THE IMPACT OF PARALINGUISTIC EVENTS ON AUTOMATIC SPEECH RECOGNITION

EVIDENCE FROM BUCKEYE CORPUS

XUEFEIYANG ZHANG

Content

Introduction

Research Background and Motivation

In daily interpersonal conversations, various paralinguistic events are often accompanied, such as laughter, throat clearing and sigh, etc. Although these non-verbal sounds do not constitute meaningful words or sentences, they play an important role in communication: On the one hand, they are natural means of expressing emotions and attitudes, capable of conveying the speaker's emotional state (good laughter, lost sighs, etc.) (Schuller et al., 2013); On the other hand, these acoustic signals can be used to regulate the rhythm of interaction and the flow of conversation. For example, laughter is often used to soften the tone and enhance affinity, while filler words or throat clearing sounds are sometimes used to maintain the right to speak and prevent the other party from interrupting, also express personality (Argyle, Alkema, & Gilmour, 1971; Isbister & Nass, 2000; Mazzocconi et al., 2022). According to research statistics, non-verbal event such as laughter account for nearly 10% of the vocalisation time in multi-person meetings, This shows their universality and importance in oral communication.(Laskowski, 2009). Therefore, studying these paralinguistic event phenomena is conducive to a better understanding of the emotional transmission and interaction regulation mechanisms in conversations.

However, the current mainstream Automatic Speech Recognition (ASR) technology lacks robustness to the above-mentioned non-verbal events. Most commercial or research-level ASR systems often use “clean” speech without interference during training. When the actual input speech signals are mixed with unconventional sounds such as laughter and sighing, the system is prone to mistake them for noise or simply ignore them. Thereby introducing deletion, replacement or insertion errors (Fukuda et al., 2018; Truong & Van Leeuwen, 2007). For example, Fukuda et al. found that when using a model trained only on neutral speech to recognise speech with emotional components, the recognition accuracy would significantly decrease. It also can be seen that the improper handling of emotional or paralinguistics components byf existing ASRs is more likely to lead to a significant reduction in transcription performance.

In an attempt to address this problem, several researchers have turned their attention to automatic detection of paralinguistics events. Relevant studies have shown that the introduction of specialised acoustic features can effectively distinguish non-verbal event from normal speech (Ludusan, 2023) . Furthermore, the Harmonics to Noise Ratio (HNR) can measure the proportion of periodic components and the spectral tilt is able to describe energy distribution. It has also been proven that these features have significant discriminative power for detecting paralinguistic events such as laughter and sighs (Ludusan & Wagner, 2022). With the help of these features to indicate the sound quality and spectral form, the accuracy of the event detection algorithm has been greatly improved. In tasks of social signal detection in Gupta, Audhkhasi, Lee, & Narayanan in 2016, the AUC of the model integrating such features for laughter events can exceed 95%. Overall, the existing literature has achieved some results in the extraction of acoustic features and event detection of paralinguistic events. these research has laid variable and solid foundation to further research.

Nevertheless, no study has systematically explored how these paralinguistic events affect the actual transcription performance of ASR. Previous work mostly focused on detecting events such as laughter itself, but lacked in-depth analysis of systematic error patterns. Are certain acoustic features (such as abnormally low HNR or significant spectral skew) associated with a high error rate? And if so, what kinds of errors are most common (deletions, insertions, substitutions) when the ASR encounters these sounds? These questions have not been effectively answered so far. For this purpose, this paper selects the commercial ASR of iFlytek as the research object. By using the paralinguistic vocal segments such as laughter, throat clearing, and sighing in the real dialogue corpus of the Buckeyes corpus, it systematically analyses the error patterns of this ASR before and after these special acoustic events. A correlation model between vocalisation features and recognition error rate is quantitatively established in combination with acoustic characteristics. iFlytek Co., Ltd. is a globally leading listed company in intelligent speech and artificial intelligence. Its core technologies cover intelligent speech recognition, synthesis, natural language processing and cognitive intelligence. Meanwhile, the ASR system of iflytek has been widely applied in fields such as government affairs, education, customer service and healthcare, with mature commercial products and large-scale real deployment cases. Therefore, it has become an ideal object for studying the performance of ASR under the challenge of paralinguistic languages. This study aims to reveal the influence mechanism of paralinguistic events on recognition performance and provide a reference for improving the robustness of ASR in natural conversation scenarios.

Research questions

Based on the above background, this paper focuses on the following Research Questions:

- RQ1: When paralinguistics events such as laughter, throat-clearing, and sigh occur

in the speech, are there systematic error patterns in the transcription of iFlytek ASR? What specific types of errors (deletion, replacement, insertion) are manifested and where do they occur frequently?

- RQ2: If such errors do exist, are they statistically associated with the acoustic characteristics of the sound emission segment (such as HNR, spectral tilt, zero cross rate ZCR, etc.)? That is to say, can these indicators be used to explain and predict situations with high error rates?

