen Leben verwendet. Sie sind zum Beispiel in intelligenten Assistenten in Smartphones oder in Roboterstaubsaugern integriert. Obwohl diese Systeme oft eine beeindruckende Leistung haben, machen sie Fehler. Deshalb ist die Integration von einer Komponente, die Nutzern ermöglicht, Fehler per gesprochene Sprache zu korrigieren, essentiell.

Bestehende Fehlerkorrektursysteme sind zu eingeschränkt, um mit ihnen alle Fehler per gesprochene Sprache zu korrigieren. Die meisten beruhen auf dem erneuten Diktat oder unflexiblen Korrekturmustern. Die korrekte Version von fehlerhaften Teilen muss in der Korrekturäußerung enthalten sein. Das ist für Wörter, die nicht von dem automatischen Spracherkennungssystem erkannt werden, wie ausländische Namen, unmöglich. Das Buchstabieren als Alternative ist langsam, nervig und schwer für Nutzer mit einer Rechtschreibschwäche.
Es gibt eine erste Arbeit in der Literatur, die eine uneingeschränkte Fehlerkorrektur bietet, aber diese ist zu langsam und zu ungenau. In dieser Dissertation wird eine schnelle und genügend präzise uneingeschränkte Fehlerkorrektur vorgeschlagen, um die Limitierung von existierenden Systemen zu überwinden. Die Entwicklung verlief in mehreren Schritten.

Als erstes wurden verschiedene Ansätze, um fehlerhafte Entitäten zu korrigieren, entwickelt und miteinander verglichen. Es wurde ein Telegram-Bot entwickelt, um menschlich erzeugte Fehlerkorrekturen für das Testen kreieren zu lassen. Die Qualität der Ansätze wird evaluiert, indem gezählt wird, wie viele Beispiele komplett korrekt gelöst werden können. Eine korrekte Lösung beinhaltet die fehlerfreie Korrektur und die Extraktion der Reparandum und Repair Paare. Diese Anzahl wird durch die gesamte Anzahl an Beispielen geteilt und ergibt die Genauigkeit. Ein Ansatz, welcher ein feinabgestimmtes Sequenz-Beschriftungs (Sequence-Labeling) Modell und ein feinabgestimmtes

Sequenz-zu-Sequenz Modell kombiniert (104 714 Trainingsbeispiele wurden verwendet, um beide Modelle zu trainieren), erreicht eine Genauigkeit von 77,81 % auf den Testdaten.

Um Fehler, die über Text hinausgehen, korrigieren zu können, wurde eine multimodale Fehlerkorrektur entwickelt. Falsch gewählte Entitäten können mit einer Kombination von gesprochener Sprache und dem Zeigen auf Entitäten korrigiert werden. Diese beruht auf einer Region-Vorschlagskomponente, die Regionen von Interesse (Regions of Interest (RoIs)) ausgibt und eine Zeigeliniegenerierungskomponente, die verwendet wird, um RoIs, die auf der Zeigelinie liegen, herauszufiltern. Diese RoIs werden der multimodalen Fehlerkorrekturkomponente übergeben. Die multimodale Fehlerkorrekturkomponente gibt die RoI, in der die korrigierte Entität vorhergesagt wird und den Namen für die Entität, der zu der fehlerhaften Vorhersage geführt hat, aus. In Fällen, bei denen eine korrekte Vorhersage möglich ist, da die gewollte Entität in einer übergebenen RoI ist, konnten nach einem Versuch 78,63 % und beim Verwenden der Interaktivität bei bis zu drei Versuchen 88,46 % von falsch gewählten Entitäten erfolgreich korrigiert und die korrekten Entitätsnamen vorhergesagt werden.

Die Fehlerkorrektur wurde erweitert, um alle Arten von Fehlern korrigieren zu können. Es wurden wieder Daten gesammelt, mit einer erweiterten Webanwendung und mit einer von Grund auf entwickelten Webanwendung. Ein feinabgestimmtes Modell konnte 63,14 % aller Segmente korrekt korrigieren. Die Levensthein Distanz und Zurückverfolgung von ihrer Berechnung wurden genutzt, um Änderungen zu extrahieren. Die von Grund auf entwickelte Webanwendung kann genutzt werden, um das System als Echtweltanwendung laufen zu lassen.

Es wurde eine uneingeschränkte Fehlerkorrekturkomponente, welche ein großes Sprachmodell (Large Language Model (LLM)) mit einer maßgeschneiderten Prompt, welche die Aufgabe beschreibt und Beispiele bietet, entwickelt, um nicht nur eine Fehlerkorrekturkomponente zu haben, die durch die Daten, mit der sie feinabgestimmt wurde, begrenzt ist. Auf dem gleichen Datensatz, der für die Evaluation des feinabgestimmten Modells verwendet wurde, ist das LLM mit einer Genauigkeit von 65,32 % etwas besser. Praktische Tests mit der Webanwendung zeigten aber eine erheblich bessere Qualität bei Korrekturen, die mehr von den Korrekturen, die im Trainingsdatensatz vom feinabgestimmten Modell enthalten sind, abweichen. Um das System in einem Echtweltszenario zu evaluieren, wurde es in das Lecture Translator Framework integriert und mit 15 Teilnehmern und 24 unterschiedlichen Ausschnitten von Ergebniskonferenzen von S&P 500 Unternehmen und ACL-Konferenzvorträgen evaluiert. Obwohl im Durchschnitt nur 61,11 % von den Fehlern eines gegebenen Ausschnitts von den Teilnehmern bei dem ersten Versuch korrigiert werden konnten, konnten nach bis zu drei Versuchen im Durchschnitt 88,35 % von den Fehlern eines Ausschnitts korrigiert werden und nach dem Ende der Korrekturversuchen konnten im Durchschnitt 98,91 % der Fehler eines Ausschnitts korrigiert werden. Im Durchschnitt waren 2,39 Versuche (Median 2,00) nötig, um einen Fehler in einem Ausschnitt zu korrigieren.

Acknowledgement

I would like to thank my advisor Prof. Dr. Alex Waibel. He made this thesis possible and accompanied me all the years on my research journey, shared his experience of three decades research on this topic and gave me invaluable ideas and feedback on my research.

Thanks a lot to Prof. Dr. Josef van Genabith for officiating as my second adviser and his friendly and motivating support.

I also want to thank Prof. Dr. Raffaela Mirandola, Prof. Dr. Katja Mombaur, and Prof. Dr. Ralf Reussner for the doctoral interviews. At the KIT, it is a requirement to have at least two doctoral interviews. These doctoral interviews gave me helpful feedback on my research and the presentation of my research.

For the supervision of my master thesis and introducing me to the Interactive Systems Lab and to Prof. Dr. Waibel, I thank Prof. Dr. Jan Niehues and Dr. Eunah Cho. Prof. Dr. Jan Niehues accompanied my research even after my master thesis and advised me on how to improve my research.

Thanks to Christian Huber for introducing me to the Lecture Translator framework and explaining me the details of it. To embed my error correction into this framework, the framework needed some adaptations and his deep understanding of the Lecture Translator framework and his help simplified the embedding for me.

Another thank you to Fevziye Irem Eyiokur, Dogucan Yaman, and Leonard Bärmann for helping me in my multimodal error correction research.

I also want to thank all my colleagues and former colleagues at the Interactive Systems Lab and the Artificial Intelligence for Language Technologies lab for fruitful discussions, testing my data collections and systems, and helping in other ways: Seymanur Akti, Maximilian Awiszus, Moritz Behr, Silke Dannenmaier, Florian Dessloch, Tu Anh Dinh, Prof. Dr. Hazim Ekenel, Mia Fuß, Dan He, Dr. Teresa Herrmann, Lukas Hilgert, Juan Hussain, Laura Kernahan, Sai Koneru, Danni Liu, Yining Liu, Zhaolin Li, Prof. Dr. Mohammed Mediani, Dr. Markus Müller, Carlos Mullov, Thai Binh Nguyen, Dr. Thai-Son Nguyen, Tuan-Nam Nguyen, Dr. Ngoc Quan Pham, Fabian Retkowski, Margit Rödder, Dr. Kay Rottmann, Christian Schmidt, Felix Schneider, Ilya Shibayev, Supriti Sinhamahapatra, Dr. Matthias Sperber, Dr. Sebastian Stüker, Dr. Thanh-Le Ha, Enes Yavuz Ugan, Franziska Vogel, Thomas Zenkel, and Maike Züfle.

Thanks to my students who I supervised in theses and seminars and for their participation in the lectures or exercises I gave. Learning is not a one-way road, supervising and teaching taught me a lot.

I would like to thank Dr. Thanh-Le Ha, thanks to him I could meet my wife Hoa. Thanks to my beloved wife Hoa for supporting me all the time and being such a great mother for our son Theo. Thanks to my mother Katharina and my father Richard for supporting me since my birth. I hope my wife and I can support Theo in the same way.

Thanks for the financial support from the following institutions: This work (Chapter 3, 4, and 5) has been supported by the German Federal Ministry of Education and Research (BMBF) under the project OML (01IS18040A).
This work (Chapter 4) has been supported by the Baden-Württemberg Ministry of Science, Research, and the Arts (MWK) as part of the state's "digital@bw" digitization strategy in the context of the Real-World Lab "Robotics AI".
Thanks to the Karlsruhe Institute of Technology that I could be part of it.

There are so many more people that contributed in one or another way like the conference organization teams, the staff for network infrastructure, or the cleaning staff that cleaned our offices. Thank you to everyone.

Contents

List of Figures

List of Tables

1. Introduction

1.1. Motivation

Natural language processing systems that use speech as input should make our life easier. Speech controlled smart assistants on smartphones can answer questions or execute tasks given by the user. To send a message to someone, you can instruct your smart assistant to do so, dictate the text and instruct to whom it should be sent. To communicate with a person who has no language in common with you, you can dictate what you want to say and it can be translated into a language that this person understands. Through robot systems, not only virtual tasks in a smartphone but physical tasks can be controlled by speech. Errors can occur on all components used in a natural language processing system, for example, the automatic speech recognition (ASR) component can misrecognize some words or the machine translation component can translate something incorrectly. Therefore, the option to correct errors is essential.

Research in the area of error correction started decades ago [95], despite some impressive advancements [144, 47], the speech interfaces are too limited. Only certain correction patterns are allowed, and therefore a lot of systems relied on other modalities than speech as addition [144, 47]. To avoid a disruption in the input modality, the error correction should only be controlled by speech if the natural language processing system is usually controlled by speech. Sometimes, users have reasons to use speech controlled systems, for example, other modalities are difficult and dangerous to handle during workout or while driving a car. However, error correction via speech is hard. An error correction system that relies on repeating misrecognized or not recognized phrases because it requires redictation or using patterns like “Replace X with Y” can get problems to correct errors like foreign names. Often there is a reason why the recognition did not work. To make it worse, users sometimes tend to do hyperarticulation [89] that could make the recognition worse [129].

To rely only on spellings also has some disadvantages. It is slow, annoying, and for users with a spelling weakness a description of the word can be easier than the spelling.

A solution for the above mentioned limitations is an unconstrained error correction component. Different error correction strategies can be used, users can be creative and can use what works the best for them. Possible strategies could be redictation, spelling, describing the desired adjustments to change a word to another like “Stephan with F” or describe the word to change “I don’t mean invidia but the company that produces GPUs”.

With the availability of an unconstrained error correction component, the mentioned multimodality is not necessary for textual corrections anymore, but it can be useful for corrections beyond text. Via speech and pointing gestures falsely selected entities could be corrected by pointing at them.

To have a good error correction system offers not only the ability to have correct outputs but to obtain the erroneous part, the correction, and the corrected part. These three elements can be used to learn from errors. Learning from errors is for the development of humans essential [53, 148] and should also be essential for systems to adapt to changing environments.

1.2. Contributions

Error correction is a big field with many subtasks. This work offers the following contributions to this field.

First, a web application that offers users the ability to dictate texts and to activate a correction mode where the users can select erroneous parts and can correct them by speech. Users can add, remove, or replace phrases, spell words, change individual characters, describe the correction, for example, "plural instead of singular", or express whatever they want to be done with the selected text. There are no limitations for the correction utterances. Even multiple languages can be used. For example, it is possible to correct English transcripts in German. A combination of German and English corrections is also possible. Among others, this feature can be helpful for nonnative speakers, for example, when the pronunciation is incorrect and leads to errors in the transcript. In this case, it can be more efficient for nonnative speakers to use their mother tongue for certain parts of the correction like spelling a word with the pronunciation of the alphabet letters in their mother tongue. This system allows not only to correct errors in transcripts but to collect the errors and their changes. In addition, the system has alignments of the parts that are changed and to what timestamps of the speech input they belong which enables the system to build a database that could help to improve the components (ASR, machine translation, ...) that cause the errors. The focus lies on multilingual unconstrained error correction since this is a big missing point in the current state of research. First steps to such an unconstrained verbally error correction were only made in [84], but their system has latency problems and is not accurate enough to be used to do user studies. Before, only smart redictation, spelling and other modalities like handwriting were used [144, 147].

Second, a multimodal error correction system that could be run on a robot like the ARMAR-6 [4] and offers the ability to correct via speech and pointing at the target object. For example, users could request an action from the robot that includes a certain object like "I want the gala on the table". However, it is possible that the robot targets a wrong object like the magazine Gala instead of the apple cultivar gala. With the help of the multimodal error correction system, a correction like "I meant this gala" together with the pointing at the apple changes the target object from the magazine Gala to the apple cultivar gala. The system is interactive and supports multiple correction tries if the correction does not work right away. Robot systems must be correctable to learn new things or to adapt to changing environments. It would be very annoying to ask every time by the smallest uncertainty (the goal is “put Computers in the Human Interaction Loop” [162] and not the opposite of it) and even then, a certain action can also be wrong although the robot is confident that it is correct.

Third, this work compares different approaches for error correction like fine-tuned models and large language models on a collected dataset to show the strengths and weaknesses of the different approaches.

Fourth, to collect the data for the multiple datasets, multiple collection systems were created and are shown in this work which also serve as demo systems for the final systems. Using a Telegram bot, one person can speak a text which will be transcribed either by an ASR system or by a human using another Telegram bot (Wizard-of-Oz setting [72]). Then, the person who spoke the text can give a correction to the transcript which will be used to correct the transcript again by either a system or by a human using another Telegram

bot.
A web application that offers the option to add corrections to subtitles and a web application that is similar to the Telegram bot chat app on which one side can correct errors and the other side (can be a human in a Wizard-of-Oz setting or a system) carries out the given corrections.

1.3. Disclaimer

To improve the sentence quality, in Chapter 3 and 4, GPT-4o [101] was used to improve the quality of the sentences by having it paraphrasing my own sentences. Every sentence was processed individually without other sentences as context.

My doctoral thesis that was the foundation for my doctoral oral examination was submitted on May 8, 2025. To have time for proofreading, this thesis is based on the state of art of April 16, 2025. After the oral examination, some improvements were made for the final publication but no additional methodologies, experiments, literature published after April 16, 2025, chapters, or sections were added.

2. Background

2.1. Error Correction

Error Correction Task

There are multiple terms for the task of editing output of a natural language processing system. Some of them limit the editing capabilities. A very broad term is editing. If the output that should be edited is in text form, it can be called text editing. The premodifier post limits the editing phrase to the point of time after the output is finished. Editing often has the goal of eliminating or mitigating errors, therefore the terms error correction or error repair are often used for such a kind of editing. Schegloff et al. [120] state that the term repair is considered broader because the term correction would imply that some errors have been corrected and that repair is not limited to replacements. However, in a lot of successor literatures correction is laid-out as broad as repair [143, 126, 147] and an insertion can theoretically also be a replacement if the replaced element is seen as empty element and a deletion can theoretically also be a replacement if the element that replaces other elements is seen as empty element. A replacement is also called substitution.

The element (e. g., word, phrase, entity that represents an object) to be replaced is called the reparandum and the element that replaces the reparandum is called repair [120] or reparans [133]. The repair often has an interregnum that is a phrase that signals the repair, e. g., the word “uh”, this interregnum or if not present, the repair is introduced with the interruption point [128]. In [128], the interruption point and the interregnum refers to the “moment of interruption” and “editing phase” in [81], respectively, and to the “cut-off” and “cut-off to repair” in [11], respectively.

The interruption point can be directly after the reparandum, see Figure 2.1, or there can be other elements between the reparandum and the interruption spoon into the drawer uh sink [23]

reparandum interruption pt. interregnum repair

Figure 2.1.: Repair example in which the interruption point follows the reparandum

spoon into the drawer no forks [23]

reparandum interruption pt. interregnum repair

Figure 2.2.: Repair example in which the interruption point does not follow the reparandum

point, see Figure 2.2 [81]. The distance, for example measured in number of words or seconds of speech, is one dimension in that an error repair can be categorized. There can be cases where it is not enough to replace the reparandum with the repair for example when the repair is spelled or is a description like "start the word german with an uppercase letter", see Figure 2.3. In Figure 2.4 an example for a verbal correction for a wrong robot action instead of a wrong text is given.

The type of input and output for the natural processing system that is corrected is also a categorization dimension. In an automatic speech recognition (ASR) system or in a speech translation system with textual output [164], the input is speech and the output is text, the textual output can be corrected, in a robot system the output could be an action like bringing some specific object where the action or the object can be corrected, and in a handwriting recognition system the input is the handwriting and the output is the recognized text which can be corrected. The repair can also have different types. A

german cars no german in uppercase letter

reparandum interruption pt. interregnum repair

Figure 2.3.: Repair example in which the repair must be transformed

$$\text{BRINGING } \underbrace{\text{GALA\_MAGAZINE}}_{\text{reparandum}} \underbrace{\qquad}_{\text{interruption pt.}} \underbrace{\text{no}}_{\text{interregnum}} \underbrace{\text{the apple cultivar gala}}_{\text{repair}}$$

Figure 2.4.: Repair example for correcting a robot action

correction speech, a pointing gesture, a correction of the initial handwriting [64, 65], or any other modality that is suitable to correct something could be used. Combinations of modalities are also possible.

A further dimension is who makes the error repair. Is it a self-repair by the person who made the error or an other-repair by another person [120]. Who initiated the repair is another dimension. The repair can be either self-initiated or other-initiated [120]. Gieselmann [41, 42] created with these two presented dimensions and their values following classification for a human-to-human conversation: a self-initiated self-repair is a self-rectification, a self-initiated other-repair is a request for help, an other-initiated self-repair is a repair to a clarification question, and an other-initiated other-repair is an other-initiated other-repair. In the context of human-to-machine conversations, the transcript of a human speech is a performance of the machine and a correction of the transcript should be classified as other-initiated other-repair. Who detects an dead and situation should try to initiate the repair of it, different strategies for this are presented in [55]. An error correction can also be categorized whether a non-understanding or a misunderstanding [43] is corrected.

Systems for Self-initiated Self-repair

The reparandum, interregnum of a self-initiated self-repair can be seen as disfluency. If the repair contains not only the repair but other elements like a description of the repair, see the previous discussion and Figure 2.3, the other elements can also be seen as disfluency. In [78], it is claimed that nearly all non-fluent sentences could be transformed into full grammatical sentences with a set of rules. This was criticized in [51] as not formally adequate, because it ignores syntactic constituenthood. A deterministic parser that needs an external edit signal for self-corrections is provided instead. Shriberg et al. [127] combine pattern matching, syntactic and semantic analysis, and acoustics for self-corrections without external edit signal. Nakatani et al. [97] adds cues based on acoustic and prosodic analysis to detect the interruption point.

Systems for Repair

In the following, the most relevant systems (first to introduce new techniques) that are able to correct textual errors via speech or other natural modalities in contrast to corrections only via keyboard are presented in historically order (except if systems build on each other, all these systems are presented in one piece). In the end, it is summarized what different techniques for error correction were developed.
A multimodal text editor that can be controlled by speech and pen gestures is presented in [156]. For the speech input, a word spotter with a vocabulary of 11 words and for the pen gesture input, a gesture recognition component with 8 gestures were trained. This constrained setting only allows actions like deleting and moving of text, but no addition of new text or substitution of text.
Storywriter [29] is a text editor offering dictation and users can select a text location via mouse pointing (without clicking to be used by users with repetitive stress injury) and to this location an action like "delete", "copy", or "insert cursor here" can be commanded via speech. A combination of "delete" and "insert cursor here" with a dictation is equivalent to a substitution.
McNair and Waibel [95, 163] present an approach that offers to repair an utterance by a repair utterance. In the repair utterance only the erroneous part of the initial utterance must be respoken or spelled. That is possible by using ASR systems that output n-best lists and an intelligent matching algorithm that uses the repair utterance to find the erroneous part in the initial utterance. If the correct utterance is not in the n-best list, the approach will not work.
In [141], a multimodal approach is employed where users could indicate incorrect phrases and correct them through respeaking, spelling, selecting an alternatives from the n-best list of the ASR component, or using handwriting. The use of gestures to delete words or create insertion points for new words to the previous described system was added in [142]. The strategies for error correction presented before have been further enhanced in [145] by considering the context.

In the previous presented system, the smallest correctable element is a word, in [140] partial words are correctable.
In 1998, commercial ASR systems were available that have integrated command recognition that allows to select errors and redictate. However, wrongly recognized commands and renewed misunderstandings made additional errors and therefore it is more frustrating for the users [70].
The system developed in [140] is evaluated in [143] with a user study. The result shows that although the multimodal error correction is faster than conventional correction without keyboard input (respeak and choose from n-best list), correction by keyboard is faster by skilled typers. In [144, 166], a system-initiated error correction by confidence measures is presented. However, the results are that this slows down the correction performance. A reason is that the used confidence models are unreliable.
The alternative list that contained all values of the n-best list of the ASR system is improved in [79]. Instead of using only the n-best list, homophones and quasi homophones are added with the help of a phone decoder and an inverse text normalization adds not normalized words to the list.
A hands-free interaction with Microsoft Word with gaze and speech is presented in [9]. Gaze enables navigation and selection; speech enables in dictation mode dictation and in command mode commands like formatting and selection. Selections enable corrections by overwriting.
In [107], elements from the n-best list could be selected by saying the corresponding index number or by gaze. Redictation updates the n-best list. The part to be corrected could also be selected via gaze.
In [75], it is suggested to rely on hyperarticulated speech to detect error corrections. By using such a system, it is important to use an ASR system whose performance is not decreased by hyperarticulated speech which is very difficult for many ASR systems [130, 129].
In [49], the cognitive load was measured. A study in [48] showed that pen, touch, and speech input could benefit the post editing task the most. Eye tracking and gestures do not seem to be so beneficial. Based on these insights the MMPE (multimodal post editing) system was developed [45]. It combines pen handwriting, touch, and speech. Pen handwriting can be used on one side to add new text and on the other side for gestures like "deletion" with a strike-through gesture. Via touch, words can be deleted and moved around. Via speech the user can insert, delete, replace, and rearrange words and subphrases. In [46], the classical mouse and keyboard are added to the previous described system. Based on feedbacks from professional translators, in [47] new features like the visualization of whitespace or allowing synonyms for

speech commands to reduce the cognitive workload to memorize the exact command keywords are added.
In [1], a system that use hand gestures and keyboard instead of mouse and keyboard is presented. The hand gestures can be used to select, delete part of the text or to place the cursor.
In [126], a word can be selected via speech by its index number and the entry from the n-best list for this word can also be selected by saying the index number or spelling can be used if it is not in the n-best list. Alternatively, words can be selected by gaze.
In [39], a smart glasses text editing solution is presented. Text could be selected via a hand-controller and replaced by redictation via speech.
A system for eyes-free text editing is presented in [40]. Transcribed text is read out (generated by a text-to-speech component) and users can navigate via speech to the text that should be read out and where the correction should occur. There are two modes. In the redictation mode, only a part of the sentence (including the correct phrase that should replace the erroneous part) needs to be redictated and an algorithm searches the corresponding location in the text for the substitution and in the commanding mode, there are commands like "insert", "delete", "change", "undo", "go to location", and "repeat audio playing". The commands have parameters like the location and the content. The processing of parameters is done by Dialogflow [92] to enable natural commands but no real open domain corrections are possible.
Another eyes-free system is presented in [35]. The text is divided in segments and the segments can be navigated through the buttons on the headset. Playback a segment via text-to-speech component is also controlled by the buttons. An erroneous part of a segment can be replaced by a correction. A trained GPT-2 model [112] finds the correct location and correct it. Due to the small dataset, the possible corrections are not really unconstrained and are more like a redictation with finding the correct location.
In [176], text is selected by touch and then can be redictated with providing more context than the actual selection has. A new alignment algorithm replaces only the erroneous part.
In [84] a system with the goal of unconstrained edit commands is presented. The ASR output is segmented. To each segment it is classified whether it is dictation or a correction command. The segmentation and the classification is done by a T5 model [113]. A segment that is a dictation is appended to the document, a correction updates the document. The update can be carried out directly or via a Lisp similar code that is generated and executed by a provided execution engine. For the update, there is either a T5 or a GPT-3 model [12].

No user studies have been done, because the system is not production ready, the more accurate GPT-3 model is too slow and both models are not accurate enough.
In [147], a Word plugin with gaze, speech, and mouse modality is presented but correction via speech is still limited with commands and replace templates.

In summary, the text to be edited can be selected by mouse, touch, spoken index number, hand controller, gaze, or intelligent matching based on the correction. For the correction, selecting from a list, redictating, spelling, handwriting, and spoken corrections that are limited to some patterns are possible.

In addition to text output, wrong indents should also be correctable. In [85], a system to repair the outputted indent is presented.
In [8], robot actions can be corrected or additional information can be given that changes the behavior of a robot.

In the previously presented systems multimodality is only used to correct text that theoretically could be corrected only by speech and the multimodality should only improve the performance. However, there are situations by interacting with entities like objects where multimodality is needed or very long descriptions would be necessary (everything can be described). To correct an entity, it is much easier to point at the object than to explain the location exactly. Such a pointing can help to connect words and the image of an entity. To learn new objects by a user that is showing them and explaining them, a system is presented in [56]. However, a pointing is not enough, the objects have to be shown to the system. It can be pointed at objects to learn them by a robot, but no natural language is used in the method presented in [154]. In [152], pointing gestures and natural language are combined to learn new objects. However, the robot initiates the learning and is not correctable. Users can initiate the learning by pointing to an object and explaining it in [6], but this is an action and not a correction to an action that is executed.

Focus of Attention

The focus of attention of users can be used to select the erroneous part.

In [121], a gaze system that allows a broader location of the user than in works before is presented. The user can move in a room and the system outputs the focus of attention of this user.
Such a broad location gaze system is combined with a microphone array in [173] to use the gaze system to extract the speech from the tracked person. However, the other way around is also possible, in [136, 138] the location of the sound improves the attention tracking. Visual and audio cues can also be used in both directions to get the location and this location can be used to extract the sound [13]. According to [134], there is no necessary coincidence of attention of a person and the gaze, therefore they developed a system that considers persons head movements and the relative locations of probable targets of interest.
Such an attention system was extended to multiple persons in [137].

The techniques developed more than 20 years ago became smaller and more production ready and can be bought as ready devices today.

Focus of attention tracking can not only be used to select erroneous parts, but can be applied to other fields like robot interaction [135] and smart rooms [159, 161] as well.

Continual Learning

In [60], a system where new words can be added to a memory and the ASR considers these new words is presented. A more in depth analysis of this system, on how much better the added words are recognized and whether this is traded with a high false positive rate is shown in [61]. Instead of only attend to the added words, in [63], a system that adds and trains adaptation weights is shown.
In [174], a gradient episodic memory method is presented to continually learn new data in an ASR system.

In [86], a method of continual learning for image classification without the need to re-train on old data is presented.

2.2. Neural Networks

For the development of today's dominant neural network architecture the Transformer architecture [153] a lot of developments in the area of neural networks were necessary. Starting with the perceptron [94], an algorithm to learn the parameters of a perceptron [118], the training of a multilayered feed-forward network [2], the backpropagation algorithm [88, 167] to train complex neural networks, recurrent neural networks that can have connections back in contrast to the feed-forward networks [94, 34, 69] and are trained with backpropagation through time [119, 168], going to more advanced structures like the time delay neural network [158, 160, 165], long short-term memory [54, 38], attention-based encoder-decoder [7], Memory networks [169, 19], and a lot of other unmentioned architectures [80].

Neural Network Training

The backpropagation algorithm [88, 167] is still used for modern neural network architectures like the Transformer architecture. A differentiable loss function outputs a value that represents the difference of the predicted output and what should be outputted. The gradient of the loss function is propagated back with respect to all network parameters via chain rule. Based on this backpropagation, the parameters are adapted to minimize the output value of the loss function. How much the adaptation per step is depends on the learning rate which can be fixed or variable. Optimizers like stochastic gradient descent (SGD) [119] and Adam [73] calculate a learning rate for every parameter. Adam needs more memory per parameter than SGD, because in addition to the model parameter and the gradient, the first and second moment estimate must be saved. To avoid oscillations, parameters are often adapted after multiple inputs (mini-batch). The backpropagation algorithm can only guarantee to converge to a local minimum.

Transformer Architecture

The Transformer architecture [153] consists of an encoder and a decoder. One of the key parts of the Transformer encoder as well as the Transformer decoder is the multi-head attention. It has three input parameters: the query

Q, the key K, and the value V. The query has n and the key and value have m vectors with dimension d_{model} that form a matrix. In the domain of natural language processing, such a matrix could consist of n / m subword token embedding vectors. For every input parameter, there are h linear layers (multi-head). These linear layers are vector matrix multiplications, where the matrices are adaptable network parameters. The Q and K inputs are mapped to h matrices with the dimension $n \times d_k$ and $m \times d_k$, respectively, and the V input is mapped to h matrices with the dimension $m \times d_v$. The mapped query matrices and the transposes of the key matrices are multiplicated with each other and the results are h matrices with the dimension $n \times m$. To scale the output matrices all values are divided by $\sqrt{d_k}$. A mask can be applied optionally. After this, the softmax function is applied. The result matrices are multiplicated with the mapped value matrices and the results are h matrices with the dimension $n \times d_v$. All h result matrices are concatenated and a linear layer with the dimension $hd_v \times d_{model}$ maps it to a matrix with dimension $n \times d_{model}$ that is the output of the multi-head attention.

The Transformer input vectors go through an embedding layer and a positional token is applied to the embeddings. These positional encoded embedding vectors are fed into the Transformer encoder. Multiple encoder layers can be stacked. Therefore, the input and the output of an encoder layer have the same dimension $n \times d_{model}$. An encoder layer has first the previous described multi-head attention. The input parameters Q as well as K and V have the same values as the input of the encoder layer. Therefore, it is called self-attention. The output of the multi-head self-attention is added to the input of the encoder layer and normalized. The output of the normalization is the input of a two layers feed-forward network that uses the Rectified Linear Unit (ReLU) activation function after the first layer. The output of this feed-forward network is added to the input of the feed-forward network and the sum is normalized and forms the output of an encoder layer. In the content of Natural Language Processing, the n output tokens represent their n input tokens but with contextual information integrated.

The output tokens of the Transformer are fed into the same embedding layer as the input tokens and the same positional embedding is applied. Like in the encoder, multiple decoder layer can be stacked and the in- and outputs are vectors of the dimension d_{model}. First, masked multi-head self-attention is applied to the input of the decoder layer to ensure that no future output information is available. Like in an encoder layer, the output of every sublayer of a decoder layer is added to its input and the sum is normalized.

The normalized output forms the query Q and the output of the encoder forms the key K and value V. These three input parameters are used for a multi-head attention that is again followed by the addition with its input and normalization. The normalized result is the input of a feed-forward network with the same architecture as in the encoder. Again, the input of this feed-forward layer is added to the output and normalized. The output of the last decoder layer is feed into a linear layer and a softmax is applied to get the output probabilities of the vocabulary.

The Transformer architecture that waives recurrence and convolutions can be computed very well in parallel and can utilize mass parallel architectures like GPUs and reduce computation time and also have good prediction results.

Pre-trained Transformer Variants

A problem of neural networks is that they often need a lot of training data to be able to predict correctly. There are two approaches to mitigate this: multi-task learning [14, 21] and pre-trained models. The pre-trained models became the dominant approach. They are pre-trained with a lot of data and have acquired a good amount of general knowledge. In the area of Natural Language Processing, pre-training enables them to have already a good knowledge about language and general facts. In the fine-tuning process, they only need to learn the in-domain problems and that requires less data.