Overview of the Paper Structure and Main Conclusion

By answering the above questions, this paper aims to fill the gap in the research on paralinguistic event and the error mechanism of ASR, deepen the understanding of the causes of recognition failure, and provide a basis for designing more robust conversational ASR systems in the future.

The following structural arrangement of this article is as follows:

Chapter 2, Methodology, introduces the experimental process, including corpus preprocessing, acoustic feature extraction, error annotation methods and statistic analysis strategy; Chapter 3 Analysis & Results presents the statistical analysis of the transcription performance of iFlytek ASR before and after paralinguistic events, quantifying the types of errors and the correlation strength with acoustic characteristics; Chapter 4 Discussion gives a glance at the further view of the research topic; Chapter 5 Conclusion summarises the contributions of this paper and presents the limitations of this study as well as the future research directions.

Methodology

Corpus description

This study conducted our experiments on the Buckeye Speech Corpus, a well-known repository of spontaneous American English conversation. The Buckeye Corpus contains recorded interviews of 40 speakers from central Ohio (20 male, 20 female), balanced by age group (half under 40 years old, half over 40 years old), in order to obtain diverse samples of adult voices. Each speaker participated in an interview about everyday topics, yielding natural conversational speech; individual interviews range from approximately 30 to 60 minutes in duration. The corpus contains approximately 300,000 words of transcribed informal speech in total. The conversations are rich in disfluencies (such as pauses and fillers) and paralinguistic vocalizations.The corpus creators explicitly annotated certain non-verbal events (such as laughter) in the transcripts, which provided a valuable starting point for our study.

Laughter is officially transcribed with special tags (such as "<LAUGH>"), allowing us to automatically locate many instances of laughter in the data. However, other paralinguistic features were not systematically marked in the original transliteration. Therefore, I made additional manual annotations to identify and mark these events.

Guided by the recording and its aligned transcribed text, the study have conducted detailed annotation of the target sublanguage event. Use Praat (version 6.4.30) for annotation. For each interview recording, for the audio and its corresponding annotation file, writer compiled the Adjacent.praat script (Appendix 1) to load it into Praat. Subsequently, annotator added tier7: para to the TextGrid for paralinguistic events, marking the occurrence of each target event with precise time boundaries. define the tags as follows:

- laughter: Any laughter segment is marked as "laughter", covering the complete duration of audible laughter. Including muffled laughter (such as light laughter with vocal cord vibration) and clear laughter or breathy laughter (such as laughter with only exhalation) (if present in the paragraph).
- Throat-clearing: Any instance of throu-clearing (typically short, rough, cough-like sounds used for clearing the throat) is marked as "throat-clearing". These events are usually very brief and characterized by low tones, shrill or rough sounds.
- sigh: Any audible sigh (usually a long exhalation, sometimes accompanied by breathy or slightly voiced sounds) is marked as "sigh". We include obvious sighs and softer breaths as long as the annotator can confidently recognize them as sighs.

In this section, annotator located a grand total of 162 target occurrences of paralinguistic events: roughly 86 of laughter, around 30 of throat-clearing, and approximately 46 of sighing. Within this conversational scenario, a ‘‘sentence’’ is defined as a portion of monologic speech by a single speaker which ends with a pause, thought change, or a breath. This research also separated out the audio and TextGrid files of sentences with paralinguistic events for further analyses. For all other analyses, this research worked with the originals, which were recorded at 16kHz (16-bit PCM monophonic) to ensure compatibility with the ASR system and the tools used for the acoustic analyses. This research also defined as ‘sentence’ a defined unit of interaction as a stretch of monologic speech by one participant. This research separated out the audio and TextGrid files with paralinguistic embellishments for further analysis in this case.

Text cleaning and annotation

To evaluate the ASR performance around each paralinguinal event, this research need to input the speech containing these events into the ASR system and then compare the machine output with the reference transcription. This research did not handle the entire long interview but isolated the short audio frequency bands containing the events of interest. Specifically, for each labeled instance of laughter, throbbing or sighing, this research extracted the corresponding event along with brief context before and after it. This method reflects a real use case (ASR transcribing the session snippet where the event occurred) and ensures that this research can precisely locate local errors in the event. This research access the iflytek ASR API to obtain the transcription of each audio clip. Meanwhile, create tier8: ASR in the annotation file and import all unsegmented ASR transliterations.

Before analyzing and identifying errors, writer conducted meticulous preprocessing on the reference transcribed text and the ASR output transcribed text to ensure that their formats were consistent and easy to compare. This research carried out text normalization processing using three original scripts, namely tierlabelcheck.praat (Appendix 2), extract_tier2.praat (Appendix 3), and clean_stan_collection.praat (Appendix 4). In this section, this research have removed any elements that might interfere with the alignment of the reference text with the recognized text. This research have generated the manually aligned (tier9: ASRseg) standard transcribed word segmentation version (tier10: stanSeg) and the standard written unsegmented version (tier10: stanUnseg). First, this research convert all the transcribed text to lowercase and remove punctuation marks and fill word markers to prevent surface differences such as case or comma from being counted as errors. A key step in preprocessing is to handle the sub-language event markup in the reference transcription. In the original transcription (and our annotations), events such as laughter or coughing may be indicated by labels or parentheses (for example, in Buckeye corpora, laughter is marked as "<LAUGH>"). To calculate the word error rate, this research removed these non-lexical event markers from the reference text because they are not spoken words that the ASR can recognize. This approach can prevent ASR from being unfairly miscounted for "not outputting laughter/coughing" and the like. For incomplete words or breaks (for example, a word is only said halfway, which is usually marked with hyphens or special symbols in transliteration), this research remove the break marks and treat such words as complete words in alignment (because ASR may guess the word or omit it completely).

In the design of the annotation scheme, an important consideration is to distinguish between the segmented version and the unsegmented version. The conversation recording is initially a continuous audio segment, but for certain analytical requirements (such as the WER calculation described in the next section), it is more convenient to process it by discourse segmentation. The reason for this approach is that while conducting error analysis at the discourse level (facilitating the calculation of WER), it is still possible to understand the exact time when the error occurred relative to the event on the continuous timeline.

Through such data preparation, this research avoided false mismatches caused by format issues and focused the error analysis on substantial differences. The output of the preprocessing stage is a set of cleaned reference texts and the corresponding ASR output texts, which can now be used for alignment and error calculation in the next stage.

Word error rate (WER) culculation and error detection

After obtaining the cleaned transcribed text, this research developed a custom script werstep1.praat (Appendix 5) to calculate the Word Error Rate (WER) of the ASR system on this dataset and identify the specific types of errors that occurred. WER is a standard indicator for evaluating the accuracy of ASR, and its definition is as follows:

WER=S+D+IN×100%, \text{WER} = \frac{S + D + I}{N} \times 100\% ,WER=NS+D+I×100%

Here, S represents the number of substitution errors, D represents the number of deletion errors, I represents the number of insertion errors, and N represents the total number of reference (correct) transcriptions. This research calculate WER in units of each segment's discourse. This research achieved this alignment and error counting through a custom script - specifically, this research developed a Praat script called WERstep1.praat to automate this process. This script takes in the cleaned reference sentences and the corresponding ASR assumption sentences, and outputs the marked alignment results. The script compares the reference text and the ASR assumption word by word.

If the words match exactly, it is counted as correct recognition. If a certain word in the reference has no corresponding hypothetical word in alignment, the algorithm would inserted an empty space at that position, it is judged as a deletion error. If redundant words that do not exist in the reference are found in the ASR assumption (additional insertions occur during alignment), it is judged as an insertion error. If a word in the assumption aligns with a different word in the reference (text mismatch), it is regarded as a substitution error (i.e., the ASR wrongly identifies this word as another one).

Through the above program, writer calculated the WER of each discourse fragment and summarized the results. The WERstep1.praat script not only calculates the number of errors but also classifies each error by type and records its location. To facilitate inspection and verification, this script generates detailed error reports for each discourse. For example, the report look like this:

```
File: TextGrid s0201a_la02
  ASR Text:          "oh yes i did both"
  Standard Text: "oh yes i did vote"

  ASR word count: 5
  Standard word count: 5
  ASR Words:        [oh | yes | i | did | both]
  Standard Words:[oh | yes | i | did | vote]

  Error analysis
  Error 1: SUBSTITUTION - ASR:"both" → Standard:"vote" (position 5)

  Summary
  WER: 20.00%
  Edit distance: 1 / Reference length: 5
  Error type distribution:
    Deletions: 0
    Insertions: 0
Substitutions: 1
```

The above automatically generated format enables us to verify the accuracy of alignment and at a glance understand what types of errors have occurred. From the results, writer can see that some discourse segments have no errors at all (WER is 0%), while others have several substitution or deletion errors, etc. After performing this alignment analysis on all the utterances, writer obtained the overall WER of the entire corpus by dividing the total number of errors by the total number of words. By identifying the types of errors, writer are well-prepared to test such hypotheses in the subsequent analysis.

Temporal mapping of errors to paralinguistic events

After obtaining the time and type information of ASR errors, the next step is to map these errors onto the timeline of paralinguistics events. This step aligns the results of error analysis with the events writer manually label, thereby determining which errors are triggered by or occur simultaneously with paralinguistics events. The overall idea is: For each labeled paralinguistics event, check whether an ASR error occurs during the event or in the period immediately following it. Writer utilized the correspondence between the discourse established in the annotation stage and the continuous timeline to write the script parastep2.praat (Appendix 6) to achieve this goal. The script traverses each event in the TextGrid (with known start and end times and labels) to find the discourse unit where it is located. Then, based on the previously aligned information, determine whether an error occurred within the duration of the event or in a short time window after the event ended.