There are different kind of Transformer models that are pre-trained. There is the encoder-only pre-trained model Bidirectional Encoder Representations from Transformers (BERT) [31]. Every input is prefixed with a special token that represents the complete sentence ([CLS] token). A linear layer and a softmax function on top of the encoder output vectors map these vectors to the predicted output probabilities over the vocabulary. Depending on the task, only the output token that represents the [CLS] token may be necessary to map to the predicted output probabilities over the vocabulary, for example in a sentiment analysis of a sentence. BERT is pre-trained with about 137 billion WordPiece [171] tokens with two tasks: masked language model and next sentence prediction. In the masked language model, 15 % of the tokens are selected. 80 % of the selected tokens are replaced by [MASK], 10 % by a random token, and the remaining selected tokens are unchanged. The task is to predict the selected tokens. In the next sentence prediction task, the inputs are two sentences and the model has to classify whether the second sentence is the successor sentence of the first sentence.

A pre-trained encoder-decoder Transformer model is Text-to-Text Transfer Transformer (T5) [113]. The idea is to prefix every input sentence depending on the task like machine translation or sentiment analysis and produce the output as text. For machine translation, it would be the translated sentence and for sentiment analysis the sentiment. It was pre-trained with about 34 billion SentencePiece [76] tokens in the following way: Like in BERT, 15 % of the input tokens are selected to drop-out. Coherent drop-out tokens are replaced by a sentinel token. The target is to output the sentinel tokens, every sentinel token followed by the drop-out tokens that it represents. For example, in the sentence "Today is an important day for the team", could be "is an" and "day" dropped out, therefore the input is "Today <X> important <Y> for the team" and the target is "<X> is an <Y> day".

In [32], it is shown that the Transformer encoder can be used for image processing. Every image is split into fixed-size patches, a linear projection is applied and on this projection a position embedding is applied. Like in BERT a prefix token is used to represent the whole input. The output of this token is used for adding a feed-forward network for classifying the input image. T5 was extended by vision to the Vision Language T5 (VL-T5) [15]. It was pre-trained on 9.18 million image-text pairs with 180 thousand distinct images. It used multimodal language modelling, where the input is a text describing an image and in the text tokens are replaced by sentinel tokens that must be restored, visual question answering, image-text matching, visual grounding (to textual content the most fitting region should be outputted), and grounded captioning (the content of the asked region should be outputted).

Decoder-only Transformer models are language models (LMs).
The decoder-only Transformer generative pre-trained Transformer (GPT) [111] is pre-trained with the BooksCorpus dataset [177]. The multi-head attention after the masked multi-head attention of the Transformer decoder of a Transformer encoder decoder is not used in GPT. As an optimization technique, a key-value cache can be used to avoid to calculate the complete key and value matrices after every new output; only the column that represents the new output word needs be calculated [106]. The pre-training task is the next token prediction. GPT has 117 million parameters. GPT-3 [12] parameter size increased to 175 billion parameters. This huge parameter size leads to the expression large language model (LLM). GPT-4 [100] improved the

quality. Lots of details like the number of parameters are officially disclosed. In GPT-4o [101], the ability to process audio and vision is added.

3. Error Correction of Wrong Entities

In this chapter GPT-4o was used to improve the sentences, more information about it is in Section 1.3.

3.1. Introduction

The introduction in this section is also presented in [23, 22]. The Association for Computational Linguists (ACL) is copyright holder (Copyright © 2022 Association for Computational Linguists (ACL). All Rights Reserved.) and Creative Commons Attribution 4.0 International Public License (CC BY 4.0) licensor of the work in [23]. The copyright holders of [22] are the authors, under exclusive license to Springer Nature Switzerland AG and the used parts of [22] are reproduced with permission from Springer Nature.

The process of correcting errors plays a vital role in human-machine communication via natural language, as errors and ambiguities are often inevitable. For example, consider a scenario where a household robot is tasked with the following command "Please put the washed forks inside the cutlery drawer". The robot lacks knowledge about the specific drawer assigned to hold the cutlery. A drawer is chosen, and the forks are placed inside it. If the robot makes an incorrect selection, the user is required to provide clarification by specifying instructions, such as saying, "No, into the drawer left of the fridge". Alternatively, the robot has the option to directly ask the user for the location of the cutlery drawer. The user's reply, like saying "It's the drawer left of the fridge", also acts as a correction because it is addressing the uncertainty and corrects it. Another type of correction takes place when individuals revise their original input, expressing something like "I changed my mind, the forks", or when the system misunderstands the individual's request because of mistakes in speech recognition or language processing.

These various types of corrections can be processed in the same way. As a result, a component is proposed that processes a request alongside a correction as input and generates an adjusted request as output. The correction substitutes the relevant expressions found in the initial request. This chapter is centered exclusively on substituting entity phrases, such as "cutlery drawer" with "drawer left of the fridge", whereas other phrases, including verb phrases, fall outside its scope. For example, a request such as "Please put the washed forks inside the cutlery drawer" followed by a correction like "No, in the drawer left of the fridge" would be corrected to ""Please put the washed forks inside the drawer left of the fridge".

This type of correction mechanism provides two benefits compared to managing corrections directly within the primary dialog module. To begin with, the need for extensive training data for the dialog component is minimized, as the presence of an open-domain correction mechanism eliminates the necessity of learning corrections. Secondly, this type of correction mechanism has the potential to be extended to generate pairs consisting of reparandum and repair elements. In the given example, the pair could be described as "cutlery drawer" and "drawer left of the fridge". These entity pairs can be applied, for example, within a dialog system's lifelong learning module to reduce the necessity for corrections in the future. As a result, the robot is capable of identifying the particular drawer designated for storing cutlery.

3.2. Methodology

The methodology in this section is also presented in [22] and the version without the GPT-3 approaches is also presented in [23]. Information about the copyright holders and the licenses of [23, 22] is in Section 3.1.

Six approaches were created to address error correction and the extraction of reparandum and repair pairs. The first approach assigns labels to sequences, whereas the second approach produces sequences by sampling output tokens from a fixed vocabulary. The third approach involves generating sequences by directly copying tokens from the source sequence into the target sequence. The last three approaches rely on the GPT-3 model. The first GPT-3 [12] approach relies on a fixed prompt applied to all test pairs, the second adapts the prompt dynamically to suit each test pair, and the third forgoes prompts entirely, opting instead for fine-tuning.

Under the sequence labeling approach, every word is categorized with one of the following labels: C (indicating copying), D (indicating deletion), R1 (representing entity 1, which may be subject to substitution), R2 (representing entity 2, which may also be subject to substitution), S1 (denoting the entity intended to substitute entity 1), or S2 (denoting the entity intended to substitute entity 2). For the correction target, entities labeled as S1 and S2 are utilized to substitute those marked as R1 and R2, respectively. With respect to the extraction target, the output includes pairs such as R1 and S1, along with R2 and S2, provided a substitute exists for either the first or second entity. Figure 3.1 illustrates an example in which a request is paired with its corresponding correction, accompanied by labels and the resulting extraction target.

request || correction	put the milk into the shelf	no the soy milk into the left shelf
labels	C R1 R1 R2 R2 R2	D S1 S1 S1 S2 S2 S2 S2
corrected request	put the soy milk into the left shelf	
pairs of reparandum and repair entity	milk → soy milk ; into the shelf → into the left shelf	

adapted from [23] (© 2022 ACL, CC BY 4.0) and
[22] (Reproduced with permission from Springer Nature)

Figure 3.1.: Error correction example

To apply the sequence labeling approach, fine-tuning the cased BERT large model is suggested. This model is composed of 24 Transformer encoder layers, features a hidden size of 1024, incorporates 16 self-attention heads, and contains a total of 340 million parameters [31].

For the sequence-to-sequence approach, in which output tokens are sampled from a fixed vocabulary, fine-tuning a T5 large model [113] is suggested as an effective strategy. The T5 model, a pre-trained Transformer network introduced in [153], is used in the large version, which includes 24 Transformer

encoder blocks, 24 Transformer decoder blocks, an input and output hidden size of 1024, an inner layer hidden size of 4096, 16 self-attention heads, and a total of 737 million parameters.
The probability distribution across the predefined vocabulary V can be calculated using the formula below.

$$P_{\text{generate}}(V) = softmax(dec^T \cdot W_{\text{generate}}) \quad [23]$$

where *dec* refers to the result produced by the Transformer decoder. The matrix $W_{\text{generate}} \in \mathbb{R}^{\text{hidden size decoder} \times \text{vocabulary size}}$ is a tunable parameter that can be modified throughout the training process.
This particular T5 model is referred to as T5 generate.

In the dataset that is evaluated in this chapter, the tokens included in the corrected request are exclusively derived from the input sequence.
To capitalize on this feature, a pointer network model [155] has been developed utilizing the T5 large model. The model identifies the input token most likely to be replicated in the output sequence. This represents the third approach. The probability distribution across the tokens in the input sequence V can be determined using the following formula:

$$P_{\text{copy}}(V') = softmax(dec^T \cdot enc^T) \quad [23]$$

where *dec* refers to the result produced by the Transformer decoder. The Transformer encoder produces an output, $enc \in \mathbb{R}^{\text{input length} \times \text{embedding size}}$, which represents the encoded representation of the input sequence.
To utilize the knowledge embedded in the pre-trained model, the encoder is provided with the source input token that holds the highest probability, rather than the position of the source token that is copied. That means, that the copy mechanism is only used in the generation phase, in other phases, it is like a normal T5 model. If the output should have any separators that are not in the input, they must be added to the input, so that they can be copied as well. The modified T5 model is referred to as T5 copy.
In the sequence-to-sequence approaches, the utterances in the input are separated with an utterance separator and in the output there are separators for the correction and the pairs of reparandum and repair entity, for the pairs of reparandum and repair entity, and for the reparandums and repairs. Dataset examples and the usage of the separators can be found in Section 3.3.

The three previously mentioned approaches can be utilized to assess if an utterance functions as a correction to the preceding request command. In the sequence labeling approach, the absence of error correction is indicated when all output labels are C, whereas the presence of error correction is signaled by any deviation from this pattern. Sequence-to-sequence approaches detect an error correction when the source and target sequences, without the utterance separators, differ from each other; if they are identical, no error correction is applied. The T5 copy approach involves comparing the original source directly, instead of utilizing the version that includes the added separators (that needs to be copied).

Beyond these three approaches, sequence classification serves as a tool for error correction detection. To achieve this, fine-tuning the cased BERT large model is suggested. This model is composed of 24 Transformer encoder layers, features a hidden size of 1024, incorporates 16 self-attention heads, and contains 340 million parameters [31].

A large language model can function effectively as a few-shot model [12]. Eliminating the need for fine-tuning. The 175 billion parameter GPT-3 [12], specifically the text-davinci-003 variant, was utilized as the large language model. In order to perform the tasks of error correction detection and/or error correction, it is necessary to provide one example (for one-shot) or multiple examples (for few-shot) as a prefix within the prompt. Two examples were utilized in which the last utterance involved an error correction, alongside two examples where the last utterance did not include an error correction, specifically for tasks related to error correction detection. Section 3.3 provides an explanation of the distinction between a static prompt prefix and a dynamic prompt prefix. In addition to the few-shot approaches that utilize static and dynamic prompts, the 175 billion parameter GPT-3 model was also fine-tuned for the task as comparison.

The original implementation described in the Sentence-BERT paper [115] was utilized, while the official OpenAI Python API [102] was employed for GPT-3. The HuggingFace [170] PyTorch [105] BERT and T5 models that are utilized for the other models have been made publicly available online[1].

[1] `https://github.com/msc42/seq2seq-transformer` `https://github.com/msc42/seq-labeling-and-classification`

3.3. Dataset

The dataset in this section is also presented in [22] and the version without the GPT-3 approaches is also presented in [23]. Information about the copyright holders and the licenses of [23, 22] is in Section 3.1.

The adapted dataset created for the evaluation is based on the natural language annotations for the actions provided in the EPIC-KITCHENS-100 dataset [27, 28]. The EPIC-KITCHENS-100 dataset contains 100 hours of video footage documenting various activities performed in kitchen environments. Every annotation is labeled with an associated verb, its verb class, relevant entities, and the classes of the entities. An example of an annotation is “put pizza slice into container” where the verb is “put-into” and the entities are identified as "slice:pizza" and "container" that have the classes “pizza” and “container”, respectively. An annotation has up to six entities. The order of the entities and their respective classes correspond to the annotation. When a verb is accompanied by a preposition, the preposition is retained alongside the verb. Entities are organized in a hierarchical format, where the broadest category appears on the left and progressively more specific terms are arranged to the right, separated by colons.

The EPIC-KITCHENS-100 dataset includes 67 218 annotations in its training set and 9669 annotations in its validation set. A distinct test dataset is not provided. Certain annotations can occur repeatedly across various recordings, leading to a total of 15 968 distinct annotations in the training dataset and 3835 unique annotations in the validation dataset.

For the given dataset, only annotations featuring one or two entities are utilized. Annotations lacking entities are omitted, as at least one entity is necessary for a potential correction. Annotations involving more than two entities represent less than 1.15 % of the total annotations, leading to their exclusion to maintain dataset balance.

The original EPIC-KITCHENS-100 dataset has an uneven distribution across its verb categories. In order to improve the balance of the validation dataset, annotations from overly represented verb classes were eliminated. The goal was to create a more balanced dataset to assess the model’s ability to manage a wide range of verb categories effectively. The desired count of remaining annotations for a verb class, represented as r, was calculated by dividing the total annotations, represented as a, by 100. A minimum number for the remaining annotations was established, requiring at least 2 for single-entity annotations ($r = max(2, a/100)$) and at least 4 for two-entities annotations ($r = max(4, a/100)$). In some instances, the EPIC-KITCHENS-100 dataset does not provide a sufficient quantity of remaining annotations for a specific verb class, so only the available annotations were utilized. The after this rules constructed validation dataset is almost balanced, comprising 142 annotations featuring a single entity and 122 annotations containing two entities. During the process of collecting annotations for a verb class, an effort was made to achieve a balanced distribution of verbs within that category. The validation dataset, after being reduced, comprises 264 annotations in total.

The training and validation sets in the EPIC-KITCHENS-100 dataset exhibit numerous common characteristics. The validation dataset includes all 78 verb classes that are also present in the training dataset. Additionally, among the 372 first-level words within the entity hierarchies of the validation dataset, 346 are present in the training dataset as well. In order to enhance diversity between the training and validation datasets, a decision was made to implement certain reductions. To begin with, the 49 verb classes that appeared least frequently were excluded from the training dataset, which originally contained a total of 98 verb classes. Furthermore, all entities were removed from the training dataset if their first word within the entity hierarchy was already included in the validation dataset. This implies that annotations in the training dataset that includes entities such as “bowl:salad” were excluded if the validation dataset included annotations that have entities such as “bowl:washing:up”. Following the reduction process, the training dataset retains a total of 4822 annotations.

In order to train and evaluate the components responsible for error correction detection and error correction, corrections had to be incorporated into the annotations. The corrections for the training and validation datasets were generated synthetically. Three alternatives for entity substitution were available: substituting the first entity, substituting the second entity, or substituting both entities. All three options were randomly chosen (uniform distribution). In situations requiring corrections for both entities, the order of correction was randomly chosen (uniform distribution). The training dataset included eight distinct templates for the correction phrase, while the validation dataset featured six distinct templates. An entity is substituted with a different one from an annotation belonging to the same verb category and occupying the same position. An example for a request and correction pair where one entity is corrected is “Kindly grab the oregano” and “That’s the chili", and an example for a pair where two entities are corrected is "Would you mind placing the tin in the cupboard?" followed by correction "No, the olives go in the fridge".

For the test dataset, nine human data creators were recruited and granted autonomy to create their own corrections. The information provided to them specified which entities needed to be substituted with which ones, while granting them the flexibility to use synonyms for the substitution entities. The corrections were grouped into three categories: correcting an incorrect action performed by the robot, providing clarification, or addressing modifications resulting from a shift in the user's decision. The distribution of these categories was balanced across the corrections. The corrections were collected with a developed bot for the messenger Telegram. The bot operates as a server application utilizing an unofficial Python wrapper [37] designed for the official Telegram Bot API [150]. The database system PostgreSQL [139] is utilized. Figure 3.2 shows a screenshot of the bot.

In order to enhance the diversity of natural language within the training and validation datasets, 19 templates were incorporated prior to the narration for the training set, while 14 templates were added for the validation set. Furthermore, the article "the" was included before the entities to address its absence in the original EPIC-KITCHENS-100 dataset. For the test dataset, narrations from the validation dataset were utilized, and the same nine data creators responsible for generating the corrections were tasked with paraphrasing them. Figure 3.2 displays a screenshot illustrating the developed Telegram bot used to perform this action.

The test dataset presents a more significant challenge than the validation dataset due to its greater deviation from the training dataset. The nine data creators received instructions to employ a diverse range of natural language expressions.

By utilizing 4822 annotations from the reduced training dataset and applying multiple data augmentation techniques, a total of 52 357 request and correction pairs were created for the error correction training dataset. The validation dataset for error correction is composed of 264 pairs of requests and corrections, whereas the test dataset for error correction includes 960 such pairs.

In order to train and assess the detection of error corrections, it was necessary to gather examples in which the last utterance is not a correction. To

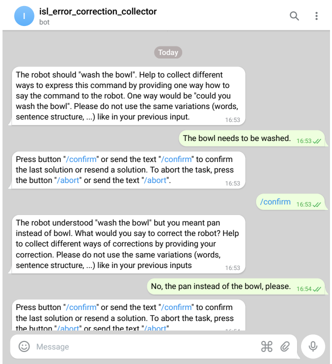


Figure 3.2.: Screenshot of the Telegram bot in the data collection mode

accomplish this, the last utterance is also a request from the available requests which were shuffled for the last utterance. As a result, the training dataset for error correction detection and error correction consists of 104 714 pairs, while the validation dataset includes 528 pairs, and the test dataset comprises 1920 pairs.

The target of the error correction datasets is to include the corrected requests alongside their corresponding reparandum and repair pairs.
The target of the error correction detection and error correction dataset is determined by whether the source includes a correction or whether it is a pair of requests. In the first scenario, involving error correction, the target aligns with the one specified in the datasets designed for error correction only. In the latter scenario, the target is to replicate both requests.
An additional dataset, referred to as the error correction detection dataset, is also available. The dataset utilizes the same sources as those used in the error correction detection and error correction dataset, but its target is represented as a binary value that specifies the presence or absence of a correction.

The datasets mentioned have been designed in diverse formats to accommodate the range of the approaches described in Section 3.2. In the sequence labeling approach, the defined labels were assigned to the source tokens, see Figure 3.3. Source and target pairs were constructed to facilitate the sequence-to-sequence approach involving generative token production, see Figure 3.4. In the sequence-to-sequence approach that generates output by copying tokens from the source, the order of copy operations was included.

Separator tokens needed in the target were added to the source, see Figure 3.5.

```
would C
it C
be C
possible C
to C
wash C
the C
table R1
?  C
| D
no D
the D
wok S1
instead D
of D
the D
table D
. D
```

adapted from [23] (© 2022 ACL, CC BY 4.0)

Figure 3.3.: Data example for the sequence labeling approach

```
source file:
Would it be possible to wash the table ?  | no
        the Wok instead of the table .

target file:
Would it be possible to wash the Wok ?  | table -> Wok
```

adapted from [23] (© 2022 ACL, CC BY 4.0)

Figure 3.4.: Data example for the sequence-to-sequence approach with a fixed vocabulary

Few-shot approaches utilizing the large language model GPT-3 [12] require carefully designed prompts to familiarize the model with the task of detecting

```
source file:  Would it be possible to wash the table ?  |
        no the Wok instead of the table .  - ->

target file:
Would it be possible to wash the Wok ?  | table -> Wok

copy target file:
3 4 5 6 7 8 9 16 17 11 12 13 10 26 27 16 17 28
```


Figure 3.5.: Data example for the sequence-to-sequence with the copy source token approach, considering the T5 prefix and the T5 tokenization

error correction and/or error correction. There are two approaches available for generating prompts. First, a static prompt generation approach is employed to create a single prompt applicable to all corrections that should be evaluated. Second, a dynamic prompt generation approach is utilized, producing tailored prompts specific to each individual correction that should be evaluated [91]. In the first approach, the error correction detection and error correction dataset were analyzed by identifying two request correction pairs and two request request pairs from the validation dataset that are most similar to all corresponding pairs in the training dataset. For the comparison Sentence-BERT [115] is utilized [90]. This idea to select meaningful instances as training data was already introduced in [33]. To identify the two most comparable pairs in which the last utterance involves an error correction, a process was conducted to compute the similarity between each request correction pair in the validation dataset and all request correction pairs in the training dataset. The similarity scores for each validation pair were then averaged to determine the closest matches. The two validation pairs with the highest averaged similarity scores were selected as the most similar ones. The same approach was implemented across all pairs in which the last utterance did not involve error correction. Figure 3.6 illustrates the static prompt designed for detecting error corrections and correcting errors within the specified dataset. The error correction dataset includes prompts that feature just the two most closely related pairs, with the last utterance serving as a correction. In the prompt generation approaches, the token “correction” was applied to pairs where the last utterance served as a correction, while “no correction" was assigned to cases where the last utterance did not function as a correction.

utterance 1: could you please empty the bowl contents into the saucepan
utterance 2: i meant into the cup .
output: could you please empty the bowl contents into the cup | saucepan -> cup

utterance 1: can you please empty the bowl contents into the saucepan .
utterance 2: could you be so kind and take the garlic press .
output: can you please empty the bowl contents into the saucepan . | could you be so kind and take the garlic press . |

utterance 1: Please empty the bowl contents into the saucepan .
utterance 2: It's into the can .
output: Please empty the bowl contents into the can . | saucepan -> can

utterance 1: can you please turn using the Tongs .
utterance 2: could you please empty the bowl contents into the saucepan .
output: can you please turn using the Tongs . | could you please empty the bowl contents into the saucepan . |

adapted from [22] (Reproduced with permission from Springer Nature)

Figure 3.6.: Static prompt of the error correction detection and error correction dataset

In the second approach for generating prompts, the process involved identifying, for each pair to be evaluated, the two most similar pairs from the training dataset where the last utterance represented a correction. Additionally, for datasets that included error correction detection, the two most similar pairs without a correction in the last utterance were also identified. Using these selected pairs, a prompt was constructed in a manner similar to the static prompt generation approach. The error correction detection dataset resembles the error correction detection and error correction dataset, but its output is a binary value indicating whether the most recent utterance constitutes a correction.

The GPT-3 model has the capability to be customized through fine-tuning. A training dataset for fine-tuning was created by randomly selecting 1000 pairs in which the last utterance involved a correction and another 1000 pairs where the final utterance did not include a correction. In the error correction dataset, the fine-tuning process incorporated the token "correction" for pairs where the last utterance represented a correction, and "okay" for cases where the last utterance did not involve a correction, enabling the fine-tuned GPT-3 model to function as a classification system that produces a single token as output.

3.4. Evaluation

The evaluation in this section is also presented in [22] and without the GPT-3 approaches in [23]. Information about the copyright holders and the licenses of [23, 22] is in Section 3.1.

This section starts by evaluating the approaches for detecting error correction, as outlined in Section 3.2. Subsequently, the error correction component approaches detailed in the same section will be evaluated. Thirdly, the effectiveness of a pipeline approach—where error correction detection and correction are handled as distinct components—will be evaluated in comparison to end-to-end approaches. The datasets outlined in Section 3.3 served as the basis for conducting all evaluations.

The sequence classification and labeling approaches underwent fine-tuning for a single epoch, utilizing the AdamW optimizer [93] with a learning rate of $2 \cdot 10^{-5}$, a batch size set to 32, and a maximum input length of 128.

The T5 generate and T5 copy models underwent fine-tuning for a single epoch using the Adam optimizer [73] with a learning rate of $2.5 \cdot 10^{-4}$, a batch size of 24, and a maximum input length set to 128. The parameters in the embedding layer and the first two encoder blocks were frozen.

The GPT-3 model employed with the static and the dynamic prompts was utilized in its original state without fine-tuning, whereas the fine-tuned GPT-3 variants underwent a four-epoch fine-tuning process conducted by OpenAI based on the provided training data (either from the error correction detection dataset, the error correction dataset, or the error correction detection and error correction dataset).

The GPT-3 approaches were evaluated disregarding case sensitivity and ignoring spaces. This action was taken due to utterances where certain data included spaces between words and punctuation marks or featured words beginning with uppercase letters in situations where this formatting was incorrect. Approaches other than GPT-3 replicate the grammatical errors, whereas GPT-3-based approaches corrected them. Overall, this is typically not considered a drawback and is unlikely to influence the evaluation.

The effectiveness of the error correction detection components is evaluated using various metrics. The number of correctly classified pairs is quantified using accuracy, and the corresponding results are presented in Table 3.1.

model	accuracy validation	test
classification	100 %	87.86 %
seq. labeling	100 %	88.49 %
T5 generate	98.67 %	84.01 %
T5 copy	71.78 %	77.45 %
GPT-3 (static prompt)	78.60 %	80.94 %
GPT-3 (dynamic prompt)	59.47 %	62.92 %
GPT-3 fine-tuned for classification	100 %	91.67 %

Table 3.1.: Accuracies of the error correction detection, the classification was trained on the error correction detection dataset, the other models were trained on the error correction detection and error correction dataset

Precision represents the proportion of correctly detected positive instances among all instances classified as positive, recall represents the proportion of correctly detected positive instances among all positive instances, and the F_1-score serves as the harmonic average of precision and recall. Precision, recall, and F_1-score were computed to evaluate the performance of the models, considering two scenarios: detecting corrections as positive examples (refer to Table 3.2) and detecting no corrections as positive examples (refer to Table 3.3).

Evaluation from both perspectives was employed to provide a more comprehensive understanding of how the various models performed. The approaches involving sequence classification, sequence labeling, and fine-tuned GPT-3 demonstrated the best performance on the validation dataset, each attaining

		detecting corrections		
dataset	model	precision	recall	F_1-score
validation	classification	100 %	100 %	100 %
validation	seq. labeling	100 %	100 %	100 %
validation	T5 generate	98.13 %	99.24 %	98.68 %
validation	T5 copy	63.92 %	100 %	77.99 %
validation	GPT-3 (static prompt)	70.35 %	98.86 %	82.20 %
validation	GPT-3 (dynamic prompt)	55.43 %	96.59 %	70.44 %
validation	GPT-3 fine-tuned for classification	100 %	100 %	100 %
test	classification	99.86 %	75.83 %	86.20 %
test	seq. labeling	99.87 %	77.08 %	87.01 %
test	T5 generate	96.58 %	70.52 %	81.52 %
test	T5 copy	73.63 %	85.52 %	79.13 %
test	GPT-3 (static prompt)	73.50 %	96.77 %	83.54 %
test	GPT-3 (dynamic prompt)	57.82 %	95.52 %	72.03 %
test	GPT-3 fine-tuned for classification	99.75 %	83.54 %	90.93 %

adapted from [23] (© 2022 ACL, CC BY 4.0) and
[22] (Reproduced with permission from Springer Nature)

Table 3.2.: Precision, recall, and F_1-score for detection error corrections

a perfect accuracy of 100 %. On the test dataset, the fine-tuned GPT-3 approach achieved the highest accuracy of 91.67 %, while the sequence labeling approach ranked second with an accuracy of 88.49 %, and the sequence classification approach came in third at 87.86 %. The three approaches achieving perfect validation accuracy demonstrated exceptional recall in detecting pairs where the last utterance lacked error correction in the test dataset, with rates of 99.79 % for the fine-tuned GPT-3 model and 99.90 % for the other two approaches. Additionally, they exhibited strong precision in detecting such cases, with the fine-tuned GPT-3 model achieving 85.84 %, the sequence labeling approach reaching 81.34 %, and the sequence classification approach attaining 80.52 %. This implies that, in the absence of any corrections, the component accurately recognizes this in the majority of pairs and minimizes unwanted corrections. This trait is advantageous because failing to detect a correction is preferable to mistakenly altering something that is already accurate. The detection of corrections yielded favorable results, achieving re-

dataset	model	detecting no corrections precision	recall	F_1-score
validation	classification	100 %	100 %	100 %
validation	seq. labeling	100 %	100 %	100 %
validation	T5 generate	99.23 %	98.11 %	98.67 %
validation	T5 copy	100 %	43.56 %	60.69 %
validation	GPT-3 (static prompt)	98.09 %	58.33 %	73.16 %
validation	GPT-3 (dynamic prompt)	86.76 %	22.35 %	35.54 %
validation	GPT-3 fine-tuned for classification	100 %	100 %	100 %
test	classification	80.52 %	99.90 %	89.17 %
test	seq. labeling	81.34 %	99.90 %	89.67 %
test	T5 generate	76.78 %	97.50 %	85.91 %
test	T5 copy	82.73 %	69.38 %	75.47 %
test	GPT-3 (static prompt)	95.27 %	65.10 %	77.35 %
test	GPT-3 (dynamic prompt)	87.13 %	30.31 %	44.98 %
test	GPT-3 fine-tuned for classification	85.84 %	99.79 %	92.29 %

adapted from [23] (© 2022 ACL, CC BY 4.0) and
[22] (Reproduced with permission from Springer Nature)

Table 3.3.: Precision, recall, and F_1-score for detection utterance pairs with no error corrections

call rates of 83.54 %, 77.08 %, and 75.83 %, alongside precision rates of 99.75 %, 99.87 %, and 99.86 % on the test dataset for the fine-tuned GPT-3 model, sequence labeling approach, and sequence classification approach, respectively. Detecting the appropriate correction can be especially difficult in certain instances of component failure, as illustrated by the example: "Kindly turn off the heat on the oven | Please turn off the water tap on the oven". The T5 generation approach demonstrated lower performance, with an accuracy of 98.67 % on the validation dataset and 84.01 % on the test dataset.

The validation dataset results for the error correction components are displayed in Table 3.4, while the corresponding results for the test dataset are provided in Table 3.5.

The accuracy metric was used to assess error correction. A correction is accurate when the predicted corrected utterance aligns with the reference correct utterance. The identification of reparandum and repair pairs is accu-

model	validation dataset correction	extraction	both
seq. labeling	96.21 %	94.70 %	94.70 %
E2E seq. labeling	96.59 %	95.08 %	95.08 %
T5 generate	92.80 %	95.83 %	91.29 %
E2E T5 generate	96.21 %	95.08 %	94.70 %
T5 copy	50.38 %	87.12 %	50.00 %
E2E T5 copy	70.83 %	92.42 %	68.94 %
GPT-3 (static prompt)	45.08 %	65.15 %	40.91 %
GPT-3 (dynamic prompt)	48.48 %	74.24 %	44.70 %
GPT-3 fine-tuned	91.29 %	57.58 %	55.68 %
E2E GPT-3 fine-tuned	59.47 %	37.88 %	34.47 %

adapted from [23] (© 2022 ACL, CC BY 4.0) and
[22] (Reproduced with permission from Springer Nature)

Table 3.4.: Accuracies of the error correction on the validation dataset, the end-to-end (E2E) models were trained on the error correction detection and error correction dataset and the other models were trained on the error correction dataset

rate when the predicted pairs match the reference pairs, disregarding both the order and instances where entities are mapped to themselves. Correction and extraction are considered correct if both elements are correct. This evaluation was done on the error correction dataset. The end-to-end sequence labeling approach, trained using the error correction detection and error correction dataset, demonstrated the highest overall accuracy of 95.08 % on the validation dataset. Specifically, it achieved an accuracy of 96.59 % for correction task and 95.08 % for extraction task. The T5 generation approach, which was trained on the error correction dataset, achieved the highest performance on the test dataset, recording an overall accuracy of 71.98 %, with a correction accuracy of 73.65 % and an extraction accuracy of 77.81 %. Overall, approaches developed using the error correction detection and error correction dataset demonstrated greater accuracy on the validation dataset, with the exception of the fine-tuned GPT-3 model. Meanwhile, those trained solely on the error correction dataset showed superior performance on the test dataset. Optimizing the T5 copy extraction process could involve tracking the sequence of copy operations, halting once the correction is complete, and leveraging this data to rebuild the reparandum and repair pairs. Nonetheless, this approach was ultimately set aside due to the notably poorer correction outcomes and the limited potential for further enhancements.

model	test dataset correction	extraction	both
seq. labeling	40.10 %	48.75 %	39.06 %
E2E seq. labeling	39.27 %	43.54 %	38.65 %
T5 generate	73.65 %	77.81 %	71.98 %
E2E T5 generate	37.40 %	38.75 %	36.25 %
T5 copy	50.52 %	62.19 %	47.92 %
E2E T5 copy	27.50 %	35.00 %	25.31 %
GPT-3 (static prompt)	53.02 %	70.62 %	49.58 %
GPT-3 (dynamic prompt)	45.63 %	73.33 %	42.71 %
GPT-3 fine-tuned	74.17 %	69.48 %	62.60 %
E2E GPT-3 fine-tuned	43.65 %	40.52 %	35.31 %

adapted from [23] (© 2022 ACL, CC BY 4.0) and
[22] (Reproduced with permission from Springer Nature)

Table 3.5.: Accuracies of the error correction on the test dataset, the end-to-end (E2E) models were trained on the error correction detection and error correction dataset and the other models were trained on the error correction dataset

The validation dataset results for the error correction detection and error correction components are displayed in Table 3.6, while the corresponding outcomes for the test dataset are provided in Table 3.7.

model(s)	validation dataset		
	correction	extraction	both
detection and seq. labeling	98.11 %	97.35 %	97.35 %
detection and E2E seq. labeling	98.30 %	97.54 %	97.54 %
E2E seq. labeling	98.30 %	97.54 %	97.54 %
detection and T5 generate	96.40 %	98.67 %	96.40 %
detection and E2E T5 generate	98.11 %	98.30 %	98.11 %
E2E T5 generate	97.54 %	97.16 %	96.40 %
detection and T5 copy	75.19 %	94.13 %	75.00 %
detection and E2E T5 copy	85.42 %	96.59 %	84.66 %
E2E T5 copy	69.70 %	78.03 %	56.25 %
detection and GPT-3 (static prompt)	72.54 %	82.58 %	70.45 %
detection and E2E GPT-3 (static prompt)	65.34 %	74.05 %	63.83 %
E2E GPT-3 (static prompt)	62.31 %	59.85 %	48.48 %
detection and GPT-3 (dynamic prompt)	74.24 %	87.12 %	72.35 %
detection and E2E GPT-3 (dynamic prompt)	69.32 %	81.25 %	66.86 %
E2E GPT-3 (dynamic prompt)	59.09 %	69.70 %	51.33 %
detection and GPT-3 fine-tuned	95.64 %	78.79 %	77.84 %
detection and E2E GPT-3 fine-tuned	96.78 %	78.79 %	78.41 %
E2E GPT-3 fine-tuned	96.59 %	29.73 %	29.36 %

Table 3.6.: Accuracies of the error correction detection and error correction validation dataset, the end-to-end (E2E) models were trained on the error correction detection and error correction dataset and the other models were trained on the error correction dataset, "detection and" states that the error correction detection was executed by the sequence labeling approach (best error correction detection model) and when it detects an error correction, the error correction is executed by the model after the "and"

model(s)	test dataset		
	correction	extraction	both
detection and seq. labeling	67.08 %	68.44 %	66.72 %
detection and E2E seq. labeling	69.58 %	71.77 %	69.27 %
E2E seq. labeling	69.58 %	71.77 %	69.27 %
detection and T5 generate	78.49 %	79.84 %	77.81 %
detection and E2E T5 generate	68.33 %	68.80 %	67.66 %
E2E T5 generate	68.07 %	68.49 %	66.88 %
detection and T5 copy	68.44 %	72.71 %	66.93 %
detection and E2E T5 copy	63.28 %	66.77 %	62.08 %
E2E T5 copy	55.00 %	58.91 %	47.40 %
detection and GPT-3 (static prompt)	71.76 %	77.86 %	70.52 %
detection and E2E GPT-3 (static prompt)	69.95 %	72.92 %	68.23 %
E2E GPT-3 (static prompt)	69.84 %	67.34 %	60.68 %
detection and GPT-3 (dynamic prompt)	69.17 %	79.06 %	68.02 %
detection and E2E GPT-3 (dynamic prompt)	69.58 %	75.62 %	67.29 %
E2E GPT-3 (dynamic prompt)	66.46 %	70.83 %	58.23 %
detection and GPT-3 fine-tuned	83.08 %	79.30 %	75.42 %
detection and E2E GPT-3 fine-tuned	82.89 %	78.51 %	76.12 %
E2E GPT-3 fine-tuned	81.88 %	57.86 %	56.41 %

Table 3.7.: Accuracies of the error correction detection and error correction test dataset, the end-to-end (E2E) models were trained on the error correction detection and error correction dataset and the other models were trained on the error correction dataset, “detection and” states that the error correction detection was executed by the sequence labeling approach (best error correction detection model) and when it detects an error correction, the error correction is executed by the model after the “and”

The accuracy metric applied was identical to the one utilized during the error correction evaluation. In the pipeline approaches, error correction detection was carried out using the sequence labeling approach, treating a pair as free of corrections when all assigned labels are marked as “C”. Once the error correction detection is done, the system proceeds to execute the correction process. The three approaches outlined in Section 3.2 were assessed in both their variants trained on the combined error correction detection and error correction dataset, as well as their versions trained solely on the error correction dataset. In the end-to-end approach, a unified component is responsible for managing both the detection and correction of errors within a single operation. The pipeline approach with the T5 generation approach for error correction demonstrated the best performance, achieving an accuracy of 96.40 % on the validation dataset and 77.81 % on the test dataset.

The evaluation results suggest that the test dataset is more challenging than the validation dataset. The nine data creators successfully incorporated a broader range of natural language variations compared to those presented in the validation dataset.

3.5. System

The Telegram bot used to create the test dataset was extended to use the trained models for a real-world system. In addition to entering text, audio can be spoken, it will be transcribed and the transcript is shown. After the second utterance, an error correction detection is run and if an error correction is detected, the error will be corrected and the corrected utterance is shown instead of the transcript of the correction. First, a from the Interactive System Labs provided ASR system based on [151, 59] was used, but it was replaced by Whisper [110] in the version that was adapted to used by the Lecture Translator framework [58]. The system is depicted in Figure 3.7.

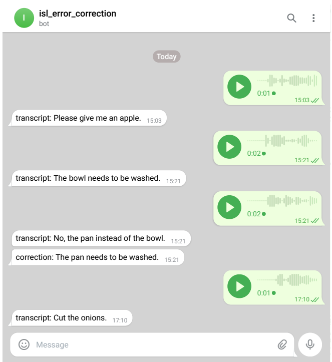


Figure 3.7.: Screenshot of the Telegram bot in the live system mode

3.6. Conclusion

The conclusion in this section is also presented in [22] and without the GPT-3 approaches in [23]. Information about the copyright holders and the licenses of [23, 22] is in Section 3.1.

A significant disparity in accuracy exists between the approaches. The T5 copy model, which was fine-tuned using the error correction detection and error correction dataset, demonstrates the poorest performance on the real-world error correction detection and correction test dataset, achieving an accuracy of only 47.40 %. The approach that utilizes a fine-tuned sequence labeling BERT model for detecting error corrections and a fine-tuned sequence-to-sequence T5 model for generating corrections achieves the highest accuracy of 77.81 % on the specified test dataset. This promising outcome on real-world data demonstrates that the approach is effectively grasping the concept of corrections, providing a useful feature for a dialog system without requiring domain-specific training for error correction. Moreover, outputting reparandum and repair pairs provides an opportunity to incorporate them into a continuous learning system, ensuring that repeated corrections are no longer required from users.

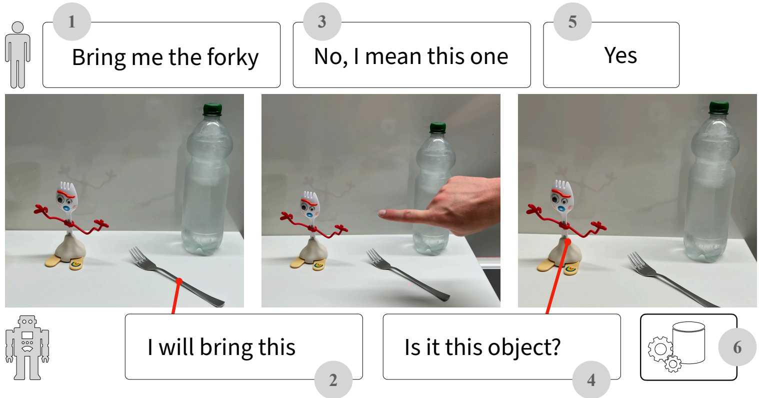
4. Multimodal Error Correction of Wrong Entities

In this chapter GPT-4o was used to improve the sentences, more information in Section 1.3.

4.1. Introduction

The error correction of wrong entities in natural language, as presented in the previous chapter, represents a crucial step toward developing a functional error correction system. However, the restriction to natural language only enables the correction of language, while other aspects, such as the entities with which the systems interact, remain unaddressed. Pointing gestures can be added to the system to enable the integration of such entities. A multimodal error correction approach utilizing natural language and pointing gestures is introduced in [18] (© 2023 IEEE) by me and my co-authors.

In [18] (© 2023 IEEE), the problem is addressed using a system composed of three components: pointing line generation, region proposal, and multimodal error correction. The pointing line generation was developed by my co-authors, the external component Faster R-CNN [116] is utilized as region proposal component, and the multimodal error correction along with the communication between the components were carried out by me. We created a live system designed to demonstrate it using a laser pointer robot. Figure 4.1 illustrates a multimodal correction. The user expresses a desire for the "forky" (a toy featured in an animated movie), but the system overgeneralizes and interprets this as a request to bring a fork. The user is able to make a correction using natural language and by pointing at the "forky". The system inquires after the correction whether it is now accurate, allowing the user to either accept it or make further adjustments.



adapted from [18] (© 2023 IEEE)

Figure 4.1.: Multimodal error correction task example: A command involving a potentially unknown entity is given by a user (1) and is misinterpreted by the system (2). In the showcase, the system is a small robot equipped with a laser pointer (see Section 4.5). Due to the error, the user carries out a multimodal correction using language and a pointing gesture (3). The system then requests confirmation (4), and if approval is granted (5), the resulting information is utilized to execute the action with the appropriate entity while incrementally learning the name of the new entity (6).

The pointing line generation process is tasked with detecting the presence of a human pointing gesture from the image or video input and calculating the pointing line if such a gesture is detected. Region proposal is employed to recognize the Regions of Interest (RoIs) of candidate entities, which are then intersected with the pointing line to generate pointing target candidates. If target candidates are available, they are sent to the multimodal error correction component. The component integrates information from vision and natural language to determine if a correction exists and, if so, clarifies the user's statement. The result of the multimodal error correction module could subsequently be utilized to initiate incremental learning of the new entity, based on its referring expression (word/phrase) and the visual grounding (RoI image). A very simple incremental learning approach was developed by my co-authors to demonstrate the usefulness of the multimodal error correction in a live system.

The following contributions are presented in [18] (© 2023 IEEE) and in this chapter: First, a system for multimodal error correction is introduced, com-

bining natural language and pointing gestures. Second, the performance of multimodal error correction was assessed by collecting a test dataset of challenging natural language utterances, which augment the pointing video dataset published in [17]. Third, extensive evaluation results are presented for the individual components and the overall approach. Furthermore, the usefulness of the system is demonstrated through a showcase application involving a small real-world laser pointer robot.
In addition, this chapter adds to the methodology of [18] (© 2023 IEEE) GPT-4o [101] as alternative backend for the multimodal error correction and evaluates and compares it to the VL-T5 [15] backend.

4.2. Methodology

The methodology except the GPT-4o backend presented in this section is also presented in [18] (© 2023 IEEE).

Overview

The objective is to carry out multimodal error correction. The input consists of the most recent user utterances u_1 and u_2 (with u_2 being the latest), along with the current camera image I or the camera video $V = (I_1, \ldots, I_N)$. The utterance u_2 can either serve as a correction to the initial utterance u_1, meaning $u_2 = c$, or it can represent a separate, independent command, meaning $u_2 = u'$. The pair $P_\mathrm{n} = (u_1, u')$ is defined as consisting of two independent utterances, while $P_\mathrm{c} = (u_1, c)$ is defined as a pair that includes a correction. Each image I depicts a collection of entities $E_I = \{e_1, \ldots, e_M\}$ and may either include a human pointing at the target entity e_p or lack such a gesture.

The multimodal error correction system $\mathcal{S}$ is intended to function in the following manner: If no correction is provided in u_2, the model is expected to detect this, i. e., $\mathcal{S}(P_\mathrm{n}, I) =$ *no correction*, and the utterance u' can be forwarded to a downstream component for Natural Language Understanding, e. g., to trigger suitable actions. Subsequently, u_1 is set equal to u_2, and the system pauses until the next utterance is received.

In the case of a correction, $\mathcal{S}(P_\mathrm{c}, I) = (e_p, l)$, where l represents the label utilized in u_1 to denote e_p, is intended. The resulting pair $(e_{\hat{p}}, \hat{l})$ can be provided to another component to facilitate incremental learning of the novel entity. For the scope of this chapter, with a focus on multimodal error correction, it is assumed that a pointing gesture is always present during a correction. Afterwards, u_2 is removed while u_1 is retained, allowing the next utterance to either provide another correction or stand independently.

The information flow for an input pair that involves a correction is depicted in Figure 4.2.

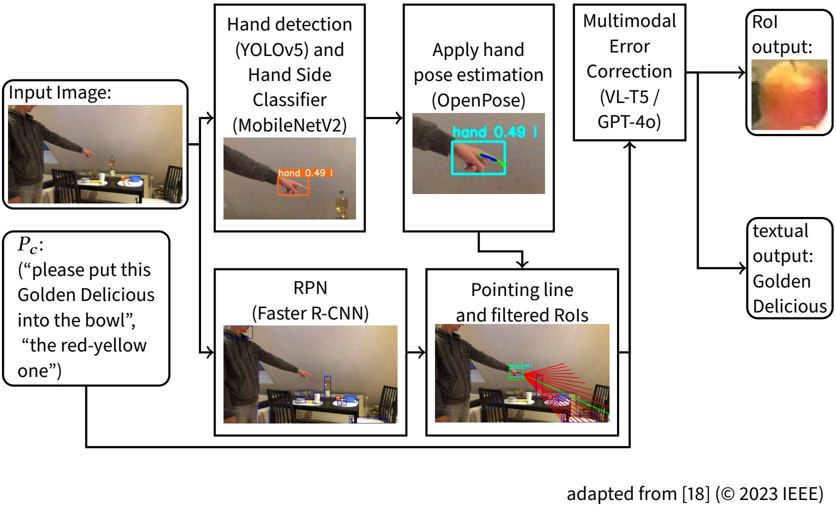


Figure 4.2.: Information flow of the multimodal error correction for an input pair that involves a correction. If no correction is present, there is no pointing line, and as a result, no regions of interest (RoIs) are provided to the multimodal error correction component. In such cases, the textual output is "no correction" and no RoI output is generated.

Multimodal Error Correction

For the multimodal error correction, two models are proposed to be used in the backend: either a VL-T5 [15] model (using a dataset described in Section 4.3 to fine-tune it) or GPT-4o [101] using a custom prompt. The following outlines the intended behavior of the model, as shaped by fine-tuning on that dataset or the custom prompt for the fine-tuned VL-T5 model or GPT-4o model, respectively. It is important to note that, because the models are neural networks, the system does not contain any hard-coded rules. VL-T5 was pre-trained in a multi-task setting, with each task being distinguished by a unique input prefix specific to that task. The pre-training process is also multimodal, meaning that the network is provided with both text and features from multiple images. In this case, the input to the VL-T5 model consists of 1) the prefix “correction visual grounding”, 2) the text of the user’s most recent two utterances, u_1 and u_2, and 3) the pointing candidate RoIs $R = \{r_1, ..., r_K\}$ with the bounding box coordinates and features associated with the respective bounding box. The input provided to the GPT-4o model consists of 1) the custom prompt, see Figure 4.3, 2) the text of the user’s most recent two utterances, u_1 and u_2, and 3) the pointing candidate RoIs $R = \{r_1, ..., r_K\}$ with a textual ID token of the RoI labeled as “IMAGE_NUMBER. image”, along with the image of the RoI defined by its bounding box coordinates cropped from the image. The system first detects whether a correction is present. If no correction is detected, the textual output t = “no correction”, indicating that the latest utterance does not serve as a correction to the preceding one. On the contrary, if a correction is detected, two outputs are produced: the ID token of the RoI $r_{\hat{p}}$ that is most likely being referenced (based on cues from language and vision) and the name $\hat{l}$ of that entity as textual output. As entity name should the name occurring in u_1 be used (i. e., not the corrected version) that resulted in the erroneous actions. When feasible within the system’s deployment scenario, an additional level of interactivity is incorporated to confirm the result before activating an incremental learning component. For example, this can be accomplished by the system displaying the cropped image of the output RoI to the user and requesting confirmation of its accuracy. Robots such as Pepper [103] and ARMAR-6 [4] which are equipped with displays, or a laser pointer to indicate an entity (refer to Section 4.5) can be used. If the user fails to confirm the result, the incorrect RoI $r_{\hat{p}}$ is eliminated from the collection of RoIs R on the pointing line, and the component is executed once more. The procedure can be repeated until the correct RoI r_p is identified, R becomes empty (in this case, the system might prompt the user to perform the pointing again), or the task is terminated by the user through natural language. In Section 4.4, up to two re-runs are evaluated.

Up to this point, the system presented addresses a single image I capturing a human pointing gesture. A more practical yet more complex scenario involves processing a video $V = (I_1, ..., I_N)$ instead; in this case, this approach is applied solely to the test dataset (refer to Section 4.3). The frames where a pointing gesture is detected are in I'. In this case, four approaches are compared to

> You are a system that gets two utterances. The first utterance is a request and the second utterance is either a correction to one entity of that request or another request. A correction comes along with a pointing gesture. On the pointing line are multiple entities. You get the images of the entities on the pointing line. If there is no pointing gesture detected, no images are given. If both utterances are requests, there should be no pointing gesture. However, as a result of wrong pointing gesture detection, images can be given.
> If there is no correction, output the string "no correction" without any other text. Otherwise, output the entity name of the first request that has been corrected and then after one space, the number of the image to which the entity belongs. If you have not received any images, output 0 as number but if there are some images, output the number of the image to which the entity belongs most likely, even if you think the entity does not belong to any of these images.
> First utterance: FIRST_UTTERANCE
> Second utterance: SECOND_UTTERANCE

Figure 4.3.: Prompt without images

determine the overall result for the VL-T5 model. First, the middle frame is selected from all frames in which a pointing gesture is detected (if the total number is even, the lower frame is chosen), and only the results from this frame are utilized. This approach is referred to as "middle frame". The second approach also utilizes the middle frame but applies a shortened sequence of the initial n frames in which a pointing gesture is detected. If fewer than n frames are included in the pointing sequence, all frames are utilized, i. e., $I'_{\lfloor \min(|I'|,n)/2 \rfloor}$ is employed. This approach, referred to as the "middle frame of the max. first n frames" is designed to simulate an online system, which is considered preferable since it is unnatural to complete the pointing process before receiving a system response. The last two approaches calculate the average of the individual results obtained from multiple frames. It is important to note that for each frame I'_i with $i \in \{1, \ldots, N'\}$, the model generates a textual output t_i (either "no correction" or the entity name $\hat{l}_i$) and, in cases where a correction is detected, an RoI $r_{\hat{p},i}$ is provided. The third approach utilizes all frames in which a pointing gesture is detected in the pointing video V. The overall textual output is determined based on a simple majority vote across all t_i. To average the output RoIs across all frames, the RoIs undergo a clustering process. For each RoI, a check is performed to determine whether there is an existing cluster with an Intersection over Union (IoU) greater than or equal to a defined threshold, and if such a cluster exists, the RoI is added to it. A new cluster is formed if no such a cluster exists. Next, the largest cluster is chosen, and all RoIs within it are averaged to produce an RoI bounding box, which serves as the system's output. This approach is referred to as "averaged frames." The fourth approach resembles the third approach, but, like approach 2, utilizes a reduced pointing sequence consisting of the first n frames in which a pointing gesture is detected to simulate an online system, i. e., $i \in \{1, ..., \min(n, N')\}$. This approach is referred to as "averaged frames of the max. first n frames." Averaging the RoIs is only meaningful for a static camera. For the GPT-4o backend only the "middle frame" of the max. first n frames is used. The averaged frames approaches are not optimal for a resource intensive model like GPT-4o and to use the max. first n frames is more realistic for a live system, because the users do not want to wait until their pointing gesture is finished but they already expect a correction during their pointing. The evaluation results of the VL-T5 model presented in Section 4.4 show that all approaches have comparable results and therefore it is reasonable to limit the GPT-4o backend to one approach.

For the generation of the RoI feature vector, the Detectron2 [172] reimplementation [149] in PyTorch is employed, which utilizes the same model and weights as the Caffe VG Faster R-CNN provided in bottom-up-attention [3]. For the multimodal error correction with the VL-T5 backend, the PyTorch code provided with [15] is utilized. Instead of utilizing 36 RoIs each time, the RoIs provided by the pointing line generation and region proposal component are employed.

4.3. Dataset

The dataset presented in this section is also presented in [18] (© 2023 IEEE).

The dataset utilized for training and validating the multimodal error correction component is derived from the EPIC-KITCHENS-100 dataset [28] for textual data and the MSCOCO dataset (2017 data split) [87] for visual data.

An annotation is a textual description of the action in the video of the EPIC-KITCHENS-100 dataset, such as "put cup into cupboard," which is repurposed as a robot command. Metadata is also present, including the set of entities and their associated classes utilized in the annotations. Each image in the MSCOCO dataset is annotated with bounding boxes, and each bounding box is linked to a category label corresponding to the entity it contains. For the dataset, a mapping is first created between the classes in EPIC-KITCHENS-100 and the category labels of MSCOCO. A total of 29 MSCOCO categories and EPIC-KITCHENS-100 class pairs were identified. Subsequently, this mapping is utilized to create subsets of EPIC-KITCHENS-100 and MSCOCO, with elements that include at least one of the mapped classes or categories.

Each element included in the dataset must consist of: 1) a textual input, specifically either an utterance pair P_c that includes a correction or a pair P_n that does not include a correction, 3) a visual input, specifically a set R of RoIs (can be empty), 3) a textual target t, which corresponds to either the entity name l or the text “no correction”, and 4) in the event of a correction, a pointing target RoI output r_p.

For constructing the textual input data (i. e., P_c, P_n, t) for the model, the process begins with the annotations of the EPIC-KITCHENS-100 subset. To enable the system to be trained to recognize unknown entities, one of the following three strategies is applied (uniformly at random) to modify the textual description of the entity. Replacement of the noun with a brand name, adding a brand name before the noun, or leaving the noun without a change. The brand names lack any semantic link to the entity, yet they serve as a simple method for introducing uncommon words. The modified textual description is utilized as u_1. To construct a correction utterance c, 14 templates (see Appendix A.1) are defined for the training dataset (e. g., “this is a LABEL”), and with a probability of 50 %, one of four prefixes (“no,”, “look,”, “i meant”, and “please,”) is randomly selected with equal distribution. The validation dataset is created using three templates (see Appendix A.1) that are different from those used in the training dataset. Some templates of the training and of the validation dataset utilize an “attribute” placeholder, which is populated with an attribute value assigned to the bounding box of the reference entity in the image (added in a subsequent step). The Detectron2 model [172], utilizing the Caffe VG Faster R-CNN weights supplied in bottom-up-attention [3], is responsible for classifying these attributes. The dataset includes 400 attributes, such as "black" and "round." The entity’s textual representation serves as the textual target $t = l$ for the model. For each annotation in the EPIC-KITCHENS-100 dataset, multiple correction pairs P_c are created by combining the generated u_1 with five and one randomly selected correction templates for the training dataset and validation dataset, respectively. In addition, the u_1 is augmented with different patterns to achieve greater natural language variety, as all annotations begin with a verb. Different patterns are used for the training and validation datasets. Furthermore, since pairs without error correction are also required, all of the generated u_1 are shuffled to create pairs P_n consisting of independent u_1 and u'.

After the described generation of the textual data, the textual and visual data are combined. Up to 10 bounding boxes from each image in the MSCOCO dataset are utilized, with each bounding box serving as the basis for a separate dataset element. For each bounding box, a P_c is selected from the generated utterance pairs in which the entity class associated with the utterance pair corresponds to the category of the bounding box. The selected P_c is then removed from the set of generated utterances; however, if the set becomes empty, it is reset to its initial state. Since the pointing component could identify any number of RoIs, including zero RoIs (i. e., the pointing gesture requires improvement), it is necessary to simulate this for the dataset used to train the multimodal error correction. With a probability of 20 %, P_c is combined with zero RoIs, while in the remaining 80 %, 1 to 35 additional RoIs (generated by the RPN) are randomly and equally distributed. From that point forward, the reference bounding box (which initiated the dataset element) is designated as r_p and incorporated into R in the same manner as any other RoI. If more than one RoI is added, its input sequence position is chosen randomly. Thus, a complete dataset element of $S_c = (P_c, R, l, r_p)$ has now been obtained, where l is defined by u_1. Each time a dataset element with a correction is added to the dataset, an additional element without a correction is included as well, ensuring the system is trained to detect whether a correction is present or not. To enhance robustness against false positives in pointing recognition, with a probability of 50 % the same RoIs are added to the dataset element that has P_n as those used for S_c. This implies that a dataset element $S_n = (P_n, R, \text{"no correction"})$ is added, where R is either identical to the value associated with the corresponding S_c or R is the empty set.

In the end, the training dataset contains 203 342 elements with a correction and the same amount without a correction. The validation set consists of 7232 elements each.

To evaluate the component under real-world conditions, the 182 videos featuring pointing gestures and the 40 videos without pointing gestures from [17] are utilized. The following labels were added to the dataset: For each video featuring a pointing gesture except for one, four individuals annotated two P_c pairs along with the reference bounding box r_p. For the exception video, only one pair P_c was annotated. The videos that did not include pointing gestures were annotated with two P_n pairs by the same group of four individuals.

4.4. Evaluation

The evaluation except the evaluation of the GPT-4o backend, the evaluation of only elements where the reference RoI is in the collection of proposed RoIs, additional examples, and statistics about the number of RoIs of the elements of the test dataset presented in this section is also presented in [18] (© 2023 IEEE).

In this section, the pointing line generation and region proposal component described in Section 4.2 will be evaluated first. Afterwards, results for multimodal error correction are presented, building on the outcomes of the previous step.

Pointing Line Generation and Region Proposal

To assess the performance of the pointing line generation and region proposal component, the evaluation involved verifying whether the annotated reference RoI r_p was included in the set of proposed RoIs $R = \{r_1, ..., r_L\}$. To achieve this, the Intersection over Union (IoU) thresholds $\tau \in \{0.25, 0.5, 0.75\}$ are utilized, and r_p is considered part of R if an i exists such that $IoU(r_i, r_p) > \tau$, meaning the IoU between one of the proposed RoIs and the reference exceeds the threshold τ. For approaches that utilize multiple frames, it is considered included if the aforementioned criterion is met in more than 50 % of the frames. The results are shown in Table 4.1. The best method involves utilizing solely the middle frame from the first 30 pointing frames in which a pointing gesture is detected, resulting in the inclusion of 62.26 % of the reference RoIs within the proposed RoIs for an IoU threshold of $\tau = 0.25$, 55.10 % for $\tau = 0.5$, and 42.42 % for $\tau = 0.75$.

approach	IoU 0.25	IoU 0.5	IoU 0.75
middle frame	60.06 %	54.55 %	40.22 %
middle frame of the max. first 30 frames	62.26 %	55.10 %	42.42 %
averaged frames	57.85 %	53.44 %	39.12 %
averaged frames of the max. first 30 frames	58.40 %	53.44 %	40.22 %

[18] (© 2023 IEEE)

Table 4.1.: Evaluation results of the pointing line generation and region proposal component, is the reference RoI in the set of proposed RoI, evaluated with the IoU thresholds of 0.25, 0.5, and 0.75

In 28 elements, no RoIs were recognized; in 29 elements, 10 or more were recognized; and in the remaining elements, between 1 and 9 RoIs were recognized. On average, 6.21 RoIs per elements were recognized. The full distribution is shown in Figure 4.4.

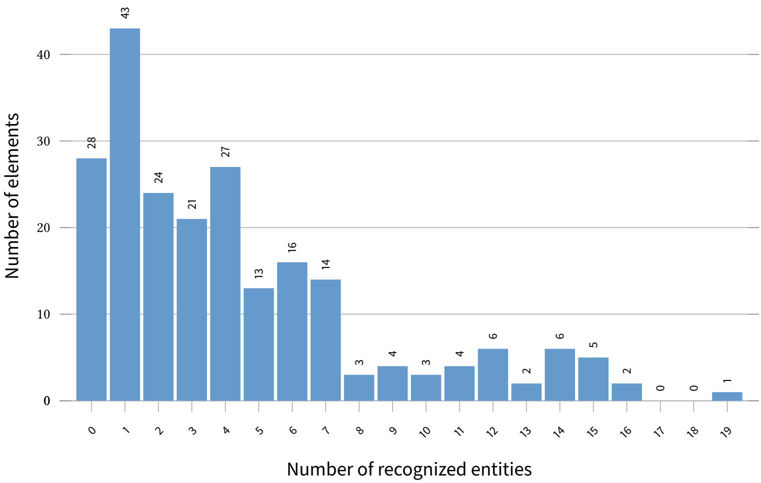


Figure 4.4.: Number of recognized entities distribution

Multimodal Error Correction

After providing a test element to the system, the textual output t is "no correction", or if a correction is detected, the output is $t = l$ along with a corresponding RoI $r_{\hat{p}}$. Given the annotated target label l and reference RoI r_p, the following evaluation metrics are calculated: First, the focus is solely on evaluating the textual output. To achieve this, t and $\hat{t}$ are compared in order to calculate the textual accuracy. Second, the aim is to evaluate the RoI proposals individually. To achieve this, the $IoU(r_p, r_{\hat{p}})$ is computed, and the RoI accuracy is reported as the percentage of elements with a correction exceeding a threshold τ of 0.25, 0.5, or 0.75, respectively. Third, the goal is to assess the overall system using the "complete output accuracy", which determines an element to be correctly solved if it is either detected accurately as "no correction" or both $t = \hat{t}$ and $IoU(r_p, r_{\hat{p}}) > \tau$. Fourth, the second and third evaluation goal is repeated for elements where the reference RoI is in the proposed RoIs or there is no correction and therefore no RoI needs to be outputted. This kind of evaluation should evaluate the multimodal error correction component without errors propagated from the region proposal and pointing line generation component.

For the clustering algorithm of the averaged frames approaches (see Section 4.2), the IoU nust be equal or greater than 0.5.

The multimodal error correction component with the GPT-4o backend that uses the "middle frame" of the first max. 30 frames has with 86.46 % the highest textual accuracy. For the VL-T5 backend, the following results were achieved: The approach that averages the maximum of the first 30 frames where a pointing gesture is detected achieves the highest textual accuracy of 81.04 %. The lowest accuracy, at 79.01 %, is only marginally worse (middle frame of the maximum first 30 frames where a pointing gesture is detected). The middle frame approach (79.68 %) and the averaged frame approach (80.81 %) demonstrate accuracies between the worst and best approach.

The assessment of the proposed RoIs is dependent upon the outcomes produced by the pointing line generation and region proposal component. An RoI can only be outputted if it is part of the RoIs proposed by the pointing line generation and region proposal component. Consequently, for the two middle frame approaches, the outcomes of the pointing line generation and region proposal component represent an upper limit. However, the results presented in Table 4.1 for the average frames approaches represent merely an approximation of an upper bound, as the evaluation of the pointing/RPN outcomes is more stringent compared to the criterion applied to average the results of these approaches. When assessing the pointing/RPN outcome, the reference RoI is required to achieve sufficient IoU with at least one of the proposed RoIs in over 50 % of the frames. In contrast, the average frame approaches rely on the cluster generated by the most frames of the output RoI across all frames and assess whether the IoU of this cluster is sufficient in relation to the reference RoI. Since the cluster generated by the most frames could be formed by fewer than half of the frames, this criterion is considered less stringent. Pointing/RPN evaluation is conducted using the 50 % criterion, to make it possible (for a perfect multimodal error correction system) to construct the largest cluster that satisfies the IoU threshold.