In actual operation, implementing this mapping requires combining the error analysis results with event annotation data. Writer constructed a comprehensive dataset, in which each entry corresponds to a specific labeled event instance. Each entry contains: event meta-information, including event type (laughter, throat_clear or sigh), and the start and end times of the event. For ASR error messages and time relationship variables, writer have noted that errors related to secondary language events are only valid (none, after, berfore, and during). The example is as follow:

```
File: TextGrid s0101a_la02
  Comparing 3 aligned intervals:
  Error 1: Interval 3 | ASR:"around" vs Standard:"horrendous" | Time: 0.695-1.507s
    -> Relationship: feature_during
  Errors: 1
  Before: 0
  During: 1
  After: 0
  None: 0
```

Acoustic feature extraction

After determining the position of each paralinguistics event in the audio, writer extracted quantitative acoustic features from it to characterize the sound attributes of these events. Our goal is to capture the acoustic characteristics of events such as laughter, coughing, and sighing, and subsequently analyze the relationship between these characteristics and ASR errors. According to the nature of the target event, writer selected three core features: Harmonics-to-Noise Ratio (HNR), Spectral Tilt and Zero-Crossing Rate (ZCR). These features were chosen because they can effectively distinguish between speech and non-speech/noise signals, and previous studies have shown that they can reflect the differences in sound quality and noise components. Intuitively speaking, paralinguinal events such as laughter and coughing often introduce more noise or irregular vibrations compared to normal speech. Therefore, writer expect them to have lower harmonics (lower HNR), different spectral energy distributions, and higher waveform zero-crossing rates.

Writer used the Praat script to measure the acoustic characteristics of each event.

Writer wrote a Praat script, acoustic feature.praat (Appendix 7), to automate file-by-file processing: The script opens each audio file and its corresponding TextGrid, and then traverses all intervals on the sub-language event annotation layer. For each interval marked as the target event (laughter, throat clearing or sighing), the script extracts the audio clip and calculates three features:

- HNR (Harmonic Noise Ratio) : The script calls Praat's algorithm to calculate the average harmonic nature (HNR) of this event period using the standard autocorrelation method (the lowr limit of the pitch tracking fundamental frequency is approximately set to 75 Hz). HNR is expressed in decibels (dB) as the ratio of the periodic component (harmonic) energy to the noise energy in sound. The higher the HNR value, the stronger the periodic components of the sound (such as clear voiced sounds), while a lower HNR indicates that the sound has more noise components and is more non-periodic. For instance, a continuous vowel may have a high HNR (indicating a clear phonetic pitch), while a cough or a shrill laugh, due to airflow disorder, will have a significantly lower HNR.
- Spectral Tilt: The t script achieves this measurement by first converting the extracted sound segments into power spectra and then calculating the long-term average spectrum. The specific approach is to measure the average energy within the low-frequency band (0-1000 Hz) and the high-frequency band (1000-4000 Hz), and then calculate the energy difference between the high and low-frequency bands and divide it by the bandwidth to obtain the spectral tilt value. This result essentially represents the slope of the spectrum (the rate of change of energy with frequency). A more negative spectral tilt indicates a relatively stronger low-frequency component (typical of fundamental frequency-dominated turbidity), while a less negative or even positive tilt indicates a relatively larger proportion of high-frequency components (typical of noise or clear sound). For instance, a sighing sound with breath may present a relatively flat (less negative) spectral tilt because it contains a large amount of high-frequency noise components. In contrast, a normal voiced vowel will have a steep negative incline.
- Zero-crossing rate (ZCR) : The Praat script calculates the ZCR by obtaining the point process of waveform zero-crossing within the segment and dividing the number of zero-crossing points by the duration. A higher ZCR indicates frequent changes in the waveform symbol, usually suggesting that the sound has significant high-frequency components or noise (as noise causes the waveform to fluctuate rapidly). On the contrary, voiced speech dominated by low-frequency fundamental tones has a lower ZCR. For instance, the ZCR of a noisy cough or a burst of laughter might be much higher than that of a smooth voiced sound.

Statistic analysis strategy

Finally, this study conducted a two-pronged statistical analysis using the completed event-level dataset: firstly, descriptive analysis was carried out to summarize patterns and features, and secondly, inferential analysis was performed to test our hypothesis regarding the relationship between the acoustic attributes of para-language events and ASR errors. This part fully utilized R and Rstudio(Version 2025.05.1+513) to coordinate the execution of data reading, model fitting and result output, ensuring the reproducibility of the entire process.

Descriptive Statistics

First, this study integrated the above content using the praat script master_table.praat (Appendix 8) and output a master analysis table. Then this study created an R markdown script, descriptive.stat,rmd(Appendix 9), which first examined the distribution of the data and simple relationships, including calculating the base frequency and error rate.

This study tallied the total number of errors that occurred in the corpus and subdivided their distribution by error type (deletion, insertion, replacement). For instance, this study focus on the proportion of various types of errors in all errors (such as deletion errors accounting for X%, replacement errors accounting for Y%, etc.) to understand which type of error is the most common and to grasp the ASR performance as a whole.