If the component failed to locate the reference RoI on the initial attempt, it was allowed up to two additional attempts.

In Table 4.2, 4.3, and 4.4 the evaluation results of the RoI accuracies on the test dataset elements requiring correction are presented for an IoU threshold of 0.25, 0.5, and 0.75, respectively.

		IoU 0.25		
backend	approach	1 try	≤ 2 tries	≤ 3 tries
VL-T5	middle frame	22.87 %	34.16 %	41.60 %
VL-T5	middle frame of the max. first 30 frames	22.59 %	33.06 %	39.12 %
VL-T5	averaged frames	22.31 %	33.61 %	42.42 %
VL-T5	averaged frames of the max. first 30 frames	22.59 %	34.99 %	44.35 %
GPT-4o	middle frame of the max. first 30 frames	50.14 %	57.30 %	59.23 %

adapted from [18] (© 2023 IEEE)

Table 4.2.: RoI accuracy on the test correction elements (IoU threshold τ of 0.25)

The GPT-4o backend has the best performance in RoI accuracy on the test data with only elements including a correction with the following accuracies: 50.14 % ($\tau = 0.25$), 41.87 % ($\tau = 0.5$), and 33.88 % ($\tau = 0.75$) for one attempt and 59.23 % ($\tau = 0.25$), 52.34 % ($\tau = 0.5$), and 40.50 % ($\tau = 0.75$) for up to three attempts. For the VL-T5 backend, there is no best approach, depending on the used IoU and the number of attempts, different approaches have the best

backend	approach	1 try	IoU 0.5 ≤ 2 tries	≤ 3 tries
VL-T5	middle frame	19.01 %	30.30 %	36.91 %
VL-T5	middle frame of the max. first 30 frames	18.73 %	28.10 %	33.33 %
VL-T5	averaged frames	18.46 %	29.48 %	36.64 %
VL-T5	averaged frames of the max. first 30 frames	18.18 %	30.03 %	37.47 %
GPT-4o	middle frame of the max. first 30 frames	41.87 %	50.14 %	52.34 %

adapted from [18] (© 2023 IEEE)

Table 4.3.: RoI accuracy on the test correction elements (IoU threshold τ of 0.5)

backend	approach	1 try	IoU 0.75 ≤ 2 tries	≤ 3 tries
VL-T5	middle frame	13.50 %	22.87 %	26.72 %
VL-T5	middle frame of the max. first 30 frames	15.43 %	23.14 %	25.90 %
VL-T5	averaged frames	14.33 %	23.14 %	28.10 %
VL-T5	averaged frames of the max. first 30 frames	14.60 %	24.52 %	29.20 %
GPT-4o	middle frame of the max. first 30 frames	33.88 %	38.57 %	40.50 %

adapted from [18] (© 2023 IEEE)

Table 4.4.: RoI accuracy on the test correction elements (IoU threshold τ of 0.75)

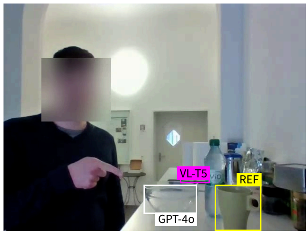
accuracy. The most practical the middle frame of the max. first 30 frames has the following accuracies: 22.59 % ($\tau = 0.25$), 18.73 % ($\tau = 0.5$), and 15.43 % ($\tau = 0.75$) for one attempt and 39.12 % ($\tau = 0.25$), 33.33 % ($\tau = 0.5$), and 25.90 % ($\tau = 0.75$) for up to three attempts. The VL-T5 backend profits more from multiple attempts than the GPT-4o backend.

In Figure 4.5 and Figure 4.6 examples from the test dataset are illustrated. A correct example of the test dataset is also illustrated in Figure 4.2.

$u_1 =$ "cut that Imperators into very fine slices"
$u_2 =$ "they are at this place here"
$l =$ "Imperators"
$l_{VL-T5} = l = l_{GPT-4o}$
$r_p \in R$
$r_{VL-T5} = r_p = r_{GPT-4o}$

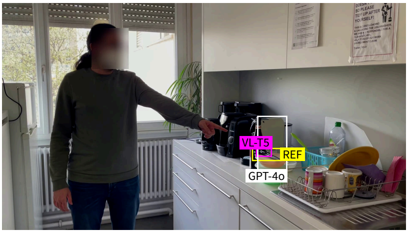
$u_1 =$ "can you shut the MacBook down?"
$u_2 =$ "it's here"
$l =$ "MacBook"
$l_{VL-T5} = l = l_{GPT-4o}$
$r_p \in R$
$r_{VL-T5} = r_p \neq r_{GPT-4o}$

$u_1 =$ "cut a gala to the pieces"
$u_2 =$ "you have to cut this apple"
$l =$ "gala"
$l_{VL-T5} =$ "no correction" $\neq l = l_{GPT-4o}$
$r_p \in R$
$r_{VL-T5} \neq r_p = r_{GPT-4o}$



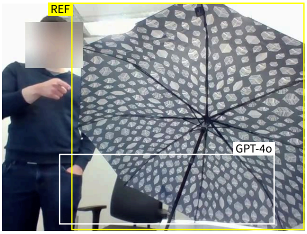
$u_1 =$ "robot please wash the chalice in the sink"
$u_2 =$ "please learn that this is a chalice"
$l =$ "chalice"
$l_{VL-T5} =$ "sink" $\neq l = l_{GPT-4o}$
$r_p \in R$
$r_{VL-T5} \neq r_p \neq r_{GPT-4o}$

Figure 4.5.: Multi error correction examples part 1, showing the predictions of the VL-T5 and GPT-4o backend, if VL-T5 and GPT-4o select the same RoI, "PRED" instead of "VL-T5" and "GPT-4o" is used



$u_1 =$ "can you please move the clippers to the shelf?"
$u_2 =$ "the one that I'm pointing"
$l =$ "clippers"
$l_{VL-T5} = l = l_{GPT-4o}$
$r_p \notin R$
$r_{VL-T5} \neq r_p \neq r_{GPT-4o}$

$u_1 =$ "would you please hand me the blade?"
$u_2 =$ "here is the blade"
$l =$ "blade"
$l_{VL-T5} = l \neq$ "no correction" $= l_{GPT-4o}$
$r_p \notin R$
$r_{VL-T5} \neq r_p \neq r_{GPT-4o}$



$u_1 =$ "Bring the sunshade to me."
$u_2 =$ "I'm pointing to that item"
$l =$ "sunshade"
$l_{VL-T5} =$ "no correction" $\neq l = l_{GPT-4o}$
$r_p \notin R$
$r_{VL-T5} \neq r_p \neq r_{GPT-4o}$

$u_1 =$ "Can you pass me cherries and a napkin, please"
$u_2 =$ "no, this is the bowl of cherries"
$l =$ "cherries"
$l_{VL-T5} =$ "bowl of cherries" $\neq l \neq$ "bowl" $= l_{GPT-4o}$
$r_p \in R$
$r_{VL-T5} = r_p = r_{GPT-4o}$

Figure 4.6.: Multi error correction examples part 2, showing the predictions of the VL-T5 and GPT-4o backend, if VL-T5 and GPT-4o select the same RoI, "PRED" instead of "VL-T5" and "GPT-4o" is used

The evaluation results for the complete output accuracy on the full test dataset are presented in Table 4.5 for an IoU threshold of 0.25, in Table 4.6 for an IoU threshold of 0.5, and in Table 4.7 for an IoU threshold of 0.75.

backend	approach	1 try	IoU 0.25 ≤ 2 tries	≤ 3 tries
VL-T5	middle frame	34.76 %	42.44 %	47.63 %
VL-T5	middle frame of the max. first 30 frames	34.31 %	41.08 %	45.37 %
VL-T5	averaged frames	34.99 %	42.89 %	49.21 %
VL-T5	averaged frames of the max. first 30 frames	34.76 %	43.12 %	49.44 %
GPT-4o	middle frame of the max. first 30 frames	53.27 %	58.47 %	60.05 %

adapted from [18] (© 2023 IEEE)

Table 4.5.: Complete output accuracy on the full test dataset (IoU threshold τ of 0.25)

backend	approach	1 try	IoU 0.5 ≤ 2 tries	≤ 3 tries
VL-T5	middle frame	31.83 %	39.50 %	44.24 %
VL-T5	middle frame of the max. first 30 frames	31.38 %	37.70 %	41.08 %
VL-T5	averaged frames	32.05 %	39.73 %	44.92 %
VL-T5	averaged frames of the max. first 30 frames	31.38 %	39.28 %	44.47 %
GPT-4o	middle frame of the max. first 30 frames	47.18 %	53.27 %	55.08 %

adapted from [18] (© 2023 IEEE)

Table 4.6.: Complete output accuracy on the full test dataset (IoU threshold τ of 0.5)

The numbers surpass the results for RoI accuracy on the test dataset limited to elements involving a correction, as the component consistently identified the “no correction” label accurately in all cases for the VL-T5 backend and in 92.50 % of the cases for the GPT-4o backend and the textual accuracy is in general high. This property is advantageous, as it is preferable to handle corrections carefully to avoid altering something that is already satisfactory. The error correction component is intended solely to enhance the overall system and should not diminish its performance. That property leads to an accuracy of the GPT-4o backend of 53.27 % ($\tau = 0.25$), 47.18 % ($\tau = 0.5$), and 41.53 % ($\tau = 0.75$) for the first attempt and increases to 60.05 % ($\tau = 0.25$), 55.08 % ($\tau = 0.5$), and 46.73 % ($\tau = 0.75$) by consideration of up to three

backend	approach	1 try	IoU 0.75 ≤ 2 tries	≤ 3 tries
VL-T5	middle frame	27.77 %	34.09 %	36.79 %
VL-T5	middle frame of the max. first 30 frames	28.67 %	33.86 %	35.21 %
VL-T5	averaged frames	28.89 %	34.99 %	38.60 %
VL-T5	averaged frames of the max. first 30 frames	28.67 %	35.44 %	38.37 %
GPT-4o	middle frame of the max. first 30 frames	41.53 %	45.15 %	46.73 %

adapted from [18] (© 2023 IEEE)

Table 4.7.: Complete output accuracy on the full test dataset (IoU threshold τ of 0.75)

attempts. The averaged frames approach was the approach with the best performance for the VL-T5 backend with an accuracy of 34.99 % ($\tau = 0.25$), 32.05 % ($\tau = 0.5$), and 28.89 % ($\tau = 0.75$) for the first attempt and increases to 49.21 % ($\tau = 0.25$), 44.92 % ($\tau = 0.5$), and 38.60 % ($\tau = 0.75$) by consideration of up to three attempts. However, the resource-saving and practical approach, the middle frame of the max first 30 frames approach, is only slightly worse. It has an accuracy of 34.31 % ($\tau = 0.25$), 31.38 % ($\tau = 0.5$), and 28.67 % ($\tau = 0.75$) for the first attempt, and for up to three attempts the accuracy increases to 45.37 % ($\tau = 0.25$), 41.08 % ($\tau = 0.5$), and 35.21 % ($\tau = 0.75$).

The results presented have the errors made by the region proposal and line generation component included because they are propagated. To only evaluate the accuracies of the multimodal error correction component, the elements of the dataset for that the reference RoI could not be predicted because it is not in the proposed RoIs are removed and only the remaining elements are evaluated. The results for the remaining elements including a correction are in Table 4.8, Table 4.9, and Table 4.10 for the IoU thresholds of 0.25, 0.5, and 0.75, respectively.

The number of successfully predicted RoIs is the same as in the previous results but the number of elements was decreased through the removement of the elements where the reference RoI is not the proposed RoIs. The number of removed elements depends on the IoU threshold and the approach. The different approaches are not directly comparable because they could have a different number of elements since the approach changes in which elements

backend	approach	1 try	IoU 0.25 ≤ 2 tries	≤ 3 tries
VL-T5	middle frame	38.07 %	56.88 %	69.27 %
VL-T5	middle frame of the max. first 30 frames	36.28 %	53.10 %	62.83 %
VL-T5	averaged frames	37.14 %	56.19 %	68.57 %
VL-T5	averaged frames of the max. first 30 frames	36.79 %	57.08 %	70.28 %
GPT-4o	middle frame of the max. first 30 frames	80.53 %	93.69 %	97.73 %

Table 4.8.: RoI accuracy on the test correction elements in which the reference RoI is in the recognized RoIs (IoU threshold τ of 0.25)

backend	approach	1 try	IoU 0.5 ≤ 2 tries	≤ 3 tries
VL-T5	middle frame	34.85 %	55.56 %	67.68 %
VL-T5	middle frame of the max. first 30 frames	34.00 %	51.00 %	60.50 %
VL-T5	averaged frames	34.54 %	54.64 %	65.98 %
VL-T5	averaged frames of the max. first 30 frames	32.99 %	54.64 %	67.01 %
GPT-4o	middle frame of the max. first 30 frames	76.00 %	92.86 %	97.44 %

Table 4.9.: RoI accuracy on the test correction elements in which the reference RoI is in the recognized RoIs (IoU threshold τ of 0.5)

the reference RoI is in the proposed RoIs and this ability is not evaluated in these tables. The GPT-4o backend has a better RoI accuracy than the VL-T5 backend (only the VL-T5 backend with the middle frame of the max. first 30 frames approach is comparable because they use the same approach) on the test dataset with only elements including a correction and where the reference RoI is in the proposed RoIs with the following accuracies: 80.53 % ($\tau = 0.25$), 76.00 % ($\tau = 0.5$), and 79.87 % ($\tau = 0.75$) for one attempt and 97.73 % ($\tau = 0.25$), 97.44 % ($\tau = 0.5$), and 97.35 % ($\tau = 0.75$) for up to three attempts. A higher IoU threshold can have a higher accuracy because the IoU threshold determines what elements have the reference IoU in the proposed IoUs and therefore the different IoU thresholds have different number of total elements.

backend	approach	1 try	IoU 0.75 ≤ 2 tries	≤ 3 tries
VL-T5	middle frame	33.56 %	56.85 %	66.44 %
VL-T5	middle frame of the max. first 30 frames	36.36 %	54.55 %	61.04 %
VL-T5	averaged frames	35.21 %	54.93 %	64.08 %
VL-T5	averaged frames of the max. first 30 frames	32.19 %	55.48 %	66.44 %
GPT-4o	middle frame of the max. first 30 frames	79.87 %	92.11 %	97.35 %

Table 4.10.: RoI accuracy on the test correction elements in which the reference RoI is in the recognized RoIs (IoU threshold τ of 0.75)

The most practical, the middle frame of the max. first 30 frames, approach has the following accuracies for the VL-T5 backend: 36.28 % ($\tau = 0.25$), 34.00 % ($\tau = 0.5$), and 36.36 % ($\tau = 0.75$) for one attempt and 62.83 % ($\tau = 0.25$), 60.50 % ($\tau = 0.5$), and 61.04 % ($\tau = 0.75$) for up to three attempts.

The accuracies for all elements on the complete output (textual and RoI) that can be solved because the reference RoI is in the proposed RoIs or it is a no correction element which needs no proposed RoI are depicted in Table 4.11, Table 4.12, and Table 4.13 for the IoU thresholds of 0.25, 0.5, and 0.75, respectively.

backend	approach	1 try	IoU 0.25 ≤ 2 tries	≤ 3 tries
VL-T5	middle frame	51.68 %	63.09 %	70.81 %
VL-T5	middle frame of the max. first 30 frames	49.67 %	59.48 %	65.69 %
VL-T5	averaged frames	52.41 %	64.48 %	72.07 %
VL-T5	averaged frames of the max. first 30 frames	51.37 %	63.36 %	71.23 %
GPT-4o	middle frame of the max. first 30 frames	77.12 %	84.64 %	86.93 %

Table 4.11.: Complete output accuracy on the full test dataset in which the reference RoI is in the recognized RoIs (IoU threshold τ of 0.25)

backend	approach	1 try	IoU 0.5 ≤ 2 tries	≤ 3 tries
VL-T5	middle frame	50.72 %	62.95 %	70.50 %
VL-T5	middle frame of the max. first 30 frames	49.64 %	59.64 %	65.00 %
VL-T5	averaged frames	51.82 %	64.23 %	71.17 %
VL-T5	averaged frames of the max. first 30 frames	50.00 %	62.41 %	69.71 %
GPT-4o	middle frame of the max. first 30 frames	74.64 %	84.29 %	87.14 %

Table 4.12.: Complete output accuracy on the full test dataset in which the reference RoI is in the recognized RoIs (IoU threshold τ of 0.5)

backend	approach	1 try	IoU 0.75 ≤ 2 tries	≤ 3 tries
VL-T5	middle frame	54.42 %	66.81 %	72.12 %
VL-T5	middle frame of the max. first 30 frames	54.27 %	64.10 %	66.67 %
VL-T5	averaged frames	56.76 %	67.57 %	72.52 %
VL-T5	averaged frames of the max. first 30 frames	53.54 %	65.93 %	71.24 %
GPT-4o	middle frame of the max. first 30 frames	78.63 %	85.47 %	88.46 %

Table 4.13.: Complete output accuracy on the full test dataset in which the reference RoI is in the recognized RoIs (IoU threshold τ of 0.75)

The GPT-4o backend has the following complete accuracy on the test data with only elements where the reference RoI is in the proposed RoIs or it is a element including no correction with the following accuracies: 77.12 % ($\tau = 0.25$), 74.64 % ($\tau = 0.5$), and 78.63 % ($\tau = 0.75$) for one attempt and 86.93 % ($\tau = 0.25$), 87.14 % ($\tau = 0.5$), and 88.46 % ($\tau = 0.75$) for up to three attempts.

The resource-saving and practical approach of the VL-T5 backend, the middle frame of the max first 30 frames approach has the following accuracies: It has an accuracy of 49.67 % ($\tau = 0.25$), 49.64 % ($\tau = 0.5$), and 54.27 % ($\tau = 0.75$) for the first attempt, and for up to three attempts the accuracy increases to 65.69 % ($\tau = 0.25$), 65.00 % ($\tau = 0.5$), and 66.67 % ($\tau = 0.75$).

The results from the GPT-4o model come from the API offered by OpenAI. First, Microsoft Azure was used but the results were worse than it could be, because in Microsoft Azure content is filtered related to sexuality, violence, hate, and self-harm. A total of 33 out of the 443 elements were excluded due to suspected sexual content in the first run. Nevertheless, it is not deterministic, as on the first attempt seven elements were accepted but six of them were filtered in the second attempt and one was even filtered on the third attempt. A filtered request includes the utterances “Please open rucksack that I can see its content.” and “This is a backpack.”, with the corresponding images shown in Figure 4.7. Because of this filtering, the API offered by OpenAI, which all elements accepted, was used.

4.5. System

The system presented in this section is also presented in [18] (© 2023 IEEE).

To show the developed components as real-world system, the VL-T5 model was fine-tuned to not only be responsible for the multimodal error correction but in addition, also for the natural language understanding. There is only one action “pointing” with one parameter: the entity that is pointed at.

When the VL-T5 model outputs such a pointing entity with a textual entity name, my co-authors developed a component where a pre-trained YOLO network [114, 68] is used for visual grounding if the name of the entity is a known class, otherwise, a simple feature clustering approach is used that learns incrementally new classes to which it can visually ground through the corrections that the multimodal error correction component provides. They use an open-source laser pointer robot [67] to point to the entity that is the result of the visual grounding. That represents the action the robot can do and a text-to-speech (TTS) system that asks for confirmation.

When the VL-T5 model outputs a correction, the bounding box is directly sent to the component developed by my co-authors that point to the entity in the bounding box and is used in the simple feature clustering approach.

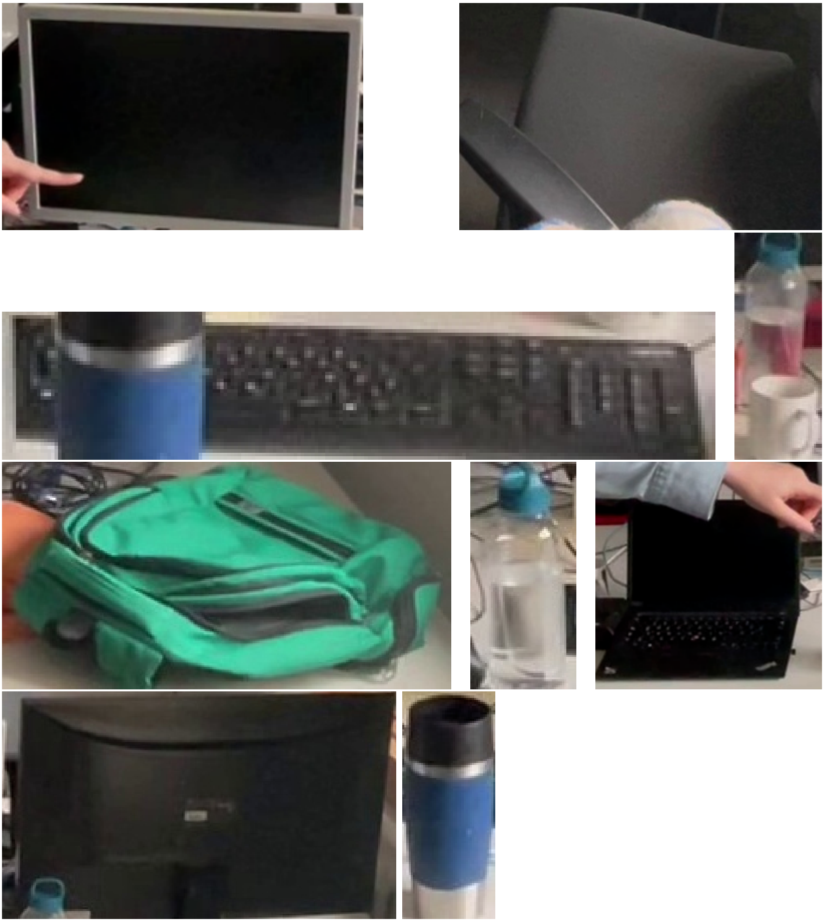


Figure 4.7.: Images affected of Azure's content filtering (suspected sexual content), are arranged in order from left to right and top to bottom.

4.6. Conclusion

The conclusions, except the conclusions about the GPT-4o backend, presented in these sections are also presented in [18] (© 2023 IEEE).

This chapter demonstrates the possibility of combining natural language and pointing gestures to achieve multimodal error correction. In particular, this chapter introduces a system that determines the name of an unknown or misclassified entity along with the corresponding region of interest (RoI) in the camera image. The two outputs can be utilized for the incremental learning of new entities. If an utterance does not involve a correction, the system is capable of detecting this in 100 % of the test cases with the VL-T5 backend and in 92.50 % of the test cases with the GPT-4o backend, and the corresponding utterances can be forwarded to another component. This indicates that the proposed method enhances the value of an interactive robot system without compromising its performance for the VL-T5 backend. The overall accuracy rates of 34.31 % on the initial attempt and 45.37 % on up to three attempts on the most practical approach (the middle frame of the max. first 30 frames) represent significant progress toward establishing a reliable component. The GPT-4o backend has a higher overall accuracy of 53.27 % on the first attempt and 60.05 % on up to three attempts but has the mentioned lower accuracy for detecting whether an utterance includes a correction. A problem was that the reference RoI was often not in the proposed RoI. Removing these elements, improves the overall accuracy of the VL-T5 backend to 49.67 % on the initial attempt and 65.69 % on up to three attempts and the overall accuracy of the GPT-4o backend to 77.12 % on the initial attempt and 86.93 % on up to three attempts. The much better accuracy from the GPT-4o model resulted from its knowledge about the entities in the test dataset. The test dataset videos are from the year 2022 and the utterances are from the year 2023. Both datasets are before the cutoff date of the used GPT-4o model that knows most of entities, tested with the prompt "Do you know what ENTITY is and how it looks?". One of the exceptions is the entity "cuppy" that should be a pet name for a cup.

5. Error Correction with Pretrained Models

5.1. Introduction

The ability to correct entities presented in the previous two chapters is one of the most important abilities of an error correction system. Especially in a robot for which not the correct transcript of the speech command is important but actions and what entities the action should have. However, the ability to correct all types of errors is also important. In some texts like laws or political speeches often every word and every punctuation is very important and small changes can have huge impacts. Texts with huge impacts are often written by busy people and an error correction system working on all types of errors could enable them to use natural language processing components like automatic speech recognition (ASR) and machine translation and be more flexible about their workplace, for example, improving a law during workout instead of sitting at their desk all the time.

Based on the learnings of the third chapter, two approaches for the error correction of all error types are presented in this thesis: first, collecting error correction data and fine-tuning a sequence-to-sequence model and second, engineering a custom prompt with the task description and some examples (few-shot) for a large language model (LLM). In this chapter, the first approach is presented and the second approach is presented in the next chapter.

This chapter shows why a sequence-to-sequence approach and not a sequence labeling approach is used, how the data for fine-tuning were collected and gives more insights about the architecture. The evaluation results of the fine-tuned model are also presented.

Like in the previous two chapters, a live system is presented again. Live systems have the advantage to test the developed models with many people in a realistic environment and to get feedback from them. Collecting data to build and evaluate such systems is often synthetic or biased because the interfaces are different, or people participate out of monetary interests and try to increase the monetary output.

5.2. Methodology

In Chapter 3, multiple error correction approaches are presented. All approaches can be categorized either as sequence labeling or as sequence-to-sequence approaches. A sequence labeling approach has a huge disadvantage in a real-world error correction application. All repairs must have the property that they can substitute the reparandum without any transformation, because only words that are in the correction utterance can be copied to the utterances that is corrected. A spelled word does not fulfill this requirement, because in most ASR systems, spelled words are not mapped to the spelled word but are represented as spelled characters separated by a character like space or hyphen and the casing is often not like in the represented word. Even if a preprocessing step would be introduced that converts spelled words to the real words, there are still cases where the repair and the intended version of the reparandum are not equal. For example, by describing changes on character level like “Stephan with F” or changing the letter case like “all characters in the company name should be in uppercase”. Often, it is easier to describe changes like “use the adjective of unconstraint” and “I don’t mean invidia but the company that produces GPUs”. This is especially relieving for people with spelling weakness.

The goal of the copy mechanism was to bring the advantages of the sequence labeling approach (to copy tokens) to the sequence-to-sequence approaches, as you can see in Chapter 3, it is shown that the results are not promising. Therefore, this approach is not considered for error correction on all types of errors. The approach for this chapter is to collect error correction data and fine-tuning a sequence-to-sequence model since the T5 large model has good results in Chapter 3, it should also be used for the error correction on all types of errors. The implementation is also the same as in Chapter 3.

In contrast to the previous two chapters, no explicit reparandum and repair output is generated, because it can be easily extracted by calculating the Levenshtein distance [82] and backtracing it using dynamic programming

[157] using the following Python package [99]. The result from the backtrace contains which phrases were substituted, inserted, or deleted. In Figure 5.1, an example is depicted.

correct	Today	is	a	nice	day	and	really	everything	is	good	.	
wrong	Today	is	a	niece	day	and			is	good	so	.
operations				sub			del	del			ins	
repair and reparandum	niece $\rightarrow$ nice, $\epsilon \rightarrow$ really everything, so $\rightarrow \epsilon$											

Figure 5.1.: Example of extracting repair and reparandum pairs with the Levensthein distance and backtracing, ϵ is the empty word

5.3. Dataset

There is a lack of good error correction datasets, therefore a new dataset was created. In order to do so, a wrong transcript, a correct transcript and the correction that corrects the wrong transcript to the correct transcript are necessary. The created dataset consists of two parts, that were collected by two different web applications.

For the first part, conference talks from the conference of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020) were used. It was held online because of the COVID-19 pandemic and therefore there are recordings of all the talks. These talks were transcribed by a from the Interactive System Labs provided ASR system based on [151, 59]. In addition, a human corrected these transcripts. Only the corrections were missing. To collect corrections, a web application was developed. This web application is based on the Slate Transcript Editor [104] (that is inspired from the React Transcript Editor of the BBC [25]). This editor was used in the ISL lab to adapt it to error correction [66] and is called Transcription Corrector. I improved it a little bit further [16]. Users can upload a ZIP file containing a video file and a VTT file [26] (format for subtitles with timestamps) to the web application. In the web application, there is a video player to play the provided video and the timestamps and the transcripts of the segments are displayed. To jump in the video between the segments, users can click on the matching timestamp. To every segment, a correction can be recorded and the recording can be played. The web application is shown in Figure 5.2. The data creators were provided with 10 minutes ACL 2020 talks. The talks were segmented by the ASR segmenter and are not always complete sentences. The errors of the automatic transcript were limited to two errors per segment to avoid redictation as most used correction technique.

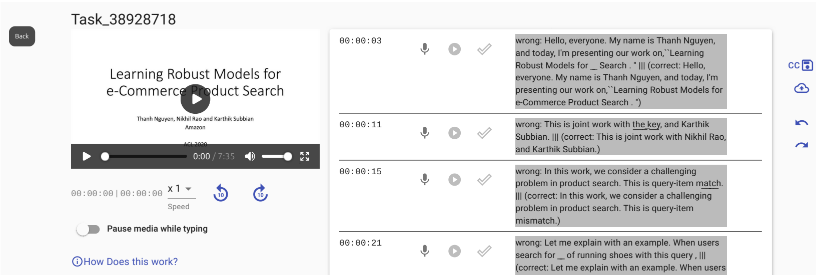


Figure 5.2.: Screenshot of the Transcription Corrector

The sheet with the instructions for the data creators is in the attachment A.3.

The corrections are transcribed by the ASR system Whisper [110] in a version that is optimized for online transcriptions that is used in the Lecture Translator framework [58]. In the collected data, the transcripts of the speech corrections are often not sufficient to correct the wrong transcripts. To have a good dataset, a human had to rate which corrections are sufficient. For this task a very simple web application was provided. First, the correct transcript appears, in the next line the wrong transcript, and in the next line after that the transcript of the correction. On the right of the transcript of the correction is a valid button. The task was to press valid on all corrections that could be used to correct the wrong transcript accurately. After every click, the task identifier was sent to a server app and the correction was saved as valid in a PostgreSQL [139] database. In Figure 5.3, a cutout of the web application is depicted. Only valid rated corrections are used in the first part of the dataset.

In the end, 1029 corrections of 6 data creators remained. The output of two data creators with 273 corrections are exclusively used for the test dataset, one data collector with 128 corrections is exclusively used for the validation dataset, and the remaining 628 corrections are used for the training dataset.

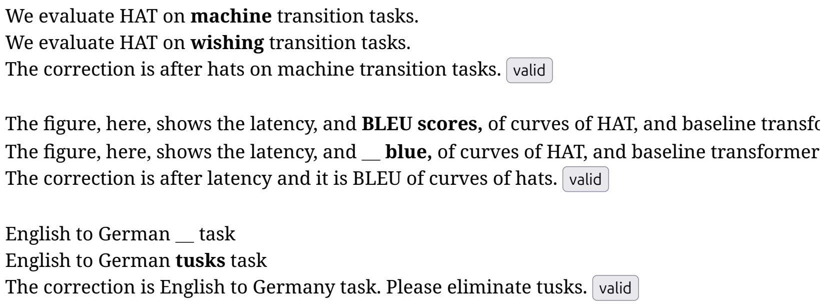


Figure 5.3.: Screenshot of the Transcription Corrector verifier

The problem with the first web application is that many of the provided corrections were not sufficient to correct the wrong transcript. A direct feedback mechanism would solve this problem. If it is not possible, even for a human, to correct the error, the data collector has to provide a new correction. This would lead to multiple turns corrections which are also important data because the system will make errors and a second try including the history can be used to accurately correct the transcript. To achieve this, a new web application was developed from scratch. In the frontend, it uses React and in the backend a Django server app with a REST interface. As database, PostgreSQL is used. Instead of uploading a ZIP file containing a video file and a VTT file, all segments are in the database and a segment randomly given to the data creators.