Then, this study particularly examined the association between paralinguistics event types and errors. For each event category (laughter, throbbing, sighing), this study calculated the frequency of ASR errors. For instance, it can be expressed as the error rate of each type of event (for example: "Among N laughter events, M are accompanied by at least one ASR error, that is, the rate is __%"). This study also further subdivide by error types: for instance, "What proportion of laughter incidents specifically correspond to deletion errors?" "Insertion error?" "Replacement error?" " . These statistical results are presented in the form of contingency tables or bar charts, etc. The purpose of doing this is to observe whether certain events (such as laughter) are more likely to trigger ASR errors than others, or are more likely to trigger specific types of errors.

This study also visualized the distribution of acoustic features (HNR, spectral tilt, ZCR) and compared the differences in these features between events that triggered errors and those that did not. For each feature, this study drew, for example, box plots or histograms, and grouped and compared the data into two groups: "any Error occurred" (Error=1) and "no error occurred" (Error=0). This enables us to make a preliminary judgment, for example, whether events with a lower HNR are more often accompanied by errors. In addition to the graphical presentation, this study also conducted a preliminary statistical test in the descriptive analysis stage.

Before explaining the model coefficients, this study tested the assumptions and performance of the model. At this step, this study conducted a multicollinearity test to examine the pairwise Spearman correlations of HNR, spectral skew, and ZC, distinguishing the independent role of each independent variable. All absolute correlations were below 0.7, indicating no severe multicollinearity. The regression coefficients can be interpreted according to their "respective independent effects" to ensure the accuracy of the logistic regression model's fitting analysis.

Inferential statistics

Based on the findings of descriptive analysis, this study established a set of logistic regression models using the script regression_model.rmd(Appendix 10) to formally test and quantify the impact of para-language event characteristics on the occurrence of ASR errors. Since our result variable is binary classification (whether an error occurs or not), logistic regression is an appropriate choice. This study constructed four independent models, each corresponding to a specific result:

- AnyError model: Dependent variable = whether any ASR error occurred (if at least one type of error occurred during/after the event, it is recorded as 1; otherwise, it is 0). This model examines the overall possibility of errors occurring.
- DelOccur model: Dependent variable = whether a deletion error occurs (1 if it occurs, 0 otherwise).
- InsOccur model: Dependent variable = Whether an insertion error occurred.
- SubOccur model: Dependent variable = Whether a substitution error occurred.

All four models use the same set of independent variables: acoustic feature HNR, spectral tilt, ZCR, and event type category. Incorporating event types (a categorical variable with three values: laughter, throat_clear, and sigh) into the model can take into account the potential systematic differences among different event categories beyond numerical characteristics. For instance, the "laughter" event itself may pose a different kind of challenge to ASR than the "sighing" event. Therefore, the event type factor is introduced to capture such categorical effects. This study performed dumb encoding processing on the event type variables (for example, taking laughter as the benchmark category) and then incorporated them into the regression. Take the AnyError model as an example, its general form can be expressed as:

logit(Pr(AnyError=1))=

β0+β1HNR+β2SpectralTilt+β3ZCR+β4(ThroatClear)+β5(Sigh)

Among them, (ThroatClear) and (Sigh) are mute variables, with laughter as the reference category. For example, expressed by the R formula, it is:

AnyError ~ HNR + spectralTilt + ZCR + EventType

The forms of other models (DelOccur, InsOccur, SubOccur) are similar, except that the binary dependent variable is defined as whether their respective error types occur or not. This study use R and Rstudio (Version: 2025.05.1+513) to fit these models.

In all models, this study have adjusted for multiple non-independent observations from the same speaker. Since each speaker may contribute multiple event instances, it may not hold true that the observations are independent of each other (the speaking style or recording conditions of the same speaker may systematically affect the error rate). To solve this problem, this study calculated the robust standard error of speaker clustering, that is, this study adjusted the standard error by clustering speakers. Specifically, after fitting each logistic regression model, this study use a robust divergence estimator (sandwich estimator) to obtain the standard error and p-value that are robust to the intra-speaker correlation. The meaning of this approach is that, for instance, even if speaker X contains many events (and thus contributes many erroneous instances), our inference takes into account the clustering of these observations in the calculation rather than treating them completely as independent data points. This method is similar to considering the speaker random effect in the model, but given that the number of speakers is relatively small (in this case, building a complete multi-level model may not be very stable), this study chose the method of clustering robust standard errors. All statistical modeling was completed in RStudio. This study used an R Markdown script (logistic_regressive.rmd) to coordinate the execution of data reading, model fitting, and result output, ensuring that the entire process was reproducible.

Each logistic regression model generates coefficients (and corresponding odds ratios) for each predictor variable, indicating the direction and significance of the predictor variable's influence on the probability of ASR errors. For instance, if the coefficient of HNR in the AnyError model is negative, it means that as HNR increases (i.e., the sound becomes more harmonious/audible), the probability of any error occurring decreases - conversely, a lower HNR (the event is noisier) is more likely to cause errors. This study did indeed discover such patterns: This study will not delve into the results here. Generally speaking, the model identified certain acoustic features as important predictors of error occurrence (for example, a lower HNR and a higher ZCR are associated with an increased probability of error occurrence, which is consistent with our expectations). The event type factor in the model also reveals the differences between different categories; For instance, after controlling for the influence of acoustic features, the probability of a certain type of event (such as throat clearing) causing errors may be higher than that of another type (such as laughter), suggesting that there are categorical influences in addition to the features this study measure. These findings will be elaborated in detail in the results section, but methodologically, logistic regression enables us to quantify these effects and assess their statistical significance.