The data for the second web application contain the same ACL conference talks as in the first part. In addition, texts from the bookcorpus dataset [177] and CNN / Dailymail dataset [50, 125] are added. The ACL conference talks are the same as used in the collection with the Transcription Corrector but the data creators only get one segment at a time. The bookcorpus dataset and the CNN / Dailymail dataset only have correct texts but no audio files because the data are text data. The errors were introduced synthetically. In a segment, one or two errors are introduced. It is randomly chosen whether it is an insertion, deletion, or substitution. The inserted, deleted, or substituted phrase consists of one word. There are four substitution approaches. The first two substitution approaches replace the word with the phonological nearest word. The most common 10 000 words from the Brown corpus [36] provided by the Natural Language Toolkit (NLTK) [10] were converted to the

International Phonetic Alphabet (IPA) representation of them with the Python package “eng-to-ipa” [96]. The word that should be replaced is also converted to the IPA and is replaced by the word with the smallest Levensthein distance between the IPA (the calculation is done with the NLTK). The difference of the first and second substitution approach is that in the first approach all words can be replaced and in the second only words that are classified by the Python library spaCy [57] as one of the following labels: “FAC” (facility), “ORG” (organization), “LOC” (location), “PRODUCT”, and “EVENT”. The third substitution approach only replaces names (classified by spaCy with the label “PERSON”) and the replacement is chosen from a list with 200 names. The fourth substitution approach is like the third, but it was checked whether before the name is one of the following words (ignoring the case and whether a point is following or not): “Mr”, “Ms”, or “Mrs”. The list with the last names that can replace a found last name is 236 names long.

A data collector gets a wrong text and a correct version and has to speak a correction. A human verifier must connect to a corrector and gets the wrong text and the correction as audio. In Figure 5.4, the view of the corrector is depicted. Only the last verifier output is shown and no transcript is shown per default. Both, all verifier outputs and all transcripts can be manually activated and in Figure 5.5 the corrector view with both options activated is shown. In Figure 5.6, the view of the verifier is shown. The verifiers only see the wrong text in the first turn and in the other turns, they only see the last text they have corrected. After a verifier send a corrected text, based on the received audio from the data collector, the verifier’s view is empty.

From 20 creators 10 248 correction histories were collected with the second web application that builds the second part of the dataset presented in this section. From the collected correction histories 8926 are one turn, 1130 are two turns, and 192 are three turns. Correction histories with more than three turns are ignored.
In the test dataset, there are segments from 14 ACL talks (none of these 14 talks are in the train or validation set of the first part of the dataset presented in this section) collected from three data creators who only provide data to the test dataset but not to the validation or training dataset. That makes 370 correction histories (292 correction histories with one turn, 57 correction histories with two turns, and 21 correction histories with three turns).
The validation dataset consists of 6 ACL talks (none of these 6 talks are in the test or train set of the first part of the dataset) from two data creators that only provide data to the validation dataset. In total, there are 88 corrections

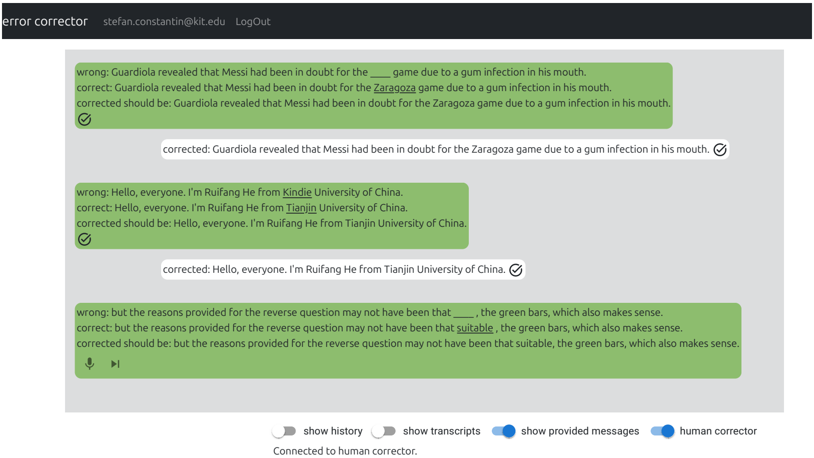


Figure 5.4.: Screenshot of the data collection web application for the second part of the dataset in the default view

histories, where 73 have one turn, 10 two turns, and 5 three turns. The remaining correction histories that are from other data creators as used in the test and validation set are 9790 in total and 8561 of them have one turn, 1063 have two turns, and 166 have three turns.

To extend the amount of collected data, additional synthetic data is added to the training dataset. The wrong and correct segments are from ACL talks (that are not included in the validation or test dataset). For every error type, there are correction templates. A segment can have up to two errors. If more than one error occurs in one segment, the correction templates are connected with the word "and". There are 42 substitution templates, 12 deletion templates (deletion in wrong segment), and 17 insertion (insertion in wrong segment) templates. In the substitution templates the correct word is there and/or spelled. Sometimes, there is context before, after, or before and after the corrected words. The number of words as context is randomly chosen from 1 to 3 words (for context before and after the corrected words, the number of words is separatly randomly chosen for both directions). To contribute to bad ASR outputs, some words are wrong written like "he plays" instead of "replace". Multiple turns can be emulated. All templates are in the Appendix A.2.

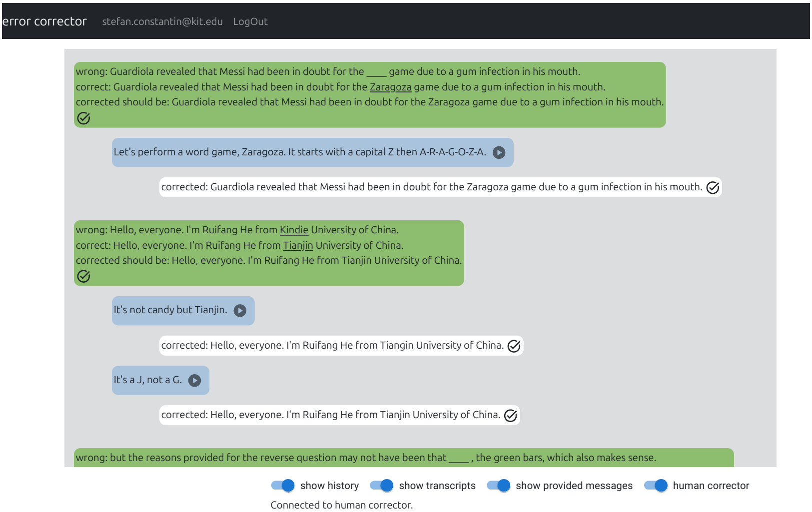


Figure 5.5.: Screenshot of the data collection web application for the second part of the dataset in the view with more details (must be manually activated)

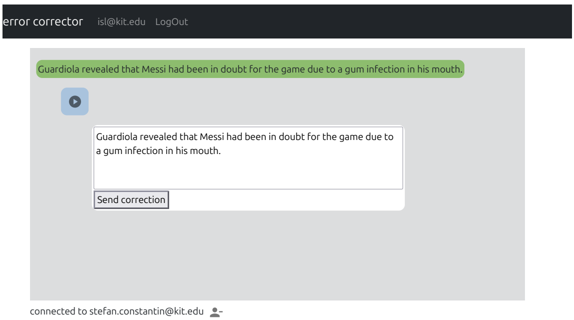


Figure 5.6.: Screenshot of the data validation web application for the second part of the dataset

5.4. Evaluation

The T5 model was fine-tuned for one epoch with the following hyperparameters: Adam optimizer [73] with a learning rate of $2.5 \cdot 10^{-4}$ and batch size of 12. In the embedding layer, the first two encoder blocks were frozen.

The parts of the dataset are evaluated with the metric accuracy for the complete segments. If a complete segment is corrected correctly, the number of successes is incremented by one and for every segment the total number of complete segments is incremented by one. In the end, the number of successfully corrected segments is divided by the total number of segments is the accuracy. The in- and output length was limited to 256. Only one correction history from the dataset exceeds with 370 tokens (208 words) In Table 5.1, the results are given. For all data, the validation accuracy is 62.96 % and the test accuracy is 63.14 %.

part	validation	test
1. part	68.75 %	68.86 %
2. part, 1 turn	53.42 %	58.90 %
2. part, 2 turns	70.00 %	61.40 %
2. part, 3 turns	40.00 %	52.38 %
all	62.96 %	63.14 %

Table 5.1.: Accuracies on the validation and test dataset

The first part of the dataset, which has only one turn corrections, has a better performance with 68.75 % and 68.86 % for the validation and test dataset, respectively, than the second part of the dataset that has only one turn corrections. A reason is, that for the first part only from the transcript valid segments were selected by a human and for the second part, it was enough if the audio was valid.

The averaged word error rate (WER) and character error rate (CER) before the correction were calculated by calculating the WER and CER for every segment to correct and averaging the results. For the averaged WER and CER after the correction, the WERs and CERs of the corrected segments after the last correction are used to form the average. The averaged WER before and after the last correction are depicted in Table 5.2 for the validation dataset and in Table 5.5 for the test dataset. The for averaged WER and CER for successfully corrected segments are depicted in Table 5.3 and the averaged WER and CER for the unsuccessfully corrected segments are depicted in Table 5.4 for the validation dataset and in Table 5.6 for the test dataset.

part	WER before	WER after	WER change	CER before	CER after	CER change
1. part	25.08	9.23	−63.22 %	14.70	5.04	−65.69 %
2. part, 1 turn	32.62	9.13	−72.01 %	16.52	5.47	−66.88 %
2. part, 2 turns	36.95	6.47	−82.50 %	15.94	1.31	−91.79 %
2. part, 3 turns	19.93	13.31	−33.21 %	11.41	2.99	−73.75 %
all	28.06	9.16	−67.35 %	15.30	4.97	−67.52 %

Table 5.2.: Averaged WERs / CERs of the validation dataset containing all segments

	validation		test	
part	WER before	CER before	WER before	CER before
1. part	18.92	14.06	18.08	9.45
2. part, 1 turn	32.80	18.63	34.02	24.27
2. part, 2 turns	40.26	19.27	30.35	16.43
2. part, 3 turns	19.17	7.84	21.10	12.05
all	24.20	15.61	26.07	16.49

Table 5.3.: Averaged WERs / CERs of the validation and test dataset containing all segments that could be corrected accurately, WERs and CERs after the correction need not be provided, because they are all 0

5.5. System

The web application that is used to collect multi-turn correction histories (depicted in Figure 5.4 and Figure 5.5) is extended so that the system that is evaluated in Section 5.4 can be used instead of the human verifier to correct utterances. The model and a wrapper for the model are bundled to a model archive (.mar) file that can be executed by PyTorch Serve [24]. Instead of sending the wrong texts, the correction, and history to a human data verifier a REST API call to the PyTorch Serve is issued and the result is presented in the view. To not only handle given texts, “show provided messages” can be deactivated (see Figure 5.4) and one’s own text can be dictated.

part	WER before	WER after	WER change	CER before	CER after	CER change
1. part	36.44	26.24	−27.99 %	15.87	14.35	−9.61 %
2. part, 1 turn	32.41	19.60	−39.53 %	14.09	11.75	−16.66 %
2. part, 2 turns	29.23	21.56	−26.24 %	8.16	4.36	−46.54 %
2. part, 3 turns	20.45	22.19	8.52 %	13.79	4.99	−63.80 %
all	34.01	23.28	−31.56 %	14.81	12.62	−14.79 %

Table 5.4.: Averaged WERs / CERs of the validation dataset containing all segments that could not be corrected accurately

part	WER before	WER after	WER change	CER before	CER after	CER change
1. part	19.36	5.62	−70.96 %	10.25	3.10	−69.72 %
2. part, 1 turn	34.77	9.31	−73.23 %	24.13	7.52	−68.81 %
2. part, 2 turns	33.33	5.60	−83.19 %	16.30	2.05	−87.44 %
2. part, 3 turns	29.76	11.55	−61.17 %	15.70	6.96	−55.66 %
all	27.94	7.49	−73.20 %	17.27	5.14	−70.21 %

Table 5.5.: Averaged WERs / CERs of the test dataset containing all segments

5.6. Conclusion

The results are promising. In 63.14 % of the texts of the test dataset, the wrong transcribed texts could be accurately corrected. Especially, the first part has good results. 68.86 % of the texts could be accurately corrected. The first part has a higher accuracy because only correction transcripts that can accurately corrected by a human are included in the first dataset. In the second part, it is only ensured that the audio of the corrections has enough information to correct the text accurately. Because of ASR errors, the transcript can lose this ability. Some of the unsuccessfully corrected segments have this problem. The averaged WERs are even more promising than the accuracies. Even in the segments that could not be corrected completly correct, the averaged WER was changed by −35.89 %. That means although the segments cannot be corrected completly, the averaged WER is lower than before. The accuracies for two and three turns corrections were comparable to the only one turn corrections. These facts and some quality tests with the system let deduct that after multiple tries most errors can be corrected. Especially

part	WER before	WER after	WER change	CER before	CER after	CER change
1. part	21.96	17.05	−22.36 %	11.88	9.42	−20.73 %
2. part, 1 turn	35.85	22.65	−36.83 %	23.92	18.31	−23.46 %
2. part, 2 turns	38.08	14.52	−61.87 %	16.09	5.30	−67.04 %
2. part, 3 turns	39.28	24.27	−38.22 %	19.72	14.62	−25.87 %
all	31.03	19.89	−35.89 %	18.56	13.67	−26.35 %

Table 5.6.: Averaged WERs / CERs of the test dataset containing all segments that could not be corrected accurately

after knowing the behavior of the system and the adaptation of the users to this behavior. However, this is not the goal of this thesis. The work should provide an unconstrained error correction without requiring users to learn how to correct. The next chapter shows that an error correction system using LLMs can achieve this goal.

A big advantage, compared to LLMs, presented in the next chapter, is the smaller number of parameters of the used T5 large model and therefore it is less computation resource intensive. That leads to less latency and the possibility to run it on personal devices (in the moment personal devices must be powerful to run these models, but the performance of personal devices is increasing) to have better privacy. Probably, LLMs, that can be prompted for a good correction quality, can run on normal personal devices one day, but that will take some more time.

6. Unconstrained Error Correction with Large Language Models

6.1. Introduction

Although the fine-tuned model, evaluated in the previous chapter, has good results on the collected data, it is limited to the data that were used to fine-tune it. Examples like “I don’t mean invidia but the company that produces GPUs” are not possible and it is difficult to imagine to be able to collect data for all possible corrections. However, the ability for unconstrained error corrections is very important to create error correction systems that are useful. Systems that need an exact redictation because you have to use patterns like “Replace X with Y” or “Change it to X” will get problems because redictation only works if the pronunciation of the initial phrase was bad, but often the problem is not the pronunciation that can be improved by a redictation but the system cannot recognize this phrase and therefore more creative corrections are needed. If such creative corrections are not unconstrained, they require cognitive load because patterns must be memorized. Especially if error correction should be used during other cognitive requiring actions like car driving, this is not possible. Spelling seems to be a solution for all errors, but it is slow, annoying, and for people with spelling weakness, it is easier to describe something than to spell.

Large language models (LLMs) have a huge knowledge about natural language and facts. In this chapter, it is proposed to engineer a custom prompt for an LLM that offers unconstrained error correction. This prompt is evaluated in this chapter in the same manner (same dataset and same metrics) as the fine-tuned model in the previous chapter to compare the results. In addition, some qualitative tests are made.

In this chapter, a web application for real-world error correction is presented that uses an LLM with a custom prompt as backend. Text can be selected by mouse and via speech and the selected part can be corrected without any constraint to the correction. This system is described in Section 6.4 and evaluated in Section 6.6.

6.2. Methodology

The LLM used for the unconstrained error correction is used with an engineered custom prompt. This prompt was tailored to the unconstrained error correction task by trial and error. The prompt is in the Appendix A.4. The prompt has 355 words in explanation and nine example corrections. Eight of the example corrections have one turn and one has five turns. The maximal output tokens are limited to 256. As temperature [44] 0.1 is used. A low value is chosen because the error correction is not so creative. A frequency penalty and a presence penalty of 0 are used to deactivate both, because such a penalty would punish accurately corrected text if in the text certain words are often repeated or if the text is not diverse enough, respectively. The remaining attributes are the default attributes of GPT-4o [101] version 6th August 2024. The LLM was used via the official OpenAI batch API.

6.3. Evaluation

For evaluation, the same dataset and the same metrics as in the previous chapter are used to compare the performance of the fine-tuned model with the LLM. The output length was limited to 256. No correction history from the validation or test dataset exceeds the limits.

In Table 6.1, the accuracies are given. For all data, the validation accuracy is 64.81 % and the test accuracy is 65.32 %. This is only slightly better than the fine-tuned model, but it is to consider that even if it was valued to separate training and test data as good as possible, the type of data collection leads to similar data.

The first part of the dataset that only has one turn corrections has with 74.22 % and 79.49 % for the validation and test dataset, respectively, a better performance than the part of the second part of the dataset that has only one turn corrections. A reason is that for the first part of the dataset only valid

dataset	validation	test
1. part	74.22 %	79.49 %
2. part, 1 turn	49.32 %	58.90 %
2. part, 2 turns	60.00 %	40.35 %
2. part, 3 turns	60.00 %	38.10 %
all	64.81 %	65.32 %

Table 6.1.: Accuracies on the validation and test dataset

sentences were selected from the transcript by a human and for the second dataset, it was enough if the audio was valid.

How the average values are formed is defined in Section 5.4. The averaged word error rate (WER) and character error rate (CER) before and after the correction(s) are depicted in Table 6.2 for the validation dataset and in Table 6.5 for the test dataset. The averaged WER and CER for successfully correctable segments are depicted in Table 6.3 and the averaged WER and CER for the segments that could not be corrected completly are depicted in Table 6.4 for the validation dataset and in Table 6.6 for the test dataset.

part	WER before	WER after	WER change	CER before	CER after	CER change
1. part	25.08	9.56	−61.87 %	14.70	4.22	−71.26 %
2. part, 1 turn	32.62	20.09	−38.40 %	16.52	10.33	−37.46 %
2. part, 2 turns	36.95	20.21	−45.31 %	15.94	6.14	−61.46 %
2. part, 3 turns	19.93	5.69	−71.43 %	11.41	1.87	−83.58 %
all	28.06	13.52	−51.80 %	15.30	6.32	−58.66 %

Table 6.2.: Averaged WERs / CERs of the validation dataset containing all segments

6.4. System

The real-world interactive error correction is embedded in the Lecture Translator framework [58]. This framework has a control component that can be extended by multiple components that are structured in a pipeline. An upstream component can pass arbitrary data to a downstream component. A component can have $n \in \mathbb{N}_0$ upstream and $m \in \mathbb{N}_0$ downstream components.

part	validation		test	
	WER before	CER before	WER before	CER before
1. part	22.80	15.16	19.30	10.00
2. part, 1 turn	23.53	15.54	28.01	20.39
2. part, 2 turns	32.62	13.07	31.48	16.27
2. part, 3 turns	21.19	11.53	35.34	14.65
all	23.51	15.07	24.37	15.23

Table 6.3.: Averaged WERs / CERs of the validation and test dataset containing all segments that could be corrected accurately, WERs and CERs after the correction need not be provided, because they are all 0

part	WER before	WER after	WER change	CER before	CER after	CER change
1. part	27.74	20.75	−25.22 %	14.16	9.16	−35.29 %
2. part, 1 turn	41.45	39.64	−4.37 %	17.47	20.38	16.68 %
2. part, 2 turns	43.45	50.52	16.27 %	20.24	15.36	−24.13 %
2. part, 3 turns	18.06	14.24	−21.15 %	11.23	4.68	−58.28 %
all	33.14	28.64	−13.59 %	15.54	13.39	−13.85 %

Table 6.4.: Averaged WERs / CERs of the validation dataset containing all segments that could not be corrected accurately

There is an API that a frontend can pass data to components and components can pass data to a frontend. There are a web application, a multiplatform Python command line client, and an Android and iOS app as frontend. In the web application, the output of every component that sends output to the frontend gets a window within the web application that can be activated and deactivated. This approach enables to see the transcript of an ASR system in one window and a translation of this transcript in another window, and a summary in another window.

The error correction system has two modes: the dictation and the evaluation mode. In the dictation mode, a text can be dictated and if something should be corrected, the correction stage can be started. If the correction is done, the correction stage can be left to go back to the dictation stage. To evaluate the system better, an evaluation mode was developed. It has in addition a reference text and it is always in the correction stage. In Figure 6.1, a screenshot of the evaluation mode is depicted. In the evaluation mode, the

part	WER before	WER after	WER change	CER before	CER after	CER change
1. part	19.36	5.55	−71.35 %	10.25	2.21	−78.45 %
2. part, 1 turn	34.77	13.24	−61.91 %	24.13	6.87	−71.52 %
2. part, 2 turns	33.33	20.58	−38.26 %	16.30	15.13	−7.17 %
2. part, 3 turns	29.76	18.13	−39.09 %	15.70	12.36	−21.27 %
all	27.94	10.79	−61.39 %	17.27	5.80	−66.39 %

Table 6.5.: Averaged WERs / CERs of the test dataset containing all segments

part	WER before	WER after	WER change	CER before	CER after	CER change
1. part	19.47	15.14	−22.22 %	10.70	6.03	−43.61 %
2. part, 1 turn	44.46	32.23	−27.52 %	29.49	16.72	−43.29 %
2. part, 2 turns	34.59	34.50	−0.25 %	16.32	25.37	55.41 %
2. part, 3 turns	26.32	29.28	11.24 %	16.35	19.97	22.14 %
all	32.96	25.97	−21.20 %	20.13	13.98	−30.58 %

Table 6.6.: Averaged WERs / CERs of the test dataset containing all segments that could not be corrected accurately

differences are bold and a missing word is represented by a fullwidth low line Unicode character and on the other side (either reference or transcript) bold.

In the correction stage, text can be selected or a cursor can be set. If text is selected, the selected text and only the selected text can be edited by speech and if a cursor is set, new text can be inserted at the place of the cursor by speech. If all of the selected text is deleted, the selection changes to a cursor and if text is inserted at the position of a cursor, the inserted text is selected. After a change, the differences are re-calculated to display correctly the differences in bold and the missing words as fullwidth low line Unicode character.

For good editing abilities, the prompt with the task description and the provided examples as well as the input for the specific requested error correction had to be optimized.

In qualitative tests, it could be seen that GPT-4o has a bad performance in correcting subwords. Therefore, a selection is extended that only whole

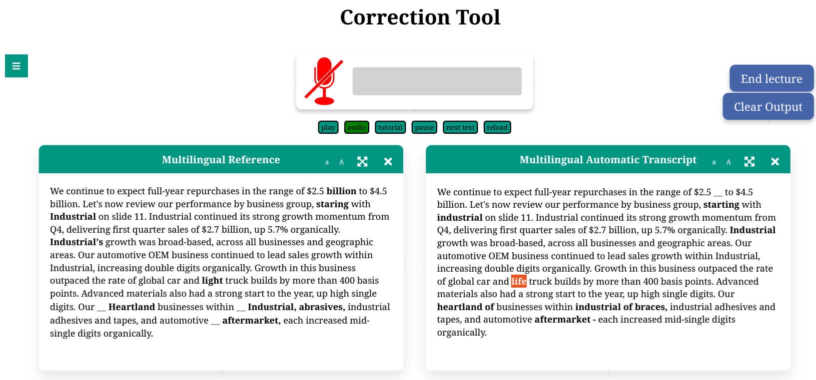


Figure 6.1.: Screenshot of the real-world error correction in evaluation mode, on the left side is the reference text and on the right side the transcript that should be corrected, differences are bold and missing words are represented by a fullwidth low line Unicode character, the word life is selected to be edited, on the upper left hand there are three bars hovering shows all windows that can be activated or deactivated

words are in the selection. Beginning and ending spaces also cause problems and therefore the previous or next, respectively, word is also selected. Only coherent text can be selected, even coherent text of multiple sentences can be selected. The selected part is embedded in the custom HTML tag “<span-highlight>”. Using HTML tags has the advantage that the LLM and the window in the web application that displays the text can use the same text and no conversion is necessary. Because in the component every sentence is embedded in its own span tag, the selected part of every sentence must have its own selection tag to have proper nesting. If a user sets a cursor to insert text, it is represented as the HTML tag “<cursor-image>”. There are two prompts, one for the correction of selected text (see Figure A.5 in the Appendix) and one for the insertion of text when the user set a cursor (see Figure A.6 in the Appendix). To include context, all sentences that have selected text or a cursor are integrated in the prompt. In addition, all corrections and their results are integrated in the prompt so that the correction history can be used. The span highlights tags will be renamed to “<span-highlight-n” where $n \in \mathbb{N}_0$ represents the index, starting with 0 on the first in the prompt included sentence. The LLM is GPT-4o and is running in the Microsoft Azure cloud because there was a lower latency than with the OpenAI API.

The differences of reference and transcript were calculated by the Levensthein distance and backtracing its calculation [82, 157] with the following Python package [99]. Substitutions, insertions, and deletions were equally weighted. Insertions are selected first.

For the input audio and the transcribed text a forced alignment was done [77, 52]. This enables to play the audio of the selected text. Even if the corrections change the text, the audio playback still works, because the changes are tracked with help of the Levensthein distance and the backtrace of its calculation.

The user corrections are incrementally processed. That could reduce the latency and according to [98, 20] does not decrease the performance much. The evaluation in [20] considered that incomplete sentences are noisy and used techniques from [132] to simulate them.

6.5. System Dataset

To evaluate the interactive system, two datasets were prepared. The first dataset is based on the sentence segmented and reference transcripts providing earnings conference calls dataset of S&P 500 companies [109]. Twelve talks were selected (a good distribution over different branches). To every sentence, the audio was transcribed with the Whisper version that the Lecture Translator framework uses. The Levensthein distance and backtracing its calculation were used to calculate the changes between the human transcripts and the automatic transcripts. Substitutions are in the reference and transcript bold. Deletions of the reference are bold in the reference and in the transcript the Unicode sign fullwidth low line as placeholder for every missing word is used. For insertions in the reference, the insertions are bold in the transcript and the Unicode character fullwidth low line for every added word is inserted in the transcript. In half of the talks, only the ASR errors that are content words are used. The other errors are replaced by the reference. For the POS tagging, the Natural Language Toolkit (NLTK) [10] with the averaged perceptron tagger was used and in this thesis, content words are defined as part of one of the following part of speech tags “NN”, “NNS”, “NNP”, “NNPS”, “VB”, “VBD”, “VBG”, “VBN”, “VBP”, “VBZ”, “JJ”, “JJR”, “JJS”, “RB”, “RBR”, “RBS”, and “CD”. To adjust the type of reference transcript and the automatic transcript, contraction like “won’t” and “can’t” were converted into the long form. A forced alignment [77, 52] approach is used to assign the start and the end timestamp for every word. This enables that the audio of the selection can be played. If the included segments exceed 60 seconds audio in the current cutout, a new cutout is created. In Figure 6.7, all used earnings conference calls, which cutout is used, and which only have content words as errors are depicted.

set	company	date	abbre-viation	only content words	minute of cutout begin
1	3M	2017/04/25	MMM	true	7
1	Amazon.com Inc.	2017/02/02	AMZN	false	7
1	AMETEK Inc.	2017/02/07	AME	true	12
1	General Motors	2017/02/07	GM	false	8
2	Hilton Worldwide Holdings Inc.	2017/02/15	HLT	true	24
2	Hormel Foods Corp.	2017/02/23	HRL	false	20
2	JPMorgan Chase & Co.	2017/07/14	JPM	true	15
2	Kimberly-Clark	2017/01/24	KMB	false	1
3	Kraft Heinz Co.	2017/02/15	KHC	true	4
3	Merck & Co.	2017/02/02	MRK	false	0
3	Tiffany & Co.	2017/05/24	TIF	true	5
3	Twitter Inc.	2017/02/09	TWR	false	4

Table 6.7.: Earnings conference calls cutouts

The second dataset is based on the ACL conference talks also used in the evaluation of the LLM in Section 6.3. For the evaluation of the system, the data were sentence aligned. For this sentence alignment, the sentence tokenizer of NLTK is used. This is necessary because in the system whole sentences are selected and assigning segments to sentence would be confusing for the user. A second problem with the data of the ACL conference talks is that there are often long pauses because the presenters had to think about their words. To enable audio playback without waiting too long for the audio (often the start time of a word is not at the beginning of the word but at the end of the previous word and therefore playback of a word can take a long time) pauses longer than two seconds were deleted. For the silence detection Pydub [117] was used. Substitutions, insertions, and deletions are shown in the same way than in the earnings conference calls dataset and if the included sentences exceed 60 seconds audio in the current cutout, a new cutout is created. An overview of the ACL conference talks dataset is in Figure 6.8.

All cutouts of both datasets contain 273 errors. In Figure 6.2, it is depicted how many errors each cutout has. In average, there are 11.38 errors per cutout and the minimal number of errors is 5 and the maximum number of errors is 15. An error can be a substitution, an insertion, or a deletion. All of these three types can have different lengths.

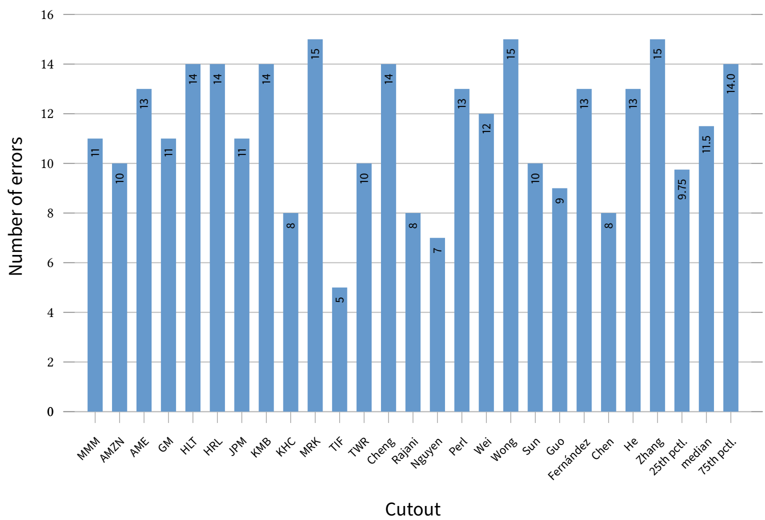


Figure 6.2.: Number of errors per cutout

set	ACL conference talk	abbre-viation	minute of cutout begin
1	SpellGCN: Incorporating Phonological and Visual Similarities into Language Models for Chinese Spelling Check	Cheng	3
1	ESPRIT: Explaining Solutions to Physical Reasoning Tasks	Rajani	8
1	Learning Robust Models for eCommerce Product Search	Nguyen	3
1	Low Resource Sequence Tagging using Sentence Reconstruction	Perl	2
2	Effective Inter-Clause Modeling for End-to-End Emotion-Cause Pair Extraction	Wei	6
2	Contextual Neural Machine Translation Improves Translation of Cataphoric Pronouns	Wong	0
2	Knowledge Distillation for Multilingual Unsupervised Neural Machine Translation	Sun	4
2	A Frame-based Sentence Representation for Machine Reading Comprehension	Guo	1
3	Transition-based Semantic Dependency Parsing with Pointer Networks	Fernández	4
3	Content Word Aware Neural Machine Translation	Chen	1
3	Learning to Tag OOV Tokens by Integrating Contextual Representation and Background Knowledge	He	2
3	WinoWhy: A Deep Diagnosis of Essential Commonsense Knowledge for Answering Winograd Schema Challenge	Zhang	1

Table 6.8.: ACL conference talks cutouts

6.6. System Evaluation

Fifteen persons participated in the evaluation. Nine of them were able to program. The discrimination in programmers and non-programmers should help to research whether the thinking of a programmer is necessary to correct successfully the errors with the interactive error correction or whether it can be used by everyone. The programmers are prefixed with “P” and the non-programmers are prefixed with “N”. Every participant had to do eight one minute cutouts of ACL conference talks and earnings conference calls (four ACL conference talk cutouts and four earnings conference call cutouts). One of the non-programmers (N4) skipped one cutout of the earnings conference calls cutouts (the earning conference call from Hilton Worldwide Holdings Inc. (HLT)).

Three programmers and two non-programmers shared the same cutouts. That means, in total, there were 12 different ACL conference talks and 12 different earnings conference call cutouts which contain 273 distinct errors in total. P1 to P3 and N1 to N2 did the first cutouts part, P4 to P6 and N3 to N4 did the second cutouts part, and P7 to P9 and N5 to N6 did the last cutouts part. Considering the shared cutouts, 1351 errors in 119 cutouts had to be corrected.

The participants were allowed to do two pauses during a task, but no one did it more than once. In total, in 8 of the 119 corrected cutouts this functionality was used.