Analysis & Results

Descriptive statistics

Distribution of paralinguistic event types

A total of 162 paralinguiistic events were marked, including 86 laughs (53.1%), 46 sighs (28.4%), and 30 throat-clearing events (18.5%), as shown in Table 1. This distribution indicates that in this dataset, laughter is the most common type of paralinguistic event, accounting for more than half of the observed. The frequency of sigh and throat-clearingis relatively low. They account for approximately 47% of the remaining events. Figure 1 visually presents this distribution. It can be seen that the frequency of laughter incidents is significantly higher than that of the other two types.

Label	n	percent
laughter	86	53.1%
sigh	46	28.4%
throat-clearing	30	18.5%
Total	162	100%

Table 1 Counts and percentages of paralinguistic event types in the dataset. Laughter was the most frequent event (n = 86, 53.1%), followed by sigh (n = 46, 28.4%) and throat-clearing (n = 30, 18.5%).

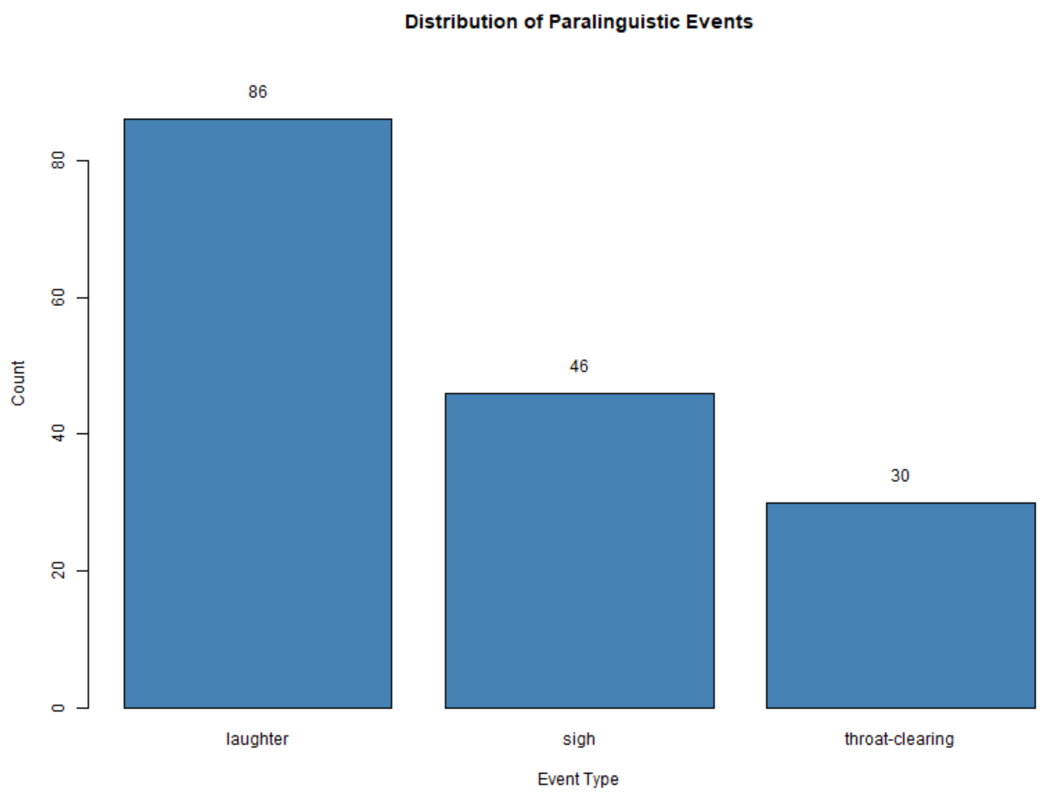

Figure 1 The distribution of paralinguistics events in the corpus. A total of 162 events werre marked, among which laughter was the most common (n = 86, accounting for 53.1%), followed by sighing (n = 46, accounting for 28.4%) and throat clearing (n = 30, accounting for 18.5%)

WER by Event Type

This paper conducted a statistical analysis of the word error rate (WER) of speech recognition for each paralinguistic event fragment. Figure 2 shows the WER distribution of the laughing, sighing and throat-clearing event segments, and Table 2 lists the corresponding descriptive statistics. The WER median of the laughing segments was only 10% (first quartile (Q1) = 0%, third quartile (Q3) = 25%), indicating that at least 25% of the laughter segments could be transcribed error-free (WER = 0). In contrast, the typical error rate of event segments containing sighing and throat-clearing was higher: the median WER of sighing segments was 14.3%, and that of throat-clearing segments was 16.0%. The quartile range of the throating-clearing segments is the narrowestern (approximately 11.5% - 20.8%, IQR ≈ 9.3%), indicating that the recognition performance of these segments is relatively consistent. On the contrary, the WER distribution of laughter segments is more dispersed (IQR = 25%), meaning that although many laughter segments are perfectly recognized, there are also some with significant recognition errors. The average WER values of the three event types are all around 17% to 19% (Table 2), among which the average WER of the segments related to sighing is the highest (19.2%).

Label	n	Mean	SD	Median	Q1	Q3	IQR
laughter	86	17.83	23.28	10	0	25	25
sigh	46	19.19	20.24	14.29	8.04	25.81	17.77
throat-clearing	30	16.54	12.95	16.03	11.46	20.79	9.33

Table 2 Summary statistics (mean, median, standard deviation, quartiles) of word error rates (WER) across event types. Laughter segments had the lowest median WER (10%), while sigh and throat-clearing showed higher central tendencies.

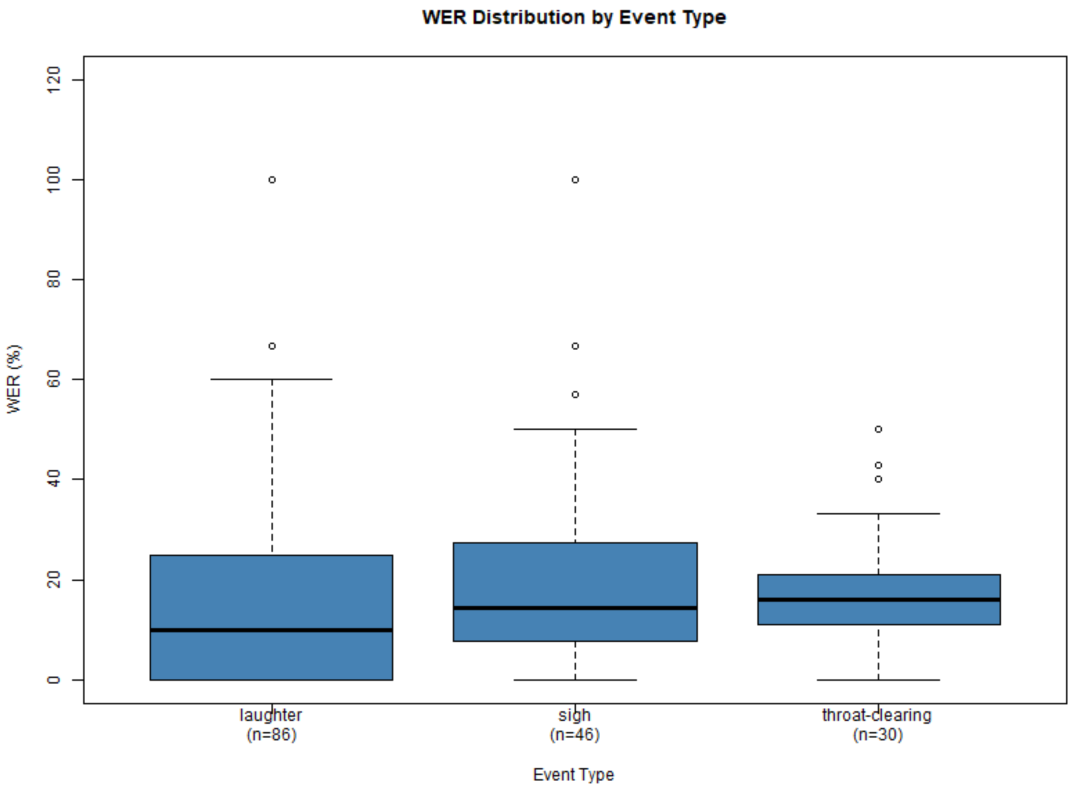

Figure 2 WER box plots of different paralinguistics event types. The median WER of the laughing segment was the lowest (10%), while the median of the sighing (14.3%) and throat-clearing (16.0%) segments was higher.

Composition of error types by event type

Subsequently, this paper analyzed the composition of the types of recognition errors corresponding to each paralinguistics event (deletion, insertion and replacement errors). Table 3 lists the quantity and percentage of each type of error in each event type fragment, and Figure 3 presents the proportion of different error types in the form of stacked bar charts. There were a total of 92 recognition errors in the smiling voice segment. Among them, nearly half were replacement errors (45 cases, accounting for

48.9%), approximately 39.1% were deletion errors (36 cases), while insertion errors were the fewest (11 cases, accounting for 12.0%). The total number of errors caused by exclamation segments was the highest (235 in total), and the composition of the errors was relatively more balanced: replacement errors accounted for approximately 43.4%, deletion errors accounted for 30.6%, and insertion errors accounted for approximately 26.0%. The total number of errors containing the throat-clearing segment was the lowest (40), and its error composition had distinct characteristics: insertion and replacement errors each accounted for 40% (16 cases each), while deletion errors only accounted for 20% (8 cases). Figure 3 (stacked bar chart) highlights these differences in the distribution of error types among different event types.