On average, 98.91 % of all errors of a cutout could be corrected. Considering the median and even the 25th percentile, 100 % of all errors of a cutout could be corrected. In Figure 6.3, the average percentage of accurately corrected errors of a cutout is averaged for each participant individually. In addition, the 25th percentile, the median, and the 75th percentile are depicted. In the following such a grouping by participant or by cutout is called aggregation by participant or aggregation by cutout, respectively. The aggregation by cutout for the average percentage of successful corrections per cutout is depicted in Figure 6.4.

Besides correcting all errors, one important point in error correction is that no additional errors are introduced. In relation to the number of all existing errors of a cutout, on average only 0.46 % substitution errors, 0.12 % insertion

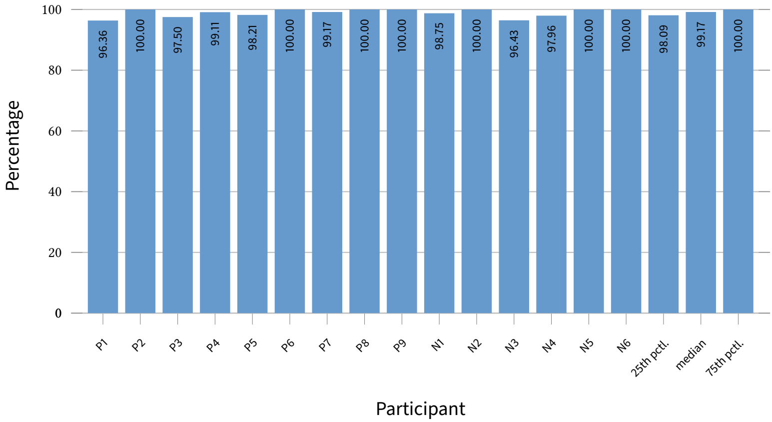


Figure 6.3.: Average cutout percentage of successful corrections (aggregated by participant)

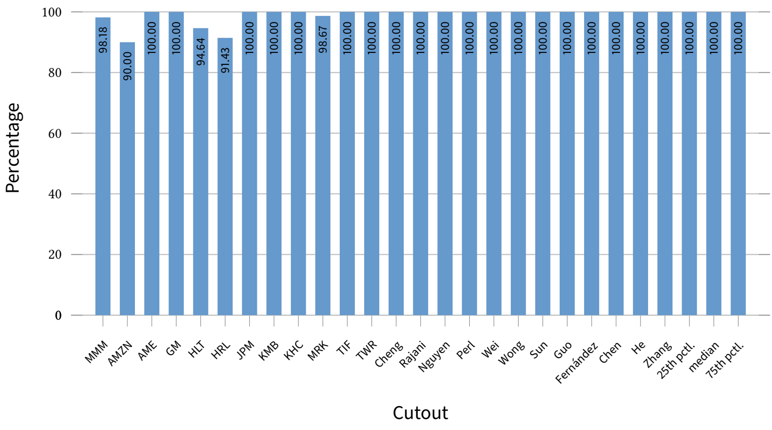


Figure 6.4.: Average cutout percentage of successful corrections (aggregated by cutout)

in reference errors, and 0.26 % deletion in reference errors were introduced in a cutout.

Before the correction, the word error rate (WER) is on average per cutout 10.66 (7.95 25th percentile, 10.14 median, and 12.04 75th percentile) and after the correction 0.12 (0.00 25th percentile, 0.00 median, and 0.00 75th percentile). The WER of a cutout is the average WER of the WERs of the sentences of the cutout. The WERs aggregated by participant are depicted in Figure 6.5 and aggregated by cutout in Figure 6.6.

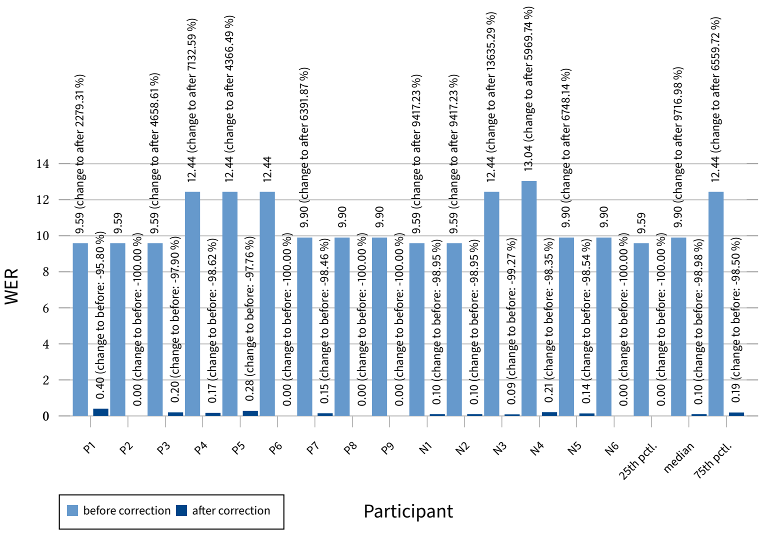


Figure 6.5.: Average word error rate (first averaged per sentence within each cutout) (aggregated by participant), change to after correction not specified if after correction has a WER of 0

Before the correction, the character error rate (CER) is on average per cutout 7.77 (4.78 25th percentile, 8.17 median, and 9.87 75th percentile) and after the correction 0.06 (0.00 25th percentile, 0.00 median, and 0.00 75th percentile). The CER of a cutout is the average CER of the CERs of the sentences of the cutout. The CERs aggregated by participant are depicted in Figure 6.7 and aggregated by cutout in Figure 6.8.

The error correction should not be only effective but also efficient. The advantage of the interactive error correction is that users can correct multiple times, but the number of required selections and corrections should be as minimal as possible.

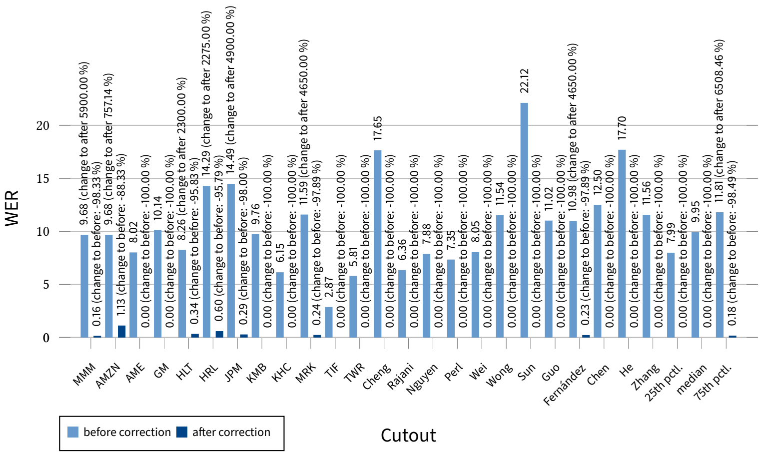


Figure 6.6.: Averaged word error rate (first averaged per sentence within each cutout) (aggregated by cutout), change to after correction not specified if after correction has a WER of 0

Mostly, the incremental processing of subsentences was not used. Only 0.41 % of all sentences were subsentences in a cutout on average. To not increase the latency, a subsentence should only be processed if there is a speech pause longer than 1.5 seconds and that was not the case most of the time. Therefore, we only consider full sentences for the following metrics.

The metrics can be subdivided into two categories. The first category is the actual time. For the actual time not only the error correction component but other components like the ASR system and the latency of the Lecture Translator framework with its network transfers play a role. The second category are absolute numbers. An absolute number is independent of the actual time needed of a system. It can be assumed that the speed of the systems will increase even if Moore's Law cannot be guaranteed forever. That means the actual times could be accelerated by technological progress, but the absolute numbers will stay the same.

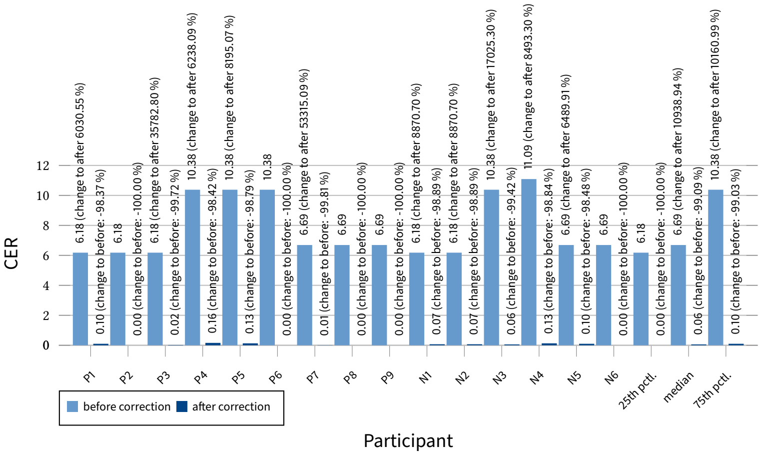


Figure 6.7.: Averaged cutout character error rate (first averaged per sentence within each cutout) (aggregated by participant), change to after correction not specified if after correction has a CER of 0

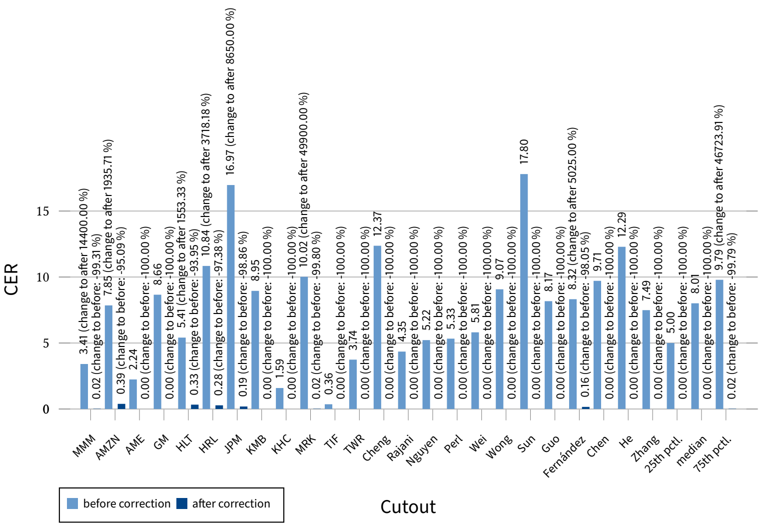


Figure 6.8.: Averaged character error rate (first averaged per sentence within each cutout) (aggregated by cutout), change to after correction not specified if after correction has a CER of 0

On average, a cutout needed 7 minutes and 47 seconds to be corrected (4 minutes and 17 seconds in the 25th percentile, 6 minutes and 26 seconds minutes in the median, and 9 minutes and 16 seconds in the 75th percentile). The aggregated durations by participant are depicted in Figure 6.9 and by cutout in Figure 6.10. Of course, you cannot really compare the times because the tasks have a different number of errors and the errors have different difficulty levels.

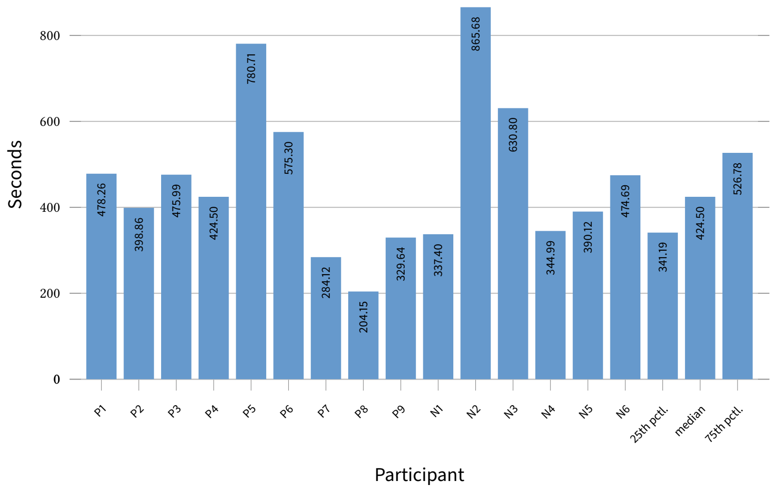


Figure 6.9.: Average time to process one cutout (aggregated by participant)

For a better comparison, the time per error is on average (first averaged per cutout) 40.63 seconds (22.79 seconds in the 25th percentile, 35.63 seconds in the median, and 49.21 seconds in the 75th percentile). The times aggregated by participant and by cutout are depicted in Figure 6.11 and 6.12, respectively.

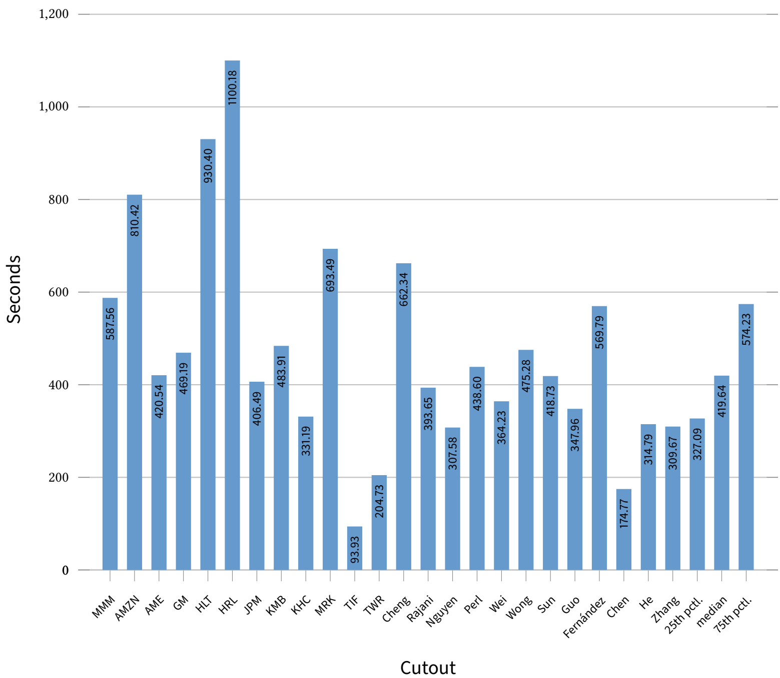


Figure 6.10.: Average time to process one cutout (aggregated by cutout)

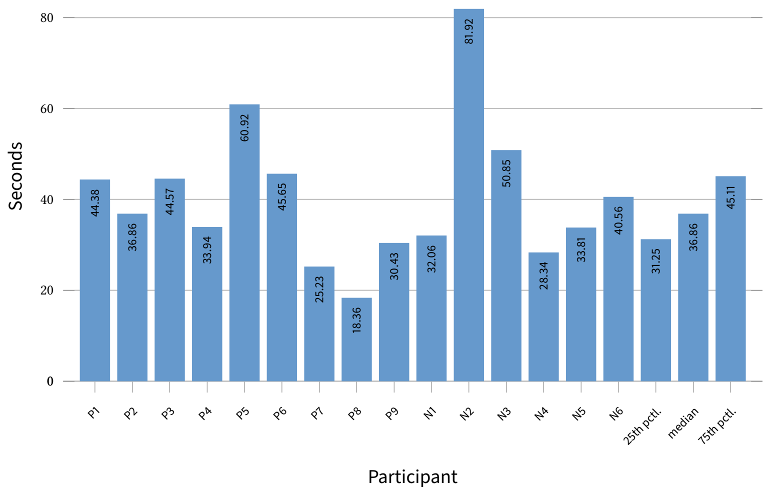


Figure 6.11.: Average time used at one error (first averaged per cutout) (aggregated by participant)

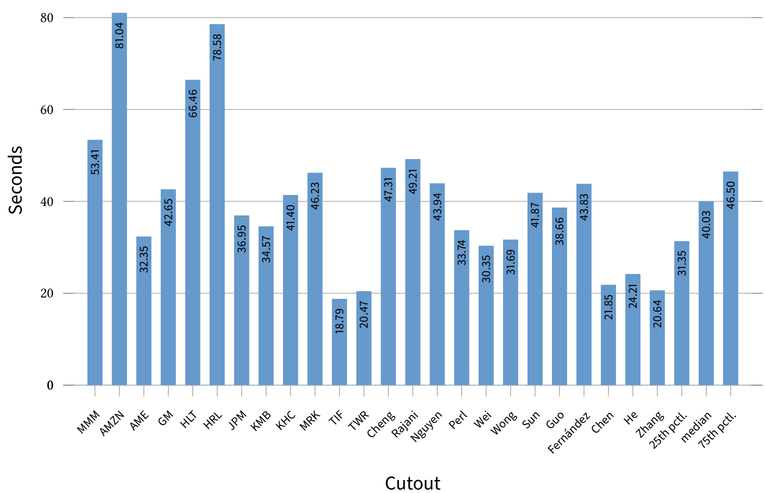


Figure 6.12.: Average time used at one error (first averaged per cutout) (aggregated by cutout)

Further interesting metrics are the time per selection and per one correction. On average (first averaged per cutout), 18.74 seconds are used at one selection (14.23 in the 25th percentile, 17.57 seconds in the median, and 21.83 seconds in the 75th percentile), the aggregated results by participant are in Figure 6.13 and by cutout in Figure 6.14.
On average (first averaged per cutout), 17.41 seconds are used at one correction (13.30 in the 25th percentile, 15.65 seconds in the median, and 18.69 seconds in the 75th percentile), the aggregated results by participant are in Figure 6.15 and by cutout in Figure 6.16.

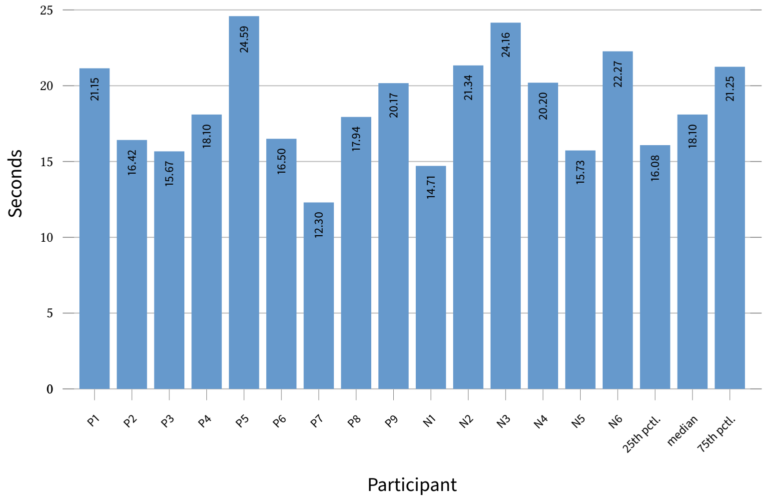


Figure 6.13.: Average time used at one selection (first averaged per cutout) (aggregated by participant)

In the following the absolute numbers and not the actual time efficient metrics are considered.

On average (first averaged per cutout), 2.21 selections per error were necessary (1.38 in the 25th percentile, 1.91 in the median, and 2.69 in the 75th percentile), the aggregated results by participant are in Figure 6.17 and by cutout in Figure 6.18.

On average (first averaged per cutout) 2.39 corrections per error were necessary (1.54 in the 25th percentile, 2.00 at the median, and 3.08 in the 75th

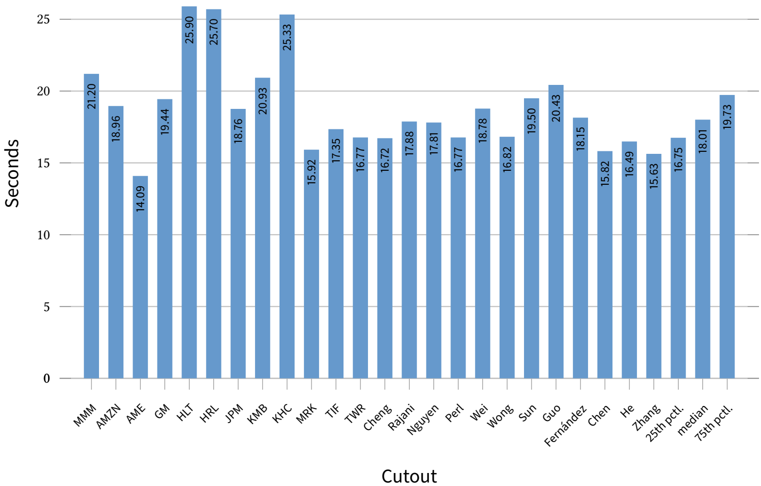


Figure 6.14.: Average time used at one selection (first averaged per cutout) (aggregated by cutout)

percentile), the aggregated results by participant are in Figure 6.19 and by cutout in Figure 6.20.

On average (first averaged per cutout), 61.11 % of the errors could be solved at the first attempt, 80.26 % in less or equal than two attempts, 88.35 % in less or equal than third attempts, for the remaining solved errors more attempts were necessary. More details are depicted in Figure 6.21, and Figure 6.22.

One problem is that the ASR transcripts of the corrections often contain errors. Corrections are spontaneous speech which is difficult [146, 175].

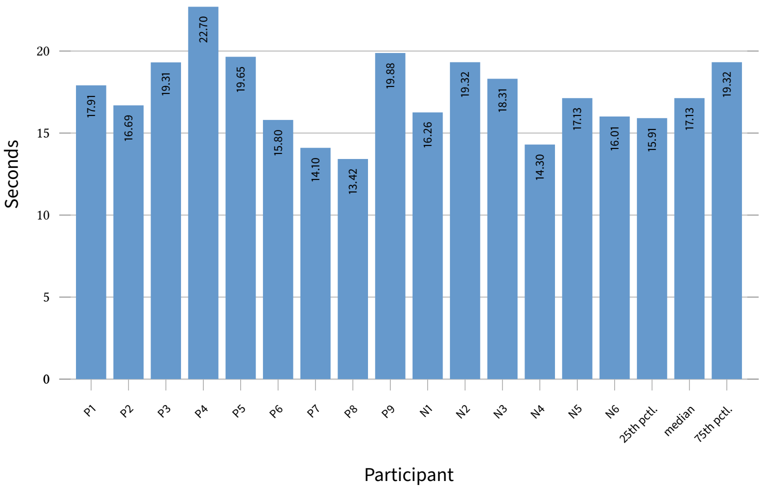


Figure 6.15.: Average time used for one correction (first averaged per cutout) (aggregated by participant)

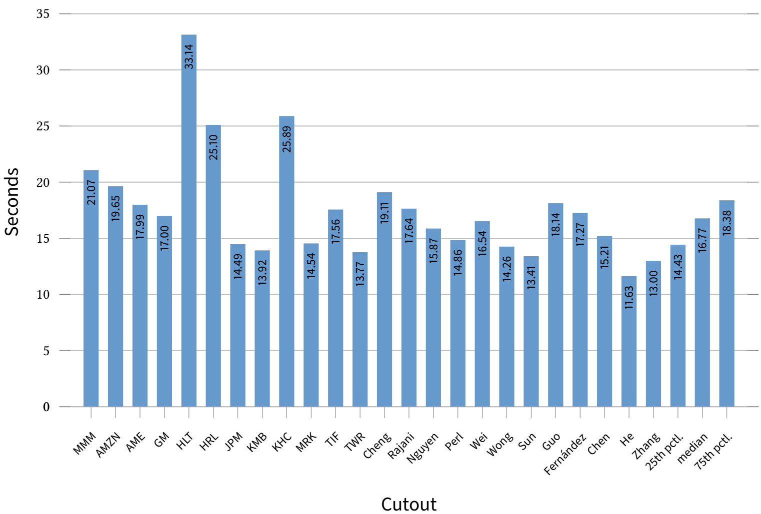


Figure 6.16.: Average time used for one correction (first averaged per cutout) (aggregated by cutout)

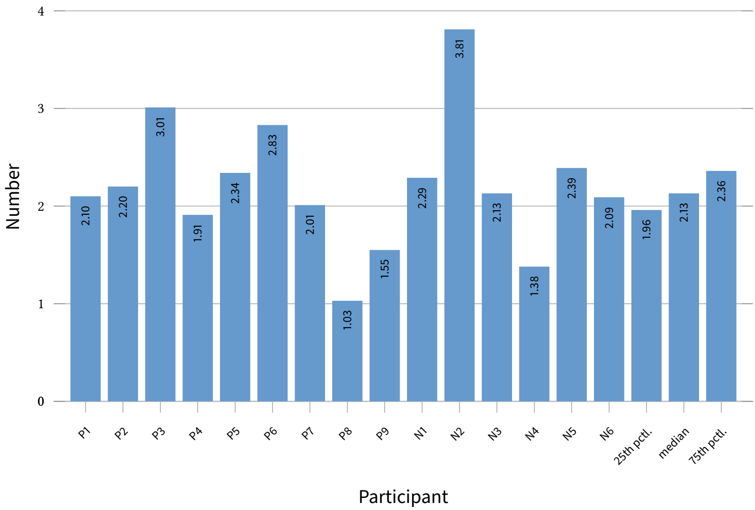


Figure 6.17.: Average selections per error (first averaged per cutout) (aggregated by participant)

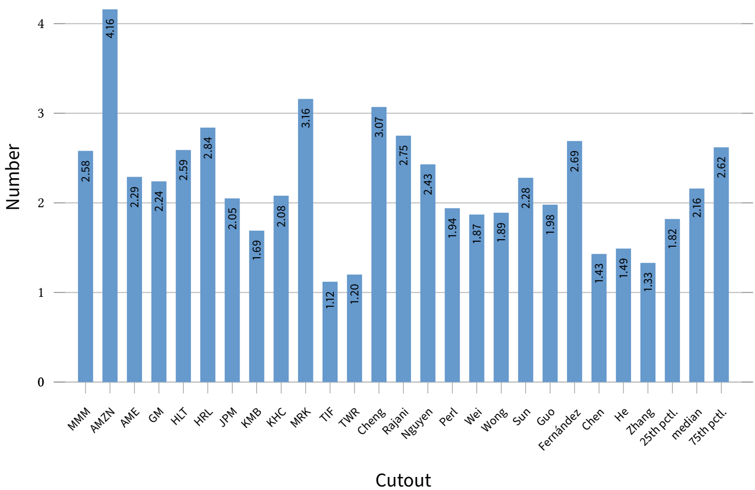


Figure 6.18.: Average selections per error (first averaged per cutout) (aggregated by cutout)

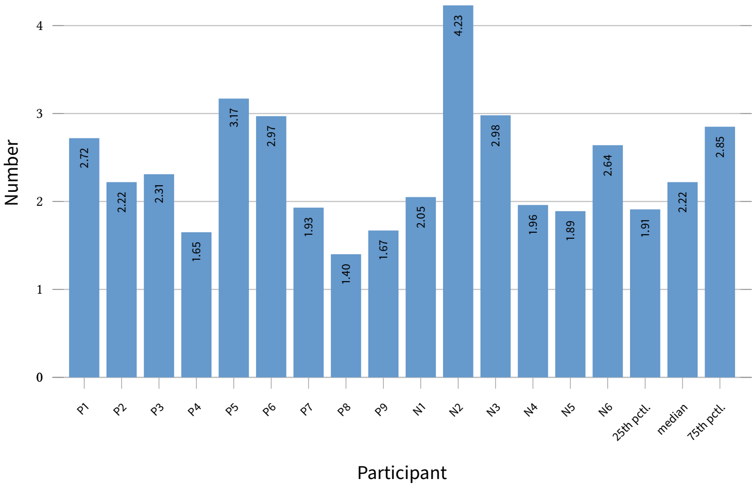


Figure 6.19.: Average corrections per error (first averaged per cutout) (aggregated by participant)

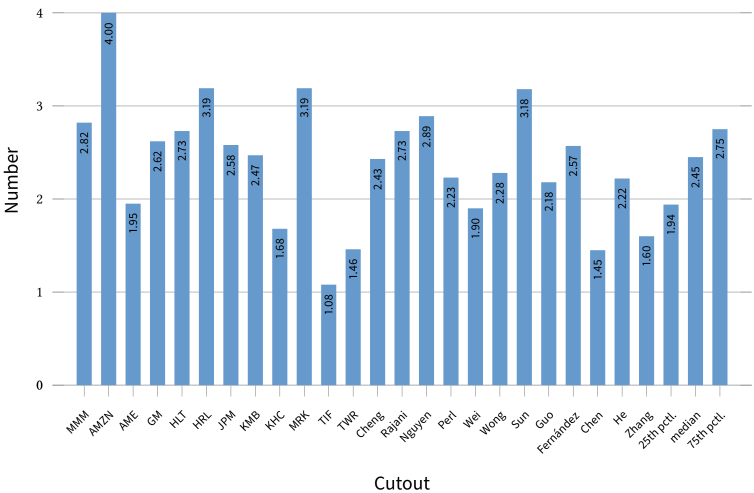


Figure 6.20.: Average corrections per error (first averaged per cutout) (aggregated by cutout)

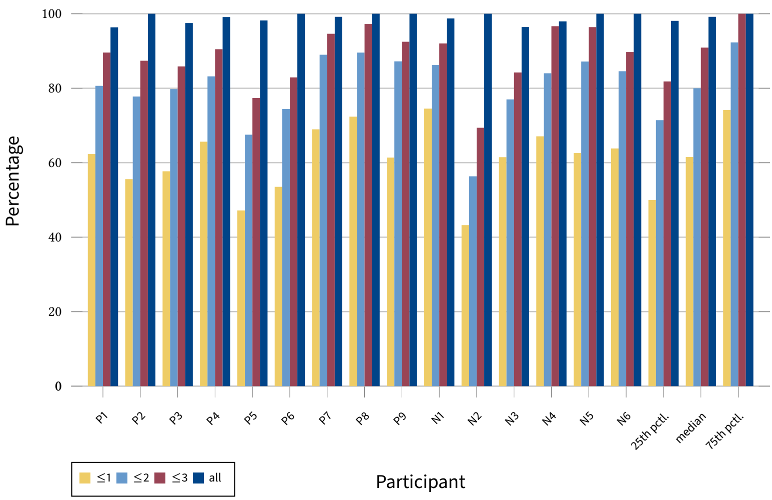


Figure 6.21.: Average cutout percentage of successful corrections w. r. t. tries (aggregated by participant)

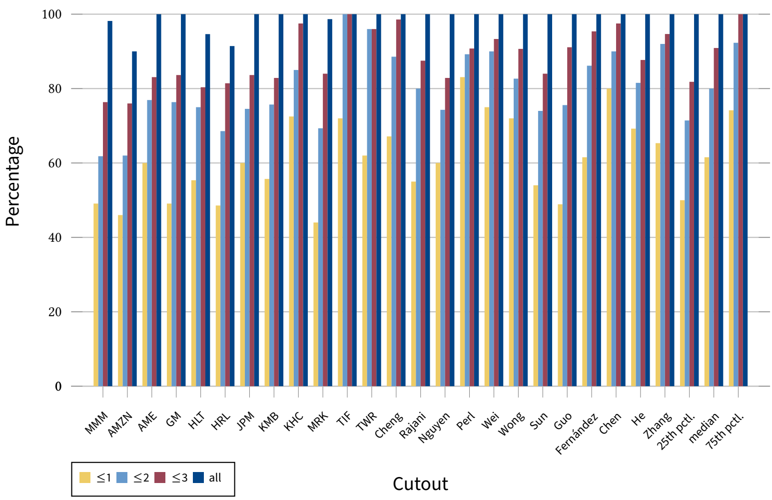


Figure 6.22.: Average cutout percentage of successful corrections w. r. t. tries (aggregated by cutout)

6.7. Unevaluated System Features

The evaluated unconstrained error correction system was limited to English language input. Tests showed that GPT-4o could handle other languages (German and French were tested). There are two possibilities to insert multilingual corrections: Without or with giving the language ID (LID) [122, 123] to the LLM. An LID was needed for monolingual ASR systems to select which parts of the speech are transcribed by which monolingual system. Every monolingual system is trained individually, but there are techniques to transfer knowledge if not enough training data is available [124]. However, modern multilingual ASR systems do not need this LID [62, 151]. It was decided to omit the LID for the correction system as well, because it already showed good results without LID. A combination of different languages for one correction is also possible. Among others, this feature can be helpful for nonnative speakers, for example, when the pronunciation is incorrect and leads to errors in the transcript. In this case, it can be more efficient for nonnative speakers to use their mother tongue for certain parts of the correction like spelling a word with the pronunciation of the alphabet letters in their mother tongue.

Another unevaluated feature added to the system is the detection of hyperarticulated speech. The system used to detect hyperarticulation on token level is in origin an ASR system with two outputs: the transcript and which tokens are hyperarticulated [5]. Because newer systems have a lower word error rate only the hyperarticulation detection is used. To combine a modern ASR system with the hyperarticulation detection, teacher forcing was used. After every token output of the hyperarticulation system, the output token is replaced by the output token of the used ASR system.

6.8. Conclusion

Even if the evaluation results are not so much better than the results of the fine-tuned model, qualitative tests showed the variety of possible natural language corrections is much higher. A reason for that is that the training and test datasets were collected through the same web applications and are more similar. On the collected data, often easy techniques like redictation works. Only in very difficult words where redictation is not possible, the LLM approach has its strenghts.

To test the system in a real-word scenario, it was integrated in the Lecture Translator framework and evaluated with 15 participants. Although on average only 61.11 % of the errors of a cutout could be corrected by the participants on the first try, after up to three tries on average 88.35 % of the errors of a cutout could be corrected and in total on average 98.91 % of the errors of a cutout could be corrected. On average 2.39 tries (median 2.00) were necessary to correct an error of an cutout. The performance on the first try is comparable to the test dataset where 65.32 % of the sentences could be solved.

7. Discussion

7.1. Conclusion

The goal of this thesis is to offer an unconstrained interactive error correction. Such unconstrained error correction was a big research gap in the existing literature. Recently, a first system that aims in this direction was developed [84], but the authors state that their system is not production-ready, because it is not accurate enough and the better model of their two presented models is even too slow. In this thesis, an accurate and fast enough to be used system is presented. The effectiveness and the efficiency were tested with 15 participants. The participants could correct on average 98.91 % of the errors of a cutout and were thus able to prove the effectiveness of the system. Although on average only 61.11 % of the errors of a cutout could be corrected on the first try, the interactiveness enabled the participants to correct on average 88.35 % of the errors of a cutout after up to three tries and in the end 98.91 % of the errors could be corrected. On average (first averaged per cutout), 2.39 tries (median 2.00) were necessary to correct an error. On first sight, the results seem a little bit inefficient, but often redictation is not working and the users have to find strategies to correct the errors and the unconstrained error correction only makes it possible to use other error correction strategies. Earlier systems used multimodality like handwriting [144] as a different strategy which however would not be possible in certain situations like while driving a car when users can only use speech as modality. The system showed that operating by speech is enough to correct on average 98.91 % of the errors of a cutout. The correction history is logged so that a life-long learning component can learn from the corrections.

Multimodality is not necessary to correct texts anymore but can still be used to go beyond texts. In this thesis, a system is presented that allows to correct an entity that are falsely selected by a robot by natural language and pointing at the correct entities. In addition, the initial used name of the entity and the image of the corrected entity are extracted so that a life-long learning component can learn from them. This system relies on the output of a region proposal component that finds regions of interest and a pointing line generation component that eliminates the regions of interest outside of the pointing line. In an end-to-end evaluation that evaluates all components, on the first try, the system could output the initial used name and the cutout of correct entity in 41.53 % of the 443 test dataset elements. The interactive system increased it to 46.73 % after three tries. However, only in 42.42 % of the 364 elements that contain a correction, the reference box was in the proposed regions of interest. Assuming to have a perfect pointing line generation and region proposal component and considering only elements where the reference box (if existing) was in the proposed regions of interest, the performance increased to 78.63 % for one try and 88.46 % for up to three tries.

7.2. Future Work

There is still a lot of work to do in the domain of error correction. In the following some ideas are presented.

To optimize the time for the finalization of the automatic speech recognition output is the most important measure to accelerate the unconstrained error correction system. If this is not possible, a good solution could be to use preliminary outputs and if one output changes, roll back the correction and correct with the changed output. This would accelerate the cases where the output needs some time for finalization but does not change in this time. The disadvantage of this approach is that it increases the number of requests to the large language model (LLM). During the development of the system this was not possible because of the rate limits of the GPT-4o model. The rate limits were increased over time, but every request still needs resources.

The presented multimodal error correction system still has the constraint that only entities can be changed. Based on the learnings of the linguistic unconstrained error correction system, the system could be extended to offer corrections of all types of errors.

The results of the multimodal error correction component are very promising for elements where the reference bounding box is included in the proposed regions of interest, but the dataset is from the year 2022 and a lot of included objects are even objects that are known long before the year 2022. To test the ability of correcting and extracting entities that are completely new like new characters from movies or new device names, a dataset with such new entities should be created.

Using a better region proposal component like [74] could improve the results for the multimodal error correction system in the end-to-end setting.

As pre-trained models T5 [113] and VL-T5 [15] are used in this thesis. Other models like BART [83] and VL-BART [15] could also be evaluated to show whether the models differ much in the results.
The same applies for the LLMs. Besides GPT-4o, there are so many LLMs like Gemini 2.5 [71] or DeepSeek V3 [30].

Often ASR system are not certain about individual words and maybe the correct word is in the n-best list, the performance of the error correction system could be improved by using not only transcripts but lattices [131] that have alternatives included.

A very big untackled research gap is to select automatically erroneous parts with a good quality. There are systems which tried confidence measurements, but the performance was much worse than manual selection [144]. Real errors are not found and correct words are selected. It requires really good confidence measurements and a mechanism to skip false positives efficiently.

Bibliography

[1] Rashad Albo Jamara, Nico Herbig, Antonio Krüger, and Josef van Genabith. "Mid-Air Hand Gestures for Post-Editing of Machine Translation." In: *Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing (ACL-IJCNLP)*. 2021. DOI: 10.18653/v1/2021.acl-long.527.

[2] Shunichi Amari. "A Theory of Adaptive Pattern Classifiers." In: *IEEE Transactions on Electronic Computers* EC-16.3 (1967). DOI: 10.1109/PGEC.1967.264666.

[3] Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. "Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering." In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018. DOI: 10.1109/CVPR.2018.00636.

[4] Tamim Asfour, Lukas Kaul, Mirko Wächter, Simon Ottenhaus, Pascal Weiner, Samuel Rader, Raphael Grimm, You Zhou, Markus Grotz, Fabian Paus, Dmitriy Shingarey, and Hans Haubert. "ARMAR-6: A Collaborative Humanoid Robot for Industrial Environments." In: *International Conference on Humanoid Robots (Humanoids)*. 2018. DOI: 10.1109/HUMANOIDS.2018.8624966.

[5] Maximilian Awiszus. "ASR System with hyperarticulated token detection (unpublished)."

[6] Pablo Azagra, Javier Civera, and Ana C. Murillo. "Incremental Learning of Object Models From Natural Human–Robot Interactions." In: *IEEE Transactions on Automation Science and Engineering* 17.4 (2020). DOI: 10.1109/TASE.2020.2980246.

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate." In: *International Conference on Learning Representations (ICLR)*. 2015. DOI: 10.48550/arXiv.1409.0473.

[8] Leonard Bärmann, Rainer Kartmann, Fabian Peller-Konrad, Jan Niehues, Alex Waibel, and Tamim Asfour. "Incremental learning of humanoid robot behavior from natural interaction and large language models." In: *Frontiers in Robotics and AI* 11 (2024). DOI: 10.3389/frobt.2024.1455375.

[9] T. R. Beelders and P. J. Blignaut. "Using vision and voice to create a multimodal interface for Microsoft Word 2007." In: *Symposium on Eye-Tracking Research & Applications (ETRA)*. 2010. DOI: 10.1145/1743666.1743709.

[10] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Inc., 2009. ISBN: 0596516495.

[11] Elizabeth R. Blackmer and Janet L. Mitton. "Theories of monitoring and the timing of repairs in spontaneous speech." In: *Cognition* 39.3 (1991). DOI: 10.1016/0010-0277(91)90052-6.

[12] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. "Language Models are Few-Shot Learners." In: *International Conference on Neural Information Processing Systems (NeurIPS)*. 2020. URL: https://papers.nips.cc/paper_files/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[13] Udo Bub, Martin Hunke, and Alex Waibel. "Knowing who to listen to in speech recognition: visually guided beamforming." In: *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 1995. DOI: 10.1109/ICASSP.1995.479827.

[14] Rich Caruana. "Multitask Learning." In: *Machine Learning* 28 (1997). DOI: 10.1023/A:1007379606734.

[15] Jaemin Cho, Jie Lei, Hao Tan, and Mohit Bansal. "Unifying Vision-and-Language Tasks via Text Generation." In: *International Conference on Machine Learning (ICML)*. 2021. URL: https://proceedings.mlr.press/v139/cho21a.

[16] Stefan Constantin. *slate-transcript-editor-wrapper*. https://github.com/msc42/slate-transcript-editor-wrapper/tree/feat/commandclips. 2022.

[17] Stefan Constantin, Fevziye Irem Eyiokur, Dogucan Yaman, Leonard Bärmann, and Alex Waibel. "Interactive Multimodal Robot Dialog Using Pointing Gesture Recognition." In: *European Conference on Computer Vision (ECCV) Workshops*. 2022. DOI: 10.1007/978-3-031-25075-0_43.

[18] Stefan Constantin, Fevziye Irem Eyiokur, Dogucan Yaman, Leonard Bärmann, and Alex Waibel. "Multimodal Error Correction with Natural Language and Pointing Gestures." In: *International Conference on Computer Vision (ICCV) Workshops*. © 2023 IEEE. 2023. DOI: 10.1109/ICCVW60793.2023.00212.

[19] Stefan Constantin, Jan Niehues, and Alex Waibel. "An End-to-End Goal-Oriented Dialog System with a Generative Natural Language Response Generation." In: *International Workshop on Spoken Dialogue System Technology (IWSDS)*. 2018. DOI: 10.1007/978-981-13-9443-0_18.

[20] Stefan Constantin, Jan Niehues, and Alex Waibel. "Incremental processing of noisy user utterances in the spoken language understanding task." In: *Workshop on Noisy User-generated Text (W-NUT) at the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Hong Kong, China: Association for Computational Linguistics, 2019. DOI: 10.18653/v1/D19-5535.

[21] Stefan Constantin, Jan Niehues, and Alex Waibel. "Multi-task learning to improve natural language understanding." In: *International Workshop on Spoken Dialogue System Technology (IWSDS)*. 2019. DOI: 10.48550/arXiv.1812.06876.

[22] Stefan Constantin and Alex Waibel. "Comparison of Error Correction and Extraction Approaches." In: *Practical Solutions for Diverse Real-World NLP Applications*. Signals and Communication Technology (SCT). Reproduced with permission from Springer Nature. 2024. DOI: 10.1007/978-3-031-44260-5_5.

[23] Stefan Constantin and Alex Waibel. "Error correction and extraction in request dialogs." In: *International Conference on Natural Language and Speech Processing (ICNLSP)*. Copyright © 2022 Association for Computational Linguists (ACL). All Rights Reserved. 2022.

[24] PyTorch Serve contributors. *Serve*. https://github.com/pytorch/serve. 2023.

[25] British Broadcasting Corporation. *React Transcript Editor*. https://github.com/bbc/react-transcript-editor. 2025.

[26] Mozilla Corporation. *WebVTT API*. https://developer.mozilla.org/en-US/docs/Web/API/WebVTT_API. 2025.

[27] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Sanja Fidler, Antonino Furnari, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, and Michael Wray. "The EPIC-KITCHENS Dataset: Collection, Challenges and Baselines." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 43.11 (2021). DOI: 10.1109/TPAMI.2020.2991965.

[28] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Antonino Furnari, Evangelos Kazakos, Jian Ma, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, and Michael Wray. "Rescaling Egocentric Vision: Collection, Pipeline and Challenges for EPIC-KITCHENS-100." In: *International Journal of Computer Vision (IJCV)* 130 (2022). DOI: 10.1007/s11263-021-01531-2.

[29] Catalina Danis, Liam Comerford, Eric Janke, Ken Davies, Jackie De Vries, and Alex Bertrand. "Storywriter: a speech oriented editor." In: *Conference on Human Factors in Computing Systems (CHI)*. 1994. DOI: 10.1145/259963.260490.

[30] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, et al. *DeepSeek-V3 Technical Report*. 2025. DOI: 10.48550/arXiv.2412.19437. arXiv: 2412.19437 [cs.CL].

[31] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." In: *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*. 2019. DOI: 10.18653/v1/N19-1423.

[32] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale." In: *International Conference on Learning Representations (ICLR)*. 2021. URL: https://openreview.net/forum?id=YicbFdNTTy.

[33] Matthias Eck, Stephan Vogel, and Alex Waibel. "Low Cost Portability for Statistical Machine Translation based on N-gram Frequency and TF-IDF." In: *International Workshop on Spoken Language Translation (IWSLT)*. 2005. URL: https://aclanthology.org/2005.iwslt-1.7.

[34] Jeffrey L. Elman. "Finding Structure in Time." In: *Cognitive Science* 14.2 (1990). DOI: 10.1207/s15516709cog1402_1.

[35] Jiayue Fan, Chenning Xu, Chun Yu, and Yuanchun Shi. "Just Speak It: Minimize Cognitive Load for Eyes-Free Text Editing with a Smart Voice Assistant." In: *Annual ACM Symposium on User Interface Software and Technology (UIST)*. 2021. DOI: 10.1145/3472749.3474795.

[36] W. Nelson Francis and Henry Kucera. *Brown Corpus Manual*. Tech. rep. Department of Linguistics, Brown University, Providence, Rhode Island, US, 1979. URL: https://listings.lib.msu.edu/public-corpora/cd421/manuals/brown/INDEX.HTM.

[37] Frank Wang et al. *pyTelegramBotAPI*. https://github.com/eternnoir/pyTelegramBotAPI. 2025.

[38] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. "Learning to forget: continual prediction with LSTM." In: *International Conference on Artificial Neural Networks (ICANN)*. 1999. DOI: 10.1049/cp:19991218.

[39] Debjyoti Ghosh, Pin Sym Foong, Shengdong Zhao, Can Liu, Nuwan Janaka, and Vinitha Erusu. "EYEditor: Towards On-the-Go Heads-Up Text Editing Using Voice and Manual Input." In: *Conference on Human Factors in Computing Systems (CHI)*. 2020. DOI: 10.1145/3313831.3376173.

[40] Debjyoti Ghosh, Can Liu, Shengdong Zhao, and Kotaro Hara. "Commanding and Re-Dictation: Developing Eyes-Free Voice-Based Interaction for Editing Dictated Text." In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 27.4 (2020). DOI: 10.1145/3390889.

[41] Petra Gieselmann. "Comparing Error-Handling Strategies in Human-Human and Human-Robot Dialogues." In: *KONVENS (Konferenz zur Verarbeitung natürlicher Sprache)*. 2006.

[42] Petra Gieselmann. "Fehlerbehandlung in Mensch-Maschine-Dialogen." PhD thesis. Universität Stuttgart, 2007. DOI: 10.18419/opus-2606.

[43] Petra Gieselmann and Alex Waibel. "What Makes Human-Robot Dialogues Struggle?" In: *Workshop on the Semantics and Pragmatics of Dialogue (SemDial)*. 2005.

[44] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. "On calibration of modern neural networks." In: *International Conference on Machine Learning (ICML)*. 2017. URL: https://proceedings.mlr.press/v70/guo17a.html.

[45] Nico Herbig, Tim Düwel, Santanu Pal, Kalliopi Meladaki, Mahsa Monshizadeh, Antonio Krüger, and Josef van Genabith. "MMPE: A Multi-Modal Interface for Post-Editing Machine Translation." In: *Annual Meeting of the Association for Computational Linguistics (ACL)*. 2020. DOI: 10.18653/v1/2020.acl-main.155.

[46] Nico Herbig, Santanu Pal, Tim Düwel, Kalliopi Meladaki, Mahsa Monshizadeh, Vladislav Hnatovskiy, Antonio Krüger, and Josef van Genabith. "MMPE: A Multi-Modal Interface using Handwriting, Touch Reordering, and Speech Commands for Post-Editing Machine Translation." In: *Annual Meeting of the Association for Computational Linguistics (ACL): System Demonstrations*. 2020. DOI: 10.18653/v1/2020.acl-demos.37.

[47] Nico Herbig, Santanu Pal, Tim Düwel, Raksha Shenoy, Antonio Krüger, and Josef van Genabith. "Improving the Multi-Modal Post-Editing (MMPE) CAT Environment based on Professional Translators' Feedback." In: *Workshop on Post-Editing in Modern-Day Translation (PEMDT) at the Conference of the Association for Machine Translation in the Americas (AMTA)*. 2020. URL: https://aclanthology.org/2020.amta-pemdt.7.

[48] Nico Herbig, Santanu Pal, Josef van Genabith, and Antonio Krüger. "Multi-Modal Approaches for Post-Editing Machine Translation." In: *Conference on Human Factors in Computing Systems (CHI)*. 2019. DOI: 10.1145/3290605.3300461.

[49] Nico Herbig, Santanu Pal, Mihaela Vela, Antonio Krüger, and Josef van Genabith. "Multi-modal indicators for estimating perceived cognitive load in post-editing of machine translation." In: *Machine Translation* 33 (2019). DOI: 10.1007/s10590-019-09227-8.

[50] Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. "Teaching Machines to Read and Comprehend." In: *International Conference on Neural Information Processing Systems (NeurIPS)*. 2015. URL: https://papers.nips.cc/paper_files/paper/2015/hash/afdec7005cc9f14302cd0474fd0f3c96-Abstract.html.

[51] Donald Hindle. "Deterministic Parsing of Syntactic Non-fluencies." In: *Annual Meeting of the Association for Computational Linguistics (ACL)*. 1983. DOI: 10.3115/981311.981336.

[52] Moto Hira. *Forced Alignment with Wav2Vec2*. https://pytorch.org/audio/stable/tutorials/forced_alignment_tutorial.html. 2023.

[53] Jacob B. Hirsh and Michael Inzlicht. "Error-related negativity predicts academic performance." In: *Psychophysiology* 47 (2010). DOI: 10.1111/j.1469-8986.2009.00877.x.

[54] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory." In: *Neural Computation* 9.8 (1997). DOI: 10.1162/neco.1997.9.8.1735.

[55] Hartwig Holzapfel and Petra Gieselmann. "A Way Out of Dead End Situations in Dialogue Systems for Human-Robot Interaction." In: *International Conference on Humanoid Robots (Humanoids)*. 2004. DOI: 10.1109/ICHR.2004.1442122.

[56] Hartwig Holzapfel, Daniel Neubig, and Alex Waibel. "A dialogue approach to learning object descriptions and semantic categories." In: *Robotics and Autonomous Systems* 56.11 (2008). DOI: 10.1016/j.robot.2008.08.012.

[57] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. *spaCy: Industrial-strength Natural Language Processing in Python*. 2023. DOI: 10.5281/zenodo.1212303.

[58] Christian Huber, Tu Anh Dinh, Carlos Mullov, Ngoc-Quan Pham, Thai Binh Nguyen, Fabian Retkowski, Stefan Constantin, Enes Ugan, Danni Liu, Zhaolin Li, Sai Koneru, Jan Niehues, and Alexander Waibel. "End-to-End Evaluation for Low-Latency Simultaneous Speech Translation." In: *Conference on Empirical Methods in Natural Language Processing (EMNLP): System Demonstrations*. 2023. DOI: 10.18653/v1/2023.emnlp-demo.2.

[59] Christian Huber, Juan Hussain, Tuan-Nam Nguyen, Kaihang Song, Sebastian Stüker, and Alexander Waibel. "Supervised Adaptation of Sequence-to-Sequence Speech Recognition Systems using Batch-Weighting." In: *Workshop on Life-long Learning for Spoken Language Systems (LifeLongNLP) at the Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing (AACL-IJCNLP)*. 2020. DOI: 10.18653/v1/2020.lifelongnlp-1.2.

[60] Christian Huber, Juan Hussain, Sebastian Stüker, and Alexander Waibel. "Instant One-Shot Word-Learning for Context-Specific Neural Sequence-to-Sequence Speech Recognition." In: *Automatic Speech Recognition and Understanding Workshop (ASRU)*. 2021. DOI: 10.1109/ASRU51503.2021.9687898.

[61] Christian Huber, Rishu Kumar, Ondřej Bojar, and Alexander Waibel. *Short-Term Word-Learning in a Dynamically Changing Environment*. 2022. DOI: 10.48550/arXiv.2203.15404. arXiv: 2203.15404 [cs.CL].

[62] Christian Huber, Enes Yavuz Ugan, and Alexander Waibel. *Code-Switching without Switching: Language Agnostic End-to-End Speech Translation*. 2022. DOI: 10.48550/arXiv.2210.01512. arXiv: 2210.01512 [cs.CL].

[63] Christian Huber and Alexander Waibel. "Continuously Learning New Words in Automatic Speech Recognition." In: *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2025. DOI: 10.1109/ICASSP49660.2025.10889216.

[64] Wolfgang Huerst, Jie Yang, and Alex Waibel. "Interactive Error Repair for an Online Handwriting Interface." In: *Conference on Human Factors in Computing Systems (CHI)*. 1998. DOI: 10.1145/286498.286818.

[65] Wolfgang Hürst, Jie Yang, and Alex Waibel. "Error Repair in Human Handwriting - An Intelligent User Interface for Automatic On-Line Handwriting Recognition." In: *International Joint Symposia on Intelligence and Systems (IJSIS)*. 1998. DOI: 10.1109/IJSIS.1998.685482.

[66] Luca Hüttner, Dominik, and Jannik Weiß. *slate-transcript-editor-wrapper*. https://github.com/TEQST/slate-transcript-editor-wrapper. 2025.

[67] JJRobots. *Remotely controlled LASER POINTER ROBOT*. https://web.archive.org/web/20230921222221/https://jjrobots.com/remotely-controlled-laser-pointer.

[68] Glenn Jocher, Alex Stoken, Jirka Borovec, NanoCode012, ChristopherSTAN, Liu Changyu, Laughing, tkianai, Adam Hogan, lorenzomammana, yxNONG, AlexWang1900, Laurentiu Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, Francisco Ingham, Frederik, Guilhen, Hatovix, Jake Poznanski, Jiacong Fang, Lijun Yu, changyu98, Mingyu Wang, Naman Gupta, Osama Akhtar, PetrDvoracek, and Prashant Rai. *ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements*. Version v3.1. 2020. DOI: 10.5281/zenodo.4154370.

[69] Michael I. Jordan. *Serial Order: A Parallel Distributed Processing Approach*. Tech. rep. ICS Report 8604. Institute for Cognitive Science, University of California, San Diego, 1986.

[70] Clare-Marie Karat, Christine Halverson, Daniel Horn, and John Karat. "Patterns of Entry and Correction in Large Vocabulary Continuous Speech Recognition Systems." In: *Conference on Human Factors in Computing Systems (CHI)*. 1999. DOI: 10.1145/302979.303160.

[71] Koray Kavukcuoglu. *Gemini 2.5: Our most intelligent AI model*. https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025. 2025.

[72] J. F. Kelley. "An empirical methodology for writing User-Friendly Natural Language computer applications." In: *Conference on Human Factors in Computing Systems (CHI)*. 1983. DOI: 10.1145/800045.801609.

[73] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In: *International Conference on Learning Representations (ICLR)*. 2015. DOI: 10.48550/arXiv.1412.6980.

[74] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. "Segment Anything." In: *International Conference on Computer Vision (ICCV)*. 2023. DOI: 10.1109/ICCV51070.2023.00371.

[75] Ivan Kraljevski and Diane Hirschfeld. "Hyperarticulation of Corrections in Multilingual Dialogue Systems." In: *Annual Meeting of the International Speech Communication Association (Interspeech)*. 2017. DOI: 10.21437/Interspeech.2017-753.

[76] Taku Kudo and John Richardson. "SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing." In: *Conference on Empirical Methods in Natural Language Processing (EMNLP): System Demonstrations*. 2018. DOI: 10.18653/v1/D18-2012.

[77] Ludwig Kürzinger, Dominik Winkelbauer, Lujun Li, Tobias Watzel, and Gerhard Rigoll. "CTC-Segmentation of Large Corpora for German End-to-End Speech Recognition." In: *International Conference on Speech and Computer (SPECOM)*. 2020. DOI: 10.1007/978-3-030-60276-5_27.

[78] William Labov. "On the Grammaticality of Every-Day Speech." In: *Annual meeting of the Linguistic Society of America (LSA)*. 1966. URL: https://www.lsadc.org/files/Ling_%20Society%20of%20America%201966.pdf.

[79] Kevin Larson and David Mowatt. "Speech Error Correction: The Story of the Alternates List." In: *International Journal of Speech Technology* 6 (2003). DOI: 10.1023/A:1022342732234.

[80] Stefan Leijnen and Fjodor van Veen. "The Neural Network Zoo." In: *Conference Theoretical Information Studies (TIS)*. 2019. DOI: 10.3390/proceedings2020047009.

[81] Willem J. M. Levelt. "Monitoring and self-repair in speech." In: *Cognition* 14.1 (1983). DOI: 10.1016/0010-0277(83)90026-4.

[82] Vladimir I. Levenshtein. "Binary Codes Capable of Correcting Deletions, Insertions and Reversals." In: *Soviet Physics-Doklady* 10.8 (1966).

[83] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension." In: *Annual Meeting of the Association for Computational Linguistics (ACL)*. 2020. DOI: `10.18653/v1/2020.acl-main.703`.

[84] Belinda Z. Li, Jason Eisner, Adam Pauls, and Sam Thomson. "Toward Interactive Dictation." In: *Annual Meeting of the Association for Computational Linguistics (ACL)*. 2023. DOI: `10.18653/v1/2023.acl-long.854`.

[85] Toby Jia-Jun Li, Jingya Chen, Haijun Xia, Tom M. Mitchell, and Brad A. Myers. "Multi-Modal Repairs of Conversational Breakdowns in Task-Oriented Dialogs." In: *Annual ACM Symposium on User Interface Software and Technology (UIST)*. 2020. DOI: `10.1145/3379337.3415820`.

[86] Zhizhong Li and Derek Hoiem. "Learning Without Forgetting." In: *European Conference on Computer Vision (ECCV)*. 2016. DOI: `10.1007/978-3-319-46493-0_37`.

[87] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. "Microsoft COCO: Common Objects in Context." In: *European Conference on Computer Vision (ECCV)*. 2014. DOI: `10.1007/978-3-319-10602-1_48`.

[88] Seppo Linnainmaa. "Taylor expansion of the accumulated rounding error." In: *BIT Numerical Mathematics* 16 (1976). DOI: `10.1007/BF01931367`.

[89] Diane Litman, Marc Swerts, and Julia Hirschberg. "Characterizing and Predicting Corrections in Spoken Dialogue Systems." In: *Computational Linguistics* 32.3 (2006). DOI: `10.1162/coli.2006.32.3.417`.

[90] Danqing Liu. "A Comparison of Zero/Few-shot Approaches to Natural Language Understanding." Karlsruhe Institute of Technology, 2023. URL: `https://isl.iar.kit.edu/english/3243_9779.php`.

[91] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. "What Makes Good In-Context Examples for GPT-3?" In: *Deep Learning Inside Out (DeeLIO): Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*. 2022. DOI: `10.18653/v1/2022.deelio-1.10`.

[92] Google LLC. *Dialogflow*. https://cloud.google.com/dialogflow/docs.

[93] Ilya Loshchilov and Frank Hutter. “Decoupled Weight Decay Regularization.” In: *International Conference on Learning Representations (ICLR)*. 2019. URL: https://openreview.net/forum?id=Bkg6RiCqY7.

[94] Warren S. McCulloch and Walter Pitts. “A Logical Calculus of Ideas Immanent in Nervous Activity.” In: *Bulletin of Mathematical Biophysics* 5 (1943). DOI: 10.1007/BF02478259.

[95] Arthur E. McNair and Alex Waibel. “Improving Recognizer Acceptance Through Robust, Natural Speech Repair.” In: *International Conference on Spoken Language Processing (ICSLP)*. 1994. DOI: 10.21437/ICSLP.1994-339.

[96] mphilli. *English to IPA (eng_to_ipa)*. https://github.com/mphilli/English-to-IPA. 2023.

[97] Christine Nakatani and Julia Hirschberg. “A speech-first model for repair detection and correction.” In: *Annual Meeting of the Association for Computational Linguistics (ACL)*. 1993. DOI: 10.3115/981574.981581.

[98] Jan Niehues, Ngoc-Quan Pham, Thanh-Le Ha, Matthias Sperber, and Alex Waibel. “Low-Latency Neural Speech Translation.” In: *Annual Meeting of the International Speech Communication Association (Interspeech)*. 2018. DOI: 10.21437/Interspeech.2018-1055.

[99] Curtis Ogle. *editops*. https://github.com/ctogle/editops. 2020.

[100] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, et al. *GPT-4 Technical Report*. 2024. DOI: 10.48550/arXiv.2303.08774. arXiv: 2303.08774 [cs.CL].

[101] OpenAI, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, Alex Paino, Alex Renzin, Alex Tachard Passos, Alexander

Kirillov, Alexi Christakis, Alexis Conneau, Ali Kamali, Allan Jabri, Allison Moyer, Allison Tam, Amadou Crookes, et al. *GPT-4o System Card.* 2024. DOI: 10.48550/arXiv.2410.21276. arXiv: 2410.21276 [cs.CL].

[102] OpenAI L.P. *openai-python.* https://github.com/openai/openai-python. 2023.

[103] Amit Kumar Pandey and Rodolphe Gelin. "A Mass-Produced Sociable Humanoid Robot: Pepper: The First Machine of Its Kind." In: *IEEE Robotics & Automation Magazine* 25.3 (2018). DOI: 10.1109/MRA.2018.2833157.

[104] Pietro Passarelli. *Slate transcript editor.* https://github.com/pietrop/slate-transcript-editor. 2025.

[105] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *International Conference on Neural Information Processing Systems (NeurIPS).* 2019. URL: https://papers.nips.cc/paper_files/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

[106] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. "Efficiently scaling transformer inference." In: *Annual Conference on Machine Learning and Systems (MLSys).* 2023. URL: https://proceedings.mlsys.org/paper_files/paper/2023/hash/c4be71ab8d24cdfb45e3d06dbfca2780-Abstract-mlsys2023.html.

[107] Matheus Vieira Portela and David Rozado. "Gaze Enhanced Speech Recognition for Truly Hands-Free and Efficient Text Input During HCI." In: *Australian Computer-Human Interaction (OzCHI) Conference.* 2014. DOI: 10.1145/2686612.2686679.

[108] Pallets Projects. *Jinja.* https://github.com/pallets/jinja. 2025.

[109] Yu Qin and Yi Yang. "What You Say and How You Say It Matters: Predicting Stock Volatility Using Verbal and Vocal Cues." In: *Annual Meeting of the Association for Computational Linguistics (ACL)*. 2019. DOI: 10.18653/v1/P19-1038.

[110] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. "Robust Speech Recognition via Large-Scale Weak Supervision." In: *International Conference on Machine Learning (ICML)*. 2023. URL: https://proceedings.mlr.press/v202/radford23a.html.

[111] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. "Improving Language Understanding by Generative Pre-Training." In: *OpenAI* (2018). URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

[112] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. "Language Models are Unsupervised Multitask Learners." In: *OpenAI* (2019). URL: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[113] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer." In: *Journal of Machine Learning Research* 21.140 (2020). URL: https://jmlr.org/papers/v21/20-074.html.

[114] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You Only Look Once: Unified, Real-Time Object Detection." In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016. DOI: 10.1109/CVPR.2016.91.

[115] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks." In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2019. DOI: 10.18653/v1/D19-1410.

[116] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster R-CNN: towards real-time object detection with region proposal networks." In: *International Conference on Neural Information Processing Systems (NeurIPS)*. 2015. URL: https://papers.nips.cc/paper_files/paper/2015/hash/14bfa6bb14875e45bba028a21ed38046-Abstract.html.

[117] James Robert. *Pydub*. https://github.com/jiaaro/pydub. 2022.

[118] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological Review* 65.6 (1958). DOI: 10.1037/h0042519.

[119] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." In: *nature* 323.6088 (1986). DOI: 10.1038/323533a0.

[120] Emanuel A. Schegloff, Gail D. Jefferson, and Harvey Sacks. "The preference for self-correction in the organization of repair in conversation." In: *Language* 53.2 (1977). DOI: 10.2307/413107.

[121] Bernt Schiele and Alex Waibel. "Gaze Tracking Based on Face-Color." In: *International Workshop on Automatic Face- and Gesture-Recognition (IWAFGR)*. 1995.

[122] Tanja Schultz, Ivica Rogina, and Alex Waibel. "Experiments With LVCSR Based Language Identification." In: *Annual Speech Research Symposium (SRS)*. 1995.

[123] Tanja Schultz, Ivica Rogina, and Alex Waibel. "LVCSR-based language identification." In: *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 1996. DOI: 10.1109/ICASSP.1996.543237.

[124] Tanja Schultz and Alex Waibel. "Experiments on cross-language acoustic modeling." In: *European Conference on Speech Communication and Technology (Eurospeech)*. 2001. DOI: 10.21437/Eurospeech.2001-636.

[125] Abigail See, Peter J. Liu, and Christopher D. Manning. "Get To The Point: Summarization with Pointer-Generator Networks." In: *Annual Meeting of the Association for Computational Linguistics (ACL)*. 2017. DOI: 10.18653/v1/P17-1099.

[126] Korok Sengupta, Sabin Bhattarai, Sayan Sarcar, I. Scott MacKenzie, and Steffen Staab. "Leveraging Error Correction in Voice-based Text Entry by Talk-and-Gaze." In: *Conference on Human Factors in Computing Systems (CHI)*. 2020. DOI: 10.1145/3313831.3376579.

[127] Elizabeth Shriberg, John Bear, and John Dowding. "Automatic Detection and Correction of Repairs in Human-Computer Dialog." In: *DARPA Speech and Natural Language Workshop*. 1992. URL: https://aclanthology.org/H92-1085.

[128] Elizabeth Ellen Shriberg. "Preliminaries to a Theory of Speech Disfluencies." PhD thesis. University of California, 1994.

[129] Hagen Soltau, Florian Metze, and Alex Waibel. "Compensating For Hyperarticulation By Modeling Articulatory Properties." In: *International Conference on Spoken Language Processing (ICSLP)*. 2002. DOI: 10.21437/ICSLP.2002-288.

[130] Hagen Soltau and Alex Waibel. "Specialized Acoustic Models For Hyperarticulated Speech." In: *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2000. DOI: 10.1109/ICASSP.2000.862098.

[131] Matthias Sperber, Graham Neubig, Jan Niehues, and Alex Waibel. "Neural Lattice-to-Sequence Models for Uncertain Inputs." In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2017. DOI: 10.18653/v1/D17-1145.

[132] Matthias Sperber, Jan Niehues, and Alex Waibel. "Toward Robust Neural Machine Translation for Noisy Input Sequences." In: *International Workshop on Spoken Language Translation (IWSLT)*. 2017. URL: https://aclanthology.org/2017.iwslt-1.13.

[133] Jörg Spilker, Martin Klarner, and Günther Görz. "Processing Self Corrections in a speech to speech system." In: *International Conference on Computational Linguistics (COLING)*. 2000. DOI: 10.3115/992730.992816.

[134] Rainer Stiefelhagen, Michael Finke, Jie Yang, and Alex Waibel. "From Gaze to Focus of Attention." In: *International Conference on Visual Information and Information Systems (VISUAL)*. 1999. DOI: 10.1007/3-540-48762-X_94.

[135] Rainer Stiefelhagen, Christian Fügen, Petra Gieselmann, Hartwig Holzapfel, Kai Nickel, and Alex Waibel. "Natural Human-Robot Interaction using Speech, Head Pose and Gestures." In: *International Conference on Intelligent Robots and Systems (IROS)*. Vol. 3. 2004. DOI: 10.1109/IROS.2004.1389771.

[136] Rainer Stiefelhagen, Jie Yang, and Alex Waibel. "Estimating Focus of Attention Based on Gaze and Sound." In: *Workshop on Perceptive User Interfaces (PUI)*. 2001. DOI: 10.1145/971478.971505.

[137] Rainer Stiefelhagen, Jie Yang, and Alex Waibel. “Modeling Focus of Attention for Meeting Indexing.” In: *International Conference on Multimedia (MM)*. 1999. DOI: 10.1145/319463.319464.

[138] Rainer Stiefelhagen, Jie Yang, and Alex Waibel. “Modeling focus of attention for meeting indexing based on multiple cues.” In: *IEEE Transactions on Neural Networks* 13.4 (2002). DOI: 10.1109/TNN.2002.1021893.

[139] Michael Stonebraker, Lawrence A. Rowe, and Michael Hirohama. “The Implementation of POSTGRES.” In: *IEEE Transactions on Knowledge and Data Engineering* 2.1 (1990). DOI: 10.1109/69.50912.

[140] Bernhard Suhm. “Empirical Evaluation of Interactive Multimodal Error Correction.” In: *Automatic Speech Recognition and Understanding Workshop (ASRU)*. 1997. DOI: 10.1109/ASRU.1997.659139.

[141] Bernhard Suhm, Brad Myers, and Alex Waibel. “Designing Interactive Error Recovery Methods for Speech Interfaces.” In: *Designing the User Interface for Speech Recognition Applications at the Conference on Human Factors in Computing Systems (CHI)*. 1996.

[142] Bernhard Suhm, Brad Myers, and Alex Waibel. “Interactive recovery from speech recognition errors in speech user interfaces.” In: *International Conference on Spoken Language Processing (ICSLP)*. 1996. DOI: 10.1109/ICSLP.1996.607738.

[143] Bernhard Suhm, Brad Myers, and Alex Waibel. “Model-based and Empirical Evaluation of Multimodal Interactive Error Correction.” In: *Conference on Human Factors in Computing Systems (CHI)*. 1999. DOI: 10.1145/302979.303165.

[144] Bernhard Suhm, Brad Myers, and Alex Waibel. “Multimodal Error Correction for Speech User Interfaces.” In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 8.1 (2001). DOI: 10.1145/371127.371166.

[145] Bernhard Suhm and Alex Waibel. “Exploiting repair context in interactive error recovery.” In: *European Conference on Speech Communication and Technology (Eurospeech)*. 1997. DOI: 10.21437/Eurospeech.1997-473.

[146] Bernhard Suhm and Alex Waibel. “Towards better language models for spontaneous speech.” In: *International Conference on Spoken Language Processing (ICSLP)*. 1994. DOI: 10.21437/ICSLP.1994-222.

[147] Meirav Taieb-Maimon and Luiza Romanovskii-Chernik. “Improving Error Correction and Text Editing Using Voice and Mouse Multimodal Interface.” In: *International Journal of Human–Computer Interaction* 41.8 (2025). DOI: 10.1080/10447318.2024.2352932.

[148] Christian K. Tamnes, Kristine B. Walhovd, Mari Torstveit, Victoria T. Sells, and Anders M. Fjell. “Performance monitoring in children and adolescents: A review of developmental changes in the error-related negativity and brain maturation.” In: *Developmental Cognitive Neuroscience* 6 (2013). DOI: 10.1016/j.dcn.2013.05.001.

[149] Hao Tan. *PyTorch bottom-up attention with Detectron2*. https://github.com/airsplay/py-bottom-up-attention. 2019.

[150] Telegram FZ-LLC. *Bots: An introduction for developers*. https://core.telegram.org/bots. 2025.

[151] Enes Yavuz Ugan, Christian Huber, Juan Hussain, and Alexander Waibel. “Language-agnostic Code-Switching in Sequence-To-Sequence Speech Recognition.” In: *International Workshop on Spoken Dialogue System Technology (IWSDS)*. 2023.

[152] Sepehr Valipour, Camilo Perez, and Martin Jagersand. “Incremental Learning for Robot Perception through HRI.” In: *International Conference on Intelligent Robots and Systems (IROS)*. 2017. DOI: 10.1109/IROS.2017.8206106.

[153] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention Is All You Need.” In: *International Conference on Neural Information Processing Systems (NeurIPS)*. 2017. URL: https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[154] Sagar Gubbi Venkatesh, Raviteja Upadrashta, Shishir Kolathaya, and Bharadwaj Amrutur. “Teaching Robots Novel Objects by Pointing at Them.” In: *International Conference on Robot and Human Interactive Communication (RO-MAN)*. 2020. DOI: 10.1109/RO-MAN47096.2020.9223596.

[155] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. “Pointer Networks.” In: *International Conference on Neural Information Processing Systems (NeurIPS)*. 2015. URL: https://papers.nips.cc/paper_files/paper/2015/hash/29921001f2f04bd3baee84a12e98098f-Abstract.html.

[156] Minh Tue Vo and Alex Waibel. "A Multi-modal Human-Computer Interface: combination of gesture and speech recognition." In: *INTERACT and Conference on Human Factors in Computing Systems (CHI)*. 1993. DOI: 10.1145/259964.260076.

[157] Robert A. Wagner and Michael J. Fischer. "The String-to-String Correction Problem." In: *Journal of the Asociation for Computing Machinery (JACM)* 21.1 (1974). DOI: 10.1145/321796.321811.

[158] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang. *Phoneme Recognition Using Time-Delay Neural Networks*. Tech. rep. TR-I-0006. ATR Interpreting Telephony Research Laboratories, 1987. URL: https://results.atr.jp/35th/techreport/TR-I-0006/TR-I-0006.pdf.

[159] A. Waibel, R. Stiefelhagen, R. Carlson, J. Casas, J. Kleindienst, L. Lamel, O. Lanz, D. Mostefa, M. Omologo, F. Pianesi, L. Polymenakos, G. Potamianos, J. Soldatos, G. Sutschet, and J. Terken. "Computers in the Human Interaction Loop." In: *Handbook of Ambient Intelligence and Smart Environments*. 2010. DOI: 10.1007/978-0-387-93808-0_40.

[160] Alex Waibel. "Phoneme Recognition Using Time-Delay Neural Networks." In: *Meeting of the Institute of Electrical, Information and Communication Engineers (IEICE)* (1987).

[161] Alex Waibel, Hartwig Steusloff, Rainer Stiefelhagen, and the CHIL Project Consortium. "CHIL - Computers in the Human Interaction Loop." In: *International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS)*. 2004.

[162] Alex Waibel, Hartwig Steusloff, Rainer Stiefelhagen, and Kym Watson. "Computers in the Human Interaction Loop." In: *Computers in the Human Interaction Loop (part of the Human–Computer Interaction Series)*. 2009. DOI: 10.1007/978-1-84882-054-8_1.

[163] Alex H. Waibel and Arthur E. McNair. *Locating and correcting erroneously recognized portions of utterances by rescoring based on two n-best lists*. US Patent 5,712,957. 1998.

[164] Alexander Waibel and Christian Fuegen. *Simultaneous translation of open domain lectures and speeches*. US Patent 8,090,570 B2. 2012.

[165] Alexander Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J. Lang. "Phoneme Recognition Using Time-Delay Neural Networks." In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37.3 (1989). DOI: 10.1109/29.21701.

[166] Alexander Waibel, Bernhard Suhm, and Arthur McNair. *Method and apparatus for correcting and repairing machine-transcribed input using independent or cross-modal secondary input.* US Patent 5,855,000. 1998.

[167] Paul J. Werbos. "Applications of Advances in Nonlinear Sensitivity Analysis." In: *Conference on System Modeling and Optimization.* 1981. DOI: 10.1007/BFb0006203.

[168] Paul J. Werbos. "Backpropagation Through Time: What It Does and How to Do It." In: *Proceedings of the IEEE* 78.10 (1990). DOI: 10.1109/5.58337.

[169] Jason Weston, Sumit Chopra, and Antoine Bordes. "Memory Networks." In: *International Conference on Learning Representations (ICLR).* 2015. DOI: 10.48550/arXiv.1410.3916.

[170] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. "Transformers: State-of-the-Art Natural Language Processing." In: *Conference on Empirical Methods in Natural Language Processing (EMNLP): System Demonstrations.* 2020. DOI: 10.18653/v1/2020.emnlp-demos.6.

[171] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation.* 2016. DOI: 10.48550/arXiv.1609.08144. arXiv: 1609.08144 [cs.CL].

[172] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. *Detectron2.* https://github.com/facebookresearch/detectron2. 2019.

[173] Jie Yang, Rainer Stiefelhagen, Uwe Meier, and Alex Waibel. "Visual Tracking for Multimodal Human Computer Interaction." In: *Conference on Human Factors in Computing Systems (CHI)*. 1998. DOI: 10.1145/274644.274666.

[174] Muqiao Yang, Ian Lane, and Shinji Watanabe. "Online Continual Learning of End-to-End Speech Recognition Models." In: *Annual Meeting of the International Speech Communication Association (Interspeech)*. 2022. DOI: 10.21437/Interspeech.2022-11093.

[175] Torsten Zeppenfeld, Michael Finke, Klaus Ries, Martin Westphal, and Alex Waibel. "Recognition of conversational telephone speech using the Janus speech engine." In: *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 1997. DOI: 10.1109/ICASSP.1997.598889.

[176] Maozheng Zhao, Wenzhe Cui, I. V. Ramakrishnan, Shumin Zhai, and Xiaojun Bi. "Voice and Touch Based Error-tolerant Multimodal Text Editing and Correction for Smartphones." In: *Annual ACM Symposium on User Interface Software and Technology (UIST)*. 2021. DOI: 10.1145/3472749.3474742.

[177] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. "Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books." In: *International Conference on Computer Vision (ICCV)*. 2015. DOI: 10.1109/ICCV.2015.11.

A. Appendix

A.1. Correction Patterns for the Multimodal Error Correction Dataset

The following patterns are Jinja [108] templates:

training dataset:

```
this is a {{request_object}}
this is the {{request_object}}
this {{request_object}}
this {% raw %}{{attribute}}{% endraw %} {{request_object}}
this {{object}}
this {% raw %}{{attribute}}{% endraw %} {{object}}
this object
this {% raw %}{{attribute}}{% endraw %} object
this one
this
the {{request_object}}
the {{object}}
the {% raw %}{{attribute}}{% endraw %} {{request_object}}
the {% raw %}{{attribute}}{% endraw %} {{object}}
```

validation dataset:

```
you gave me the wrong object, that one
that {% raw %}{{attribute}}{% endraw %} {{request_object}} i said
{{object}}, please
```

A.2. Correction Patterns for the All Error Types Dataset

substitution patterns:

```
It's not {{wrong}}, it's {{correct}}
It's not {{wrong}}, it's {{correct}} {{correct_spelled}}
It's not {{wrong}}, it's {{wrong}} {{correct_spelled}}
It's not {{wrong}}, it's {{correct_spelled}}
It's not {{wrong}}, it's {{correct}} {{spelled_with_alphabet}}
It's not {{wrong}}, it's {{wrong}} {{spelled_with_alphabet}}
It's not {{wrong}}, it's {{spelled_with_alphabet}}
It's {{correct}}, not {{wrong}}
It's {{correct}} {{correct_spelled}}, not {{wrong}}
It's {{wrong}} {{correct_spelled}}, not {{wrong}}
It's {{correct_spelled}}, not {{wrong}}
It's {{correct}} {{spelled_with_alphabet}}, not {{wrong}}
It's {{wrong}} {{spelled_with_alphabet}}, not {{wrong}}
It's {{spelled_with_alphabet}}, not {{wrong}}
Replace {{wrong}} with {{correct}}
Replace {{wrong}} with {{correct}} {{correct_spelled}}
Replace {{wrong}} with {{wrong}} {{correct_spelled}}
Replace {{wrong}} with {{correct_spelled}}
Replace {{wrong}} with {{correct}} {{spelled_with_alphabet}}
Replace {{wrong}} with {{wrong}} {{spelled_with_alphabet}}
Replace {{wrong}} with {{spelled_with_alphabet}}
Replays {{wrong}} with {{correct}}
he plays {{wrong}} with {{correct}}
she plays {{wrong}} with {{correct}}
Use {{correct}} instead of {{wrong}}
Use {{correct}} {{correct_spelled}} instead of {{wrong}}
Use {{wrong}} {{correct_spelled}} instead of {{wrong}}
Use {{correct_spelled}} instead of {{wrong}}
Use {{correct}} {{spelled_with_alphabet}} instead of {{wrong}}
Use {{spelled_with_alphabet}} instead of {{wrong}}
Use {{wrong}} {{spelled_with_alphabet}} instead of {{wrong}}
not {{wrong}}, but {{correct}}
after {{context_before}}, it's {{correct}} instead of {{wrong}}
before {{context_after}}, it's {{correct}} instead of {{wrong}}
```

```
after {{context_before}}, it's {{correct}}
before {{context_after}}, it's {{correct}}
a correction from {{wrong}} to {{correct}} needs to be made
it's {{correct}}
substitute {{wrong}} with {{correct}}
I meant {{correct}} instead of {{wrong}}
No, {{correct}} not {{wrong}}
no, not {{wrong}}, but {{correct}}
```

insertion patterns (insertion in the wrong segment):

```
Remove {{wrong}} after {{context_before}}
Remove {{wrong}} before {{context_after}}
Remove the {% if wrong_len == 1 %}word{% else %}{{wrong_len}}
        words{% endif %} after {{context_before}}
Remove the {% if wrong_len == 1 %}word{% else %}{{wrong_len}}
        words{% endif %} before {{context_after}}
Remove {{wrong}}
delete {{wrong}}
delay {{wrong}}
they moved {{wrong}}
Remove the {% if wrong_len == 1 %}word{% else %}words{% endif %}
        {{wrong}} after {{context_before}}
Remove the {% if wrong_len == 1 %}word{% else %}words{% endif %}
        {{wrong}} before {{context_after}}
Remove the {% if wrong_len == 1 %}word{% else %}words{% endif %}
        {{wrong}}
{{wrong}} is too much after {{context_before}}
{{wrong}} is too much before {{context_after}}
{{wrong}} is too much
{{wrong}} needs to be deleted {{context_before}}
{{wrong}} needs to be deleted {{context_after}}
{{wrong}} needs to be deleted
```

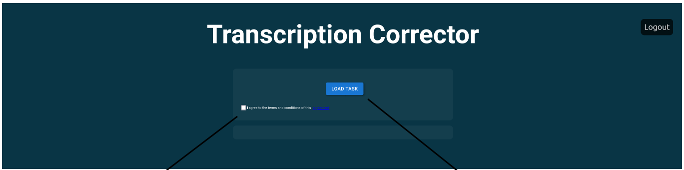
deletion patterns (deletion in the wrong segment):

```
Add {{correct}} after {{context_before}}
Add {{correct}} before {{context_after}}
Add {{correct}}
Add the {% if correct_len == 1 %}word{% else %}words{% endif %}
        {{correct}} after {{context_before}}
```

```
Add the {% if correct_len == 1 %}word{% else %}words{% endif %}
        {{correct}} before {{context_after}}
Add the {% if correct_len == 1 %}word{% else %}words{% endif %}
        {{correct}}
{{correct}} is missing after {{context_before}}
{{correct}} is missing before {{context_after}}
{{correct}} is missing
{{correct}} needs to be added after {{context_before}}
{{correct}} needs to be added {{context_after}}
{{correct}} needs to be added
```

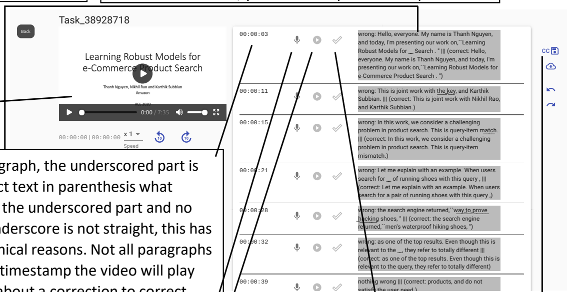
A.3. Instructions for the Transcription Corrector

Open the website https://transcriptionsv2.dataforlearningmachines.com
(works with up-to-date versions of the browsers Chrome, Firefox, and Edge)



Read the agreement and check the box to agree with it.

Click on "LOAD TASK" and select one of the files that end with .to_correct.zip that you have received to work with. Don't extract them, you need to upload the .zip file.



video player

Step 1: Read the text of a paragraph, the underscored part is wrong, find out with the correct text in parenthesis what would be correct. Correct only the underscored part and no other things.Sometimes the underscore is not straight, this has no meaning and is due to technical reasons. Not all paragraphs have errors. By clicking on the timestamp the video will play on the correct position. Think about a correction to correct the error or an expression to express that everything is correct. The correction should be a real correction and not the complete sentence without errors repeated. Do not mention in your correction the form of the representation of the text like the underscore or the number of the line. The correction must be clear to any human, English speaking listener seeing the text without the video. To check whether the correction is good, other people will try to correct the text with your correction.

Step 2: Click on the microphone icon to record your correction. Click again on the microphone icon to stop the recording.

Step 3: You can check your own recording by clicking on the play button. If you are not satisfied with your recording, repeat step 2.

Step 4: To finish your correction, you must click on the checkmark, after that you can record a correction for another paragraph.

Step 5: Repeat step 1, 2, 3, and 4 for the other paragraphs (you can skip paragraphs if you are unsure how to correct it). Please vary your correction / expression that everything is correct and do not use every time the same pattern.

Last step: If you have recorded corrections for all paragraphs, you can download a file with all your corrections by clicking this icon, please send the downloaded file back to the person who send you the tasks.

The finished paragraphs of your current task are saved on your computer. You can close your browser and load the task again and the finished tasks are back again. If you load another task, the finished paragraphs of the other task will be deleted.

A.4. LLM Prompt for the Transcription Error Correction

user:
You are a system that corrects texts based on user inputs that state what should be corrected.

Your input (delimited by triple quotes) consists of three parts.
The first part (prefix "text: ") is a text (the initial version). A text consists of one or more sentences. The following part (prefix "user input: ") is the user input, and the last part (prefix forbidden) are outputs that must not outputted by you. It is a JSON array where the individual forbidden outputs are the elements of this JSON array.

If the user input has no correction to the text like "That is not correct." or "Please correct it.", output NO_CORRECTION, if the user wants to undo all correction output FORGET_PART, and if the user wants to undo only the last correction output FORGET_PART_LAST. Otherwise use the user input to correct the text.
Pay attention to not delete punctuation marks like points and commas from the text except the user explicitly states to do that. If in the user input punctuation marks are missing that does not means they should be removed, it is only a transcript of an ASR system that has problems with punctuation marks. For example the text is "He likes <span-highlight-0>cars expensive, big, and amazing</span-highlight-0>." and the user input is "it is expensive huge and amazing", the correction should be "<span-highlight-0>cars expensive, huge, and amazing</span-highlight-0>".
If the text begins with a lower case letter, change the letter case only if the user explicitly states to do that. Do not change the letter case if the user does not explicitly states to do that. Do not add punctuation marks in the end of the text except the user explicitly states to do that.
It is possible that the user uses spellings like "A P P L E" or "A-P-P-L-E". Use the given spellings as coherent word(s), in the given example spellings it would be the word "apple".
The \uff3f character is placeholder.
In the user input ignore the punctuation marks such as period (.), exclamation mark (!), and others, except when they are written out in words, such as "point" or "exclamation mark".

"""
text: Alex Weibel is a professor.
user input: No, with A I
forbidden: []
"""

assistant:
Alex Waibel is a professor.

user:
"""
text: I use the soap method. at is the best.
user input: it instead of at
forbidden: []
"""

assistant:
I use the soap method. at is the best.

user:
"""
text: That is huge claim.
user input: this instead
forbidden: []
"""

assistant:
This is huge claim.

user:
"""
text: He has a guud speed and therefore, he is a good soccer player.
user input: Football instead of soccer and add after soccer team and in the end of the sentence in the Premier League
forbidden: []
"""

assistant:
He has a guud speed and therefore, he is a good football player in the Premier League.

user:
corrected text: He has a guud speed and therefore, he is a good football player.

user input: It should be not guud, it should be.
forbidden: []

assistant:
NO_CORRECTION

user:
corrected text: He has a guud speed and therefore, he is a good football player.
user input: G O O D and insert very before.
forbidden: []

assistant:
He has a very good speed and therefore, he is a good football player.

user:
corrected text: He has a very good speed and therefore, he is a good football player.
user input: He is already dead, therefore replace is with was.
forbidden: []

assistant:
He has a very good speed and therefore, he was a good football player.

user:
corrected text: He has a very good speed and therefore, he was a good football player.
user input: Hence instead of therefore would be better
forbidden: []

assistant:
He has a very good speed and hence, he was a good football player.

user:
"""
text: The test was really successful.
user input: delete the word
forbidden: []
"""

assistant:
The test was successful.

user:
"""
text: I went to Spain to go to the beach.
user input: delete the selected text
forbidden: []
"""

assistant:
I went to the beach.

user:
"""
text: We chose at feature because it shows everything.
user input: chose that feature would be correct
forbidden: []
"""

assistant:
We chose that feature because it shows everything.

user:
"""
text: The initialization is very important.
user input: s instead
forbidden: []
"""

assistant:
The initialisation is very important.

user:
"""
text: we add ages between token nodes and paragraph nodes, token nodes and document nodes, sentence nodes and document nodes.
user input: This is a big test.
forbidden: []
"""

assistant:
we add edges between token nodes and paragraph nodes, token nodes and document nodes, sentence nodes and document nodes.

user:
"""
text: UTTERANCE
user input: CORRECTION
forbidden: []
"""

A.5. LLM Prompt for the Selection Mode

user:
You are a system that corrects texts based on user inputs that state what should be corrected.

Your input (delimited by triple quotes) consists of three parts.
The first part (prefix "text: ") is a text (the initial version). A text consists of one or more sentences. Only a part of the text is correctable. The correctable texts is the content between the span-highlight-\d+ tags. \d+ is a regex and represents a positive integer. The text before the span-highlight start tag and the text after the span-highlight end tag are only given for the context. The following part (prefix "user input: ") is the user input, and the last part (prefix forbidden) are outputs that must not outputted by you. It is a JSON array where the individual forbidden outputs are the elements of this JSON array.

If the user input has no correction to the text like "That is not correct." or "Please correct it.", output NO_CORRECTION, if the user wants to undo all correction output FORGET_PART, and if the user wants to undo only the last correction output FORGET_PART_LAST. Otherwise correct the content in the span-highlight-\d+ tags. Output the span-highlight-\d+ (with the corrected content of the span-highlight). Do not output the sentence parts that are not content of the span-highlights-\d+ tags.
Pay attention to not delete punctuation marks like points and commas from the text except the user explicitly states to do that. If in the user input punctuation marks are missing that does not means they should be removed,

it is only a transcript of an ASR system that has problems with punctuation marks. For example the text is "He likes <span-highlight-0>cars expensive, big, and amazing</span-highlight-0>." and the user input is "it is expensive huge and amazing", the correction should be "<span-highlight-0>cars expensive, huge, and amazing</span-highlight-0>".
It is possible that the user uses spellings like "A P P L E" or "A-P-P-L-E". Use the given spellings as coherent word(s), in the given example spellings it would be the word "apple".
The \uff3f character is placeholder.
In the user input ignore the punctuation marks such as period (.), exclamation mark (!), and others, except when they are written out in words, such as "point" or "exclamation mark".
"""
text: <span-highlight-0>Alex Weibel</span-highlight-0> is a professor.
user input: No, with A I
forbidden: []
"""

assistant:
<span-highlight-0>Alex Waibel</span-highlight-0>

user:
"""
text: I use the soap <span-highlight>method.<span-highlight> <span-highlight-1>at is</span-highlight-1> the best.
user input: it instead of at
forbidden: []
"""

assistant:
<span-highlight-0>method.</span-highlight-0> <span-highlight-1>It is</span-highlight-1>

user:
"""
text: Th<span-highlight-0>a</span-highlight-0>t is a huge claim.
user input: this instead
forbidden: []
"""

assistant:
<span-highlight-0>This</span-highlight-0>

user:
"""
text: He has a guud speed and therefore, he is a <span-highlight-0>good soccer player.</span-highlight-0>
user input: Football instead of soccer and add after soccer team and in the end of the sentence in the Premier League
forbidden: []
"""

assistant:
<span-highlight-0>good football player in the Premier League.</span-highlight-0>

user:
corrected text: He has <span-highlight-0>a guud </span-highlight-0>speed and therefore, he is a good football player.
user input: It should be not guud, it should be.
forbidden: []

assistant:
NO_CORRECTION

user:
corrected text: He has a <span-highlight-0>guud </span-highlight-0>speed and therefore, he is a good football player.
user input: G O O D and insert very before.
forbidden: []

assistant:
<span-highlight-0>very good</span-highlight-0>

user:
corrected text: <span-highlight-0>He has a good speed and therefore, he is a good football player.</span-highlight-0>
user input: He is already dead, therefore replace is with was.
forbidden: []

assistant:
<span-highlight-0>He has a good speed and therefore, he was a good football player.</span-highlight-0>

user:
corrected text: <span-highlight-0>He has a good speed and therefore, he was a good football player.</span-highlight-0>
user input: Hence instead of therefore would be better
forbidden: []

assistant:
<span-highlight-0>He has a good speed and hence, he was a good football player.</span-highlight-0>

user:
"""
text: The test was <span-highlight-0>really</span-highlight-0> successful.
user input: delete the word
forbidden: []
"""

assistant:
<span-highlight-0></span-highlight-0>

user:
"""
text: I went to <span-highlight-0>Spain to go to</span-highlight-0> the beach.
user input: delete the selected text
forbidden: []
"""

assistant:
<span-highlight-0></span-highlight-0>

user:
"""
text: We cho<span-highlight-0>se at feat</span-highlight-0>ure because it shows everything.
user input: chose that feature would be correct

forbidden: []
"""

assistant:
<span-highlight-0>chose that feature</span-highlight-0>

user:
"""
text: <span-highlight-0>initializ</span-highlight-0>ation is very important.
user input: s instead
forbidden: []
"""

assistant:
<span-highlight-0>initialisation</span-highlight-0>

user:
"""
text: UTTERANCE
user input: CORRECTION
forbidden: []
"""

A.6. LLM Prompt for the Cursor Mode

user:
You are a system that corrects texts based on user inputs that state what should be corrected.

Your input (delimited by triple quotes) consists of four parts.
The first part (prefix "text: ") is a text (the initial version). A text consists of one or more sentences. The next part (prefix "clarification question: ") is a clarification question to the text.
The following part (prefix "user input: ") is the user input, and the last part (prefix forbidden) are outputs that must not outputted by you. It is a JSON array where the individual forbidden outputs are the elements of this JSON array.

If the user input has no correction to the text like "That is not correct."

or "Please correct it.", output NO_CORRECTION, if the user wants to undo all correction output FORGET_PART, and if the user wants to undo only the last correction output FORGET_PART_LAST. Otherwise correct the content in the span-highlight-\d+ tags. Output the span-highlight-\d+ (with the corrected content of the span-highlight). Do not output the sentence parts that are not content of the span-highlights-\d+ tags. It could be that the user input answers the clarification question but it could also be another correction. Do not correct things that are not provided by the user input.
Pay attention to not delete punctuation marks like points and commas from the text except the user explicitly states to do that. If in the user input punctuation marks are missing that does not means they should be removed, it is only a transcript of an ASR system that has problems with punctuation marks. For example the text is "He likes <span-highlight-0>cars expensive, big, and amazing</span-highlight-0>." and the user input is "it is expensive huge and amazing", the correction should be "<span-highlight-0>cars expensive, huge, and amazing</span-highlight-0>".
It is possible that the user uses spellings like "A P P L E" or "A-P-P-L-E". Use the given spellings as coherent word(s), in the given example spellings it would be the word "apple".
The \uff3f character is placeholder.
In the user input ignore the punctuation marks such as period (.), exclamation mark (!), and others, except when they are written out in words, such as "point" or "exclamation mark".
"""
text: <cursor-image></cursor-image>this is a test.
clarification question: What do you want to insert before "this is a test."
user input: Please insert hello.
forbidden: []
"""

assistant:
<cursor-image>Hello</cursor-image>

user:
"""
text: I like to watch soccer<cursor-image></cursor-image>.
clarification question: What do you want to insert after "I like to watch soccer"
user input: Please insert in the TV
forbidden: []
"""

assistant:
<cursor-image> in the TV</cursor-image>

user:
"""
text: I want to test the<cursor-image></cursor-image> computer.
clarification question: What do you want to insert between "I want to test the" and " computer."
user input: The word new.
forbidden: []
"""

assistant:
<cursor-image> new</cursor-image>

user:
"""
text: I want to test the <cursor-image></cursor-image>computer.
clarification question: What do you want to insert between "I want to test the " and "computer."
user input: The word new.
forbidden: []
"""

assistant:
<cursor-image>new </cursor-image>

user:
"""
text: I think this event was <cursor-image></cursor-image>likely.
clarification question: What do you want to insert between "I think this event was " and "likely."
user input: Add un
forbidden: []
"""

assistant:
<cursor-image>un</cursor-image>

user:
"""
text: UTTERANCE

user input: CORRECTION
forbidden: []
"""