Label	Deletions	Insertions	Substitutions	TotalErrors	D_Percent	I_Percent	S_Percent
laughter	36	11	45	92	39.1	12	48.9
sigh	72	61	102	235	30.6	26	43.4
throat-clearing	8	16	16	40	20	40	40

Table 3 Frequencies and percentages of recognition error types (deletions, insertions, substitutions) within each event category.

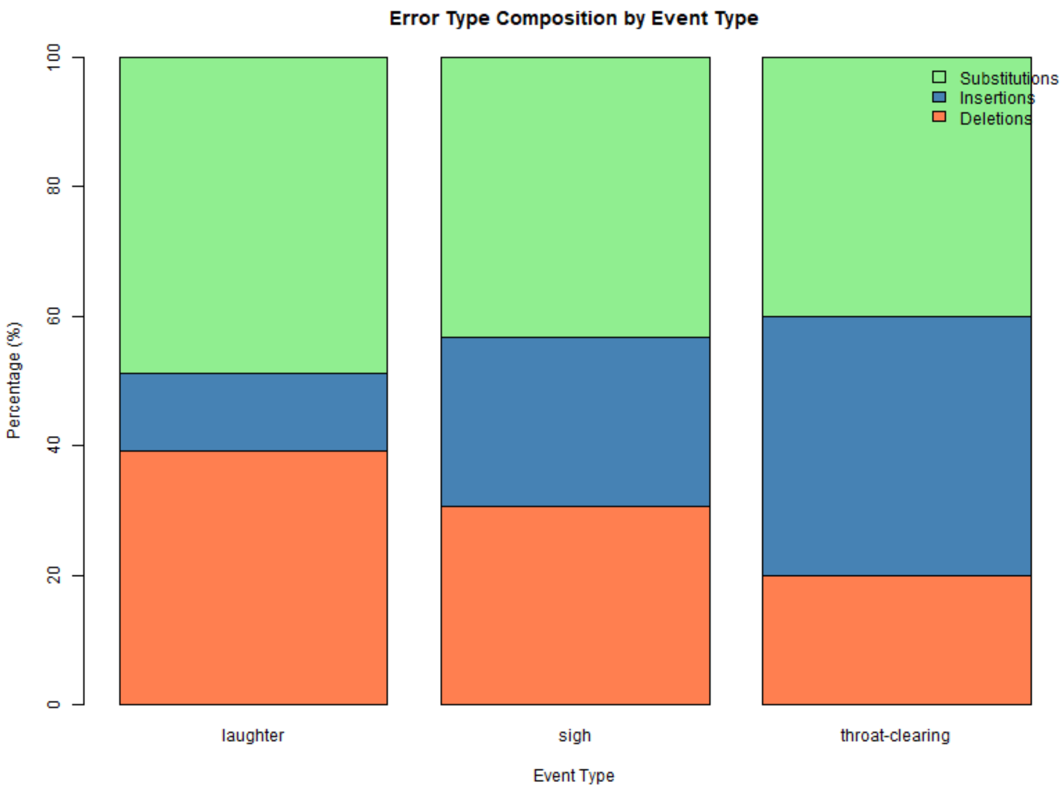

Figure 3 The composition of error types for each event type (deletion, insertion, replacement). The errors in the laughter segments are mainly replacement and deletion, while the distribution of errors

in the sighing segments is more balanced.

Temporal localization of errors

Temporal localization of errors. This paper examined where error tokens occur relative to the paralinguistic event (Table 5; Figure 5). Aggregating across sentences within each event type, 96.1% of all errors fell within a ±1 s neighborhood of the event boundary (Before + During + After), indicating that recognition mistakes cluster tightly around the event.

By event type, laughter shows a balanced split between Before (41.9%) and During (45.9%), with a smaller After share (9.5%) and very few errors elsewhere (No-Feature = 2.7%). Sigh concentrates more Before the event (50.0%) than During (31.2%), with After accounting for 16.7% and a minimal No-Feature fraction (2.1%). Throat-clearing places the largest share During the event (46.7%), with Before = 30.0%, After = 13.3%, and a comparatively larger No-Feature proportion (10.0%). These temporal profiles complement the error-type compositions (Figure 3): e.g., the stronger During concentration for throat-clearing aligns with turbulent bursts disturbing speech precisely at the event core, whereas sighs tend to perturb the lead-in portion of the sentence. Counts for reference. The underlying error-token totals by event type are: laughter 74, sigh 48, and throat-clearing 30.

Label	Before	During	After	No_Feature	Row_Total	Before_Pct	During_Pct	After_Pct	No_Feature_Pct
laughter	31	34	7	2	74	41.9	45.9	9.5	2.7
sigh	24	15	8	1	48	50	31.2	16.7	2.1
throat-clearing	9	14	4	3	30	30	46.7	13.3	10

Table 4 Distribution of error tokens relative to event timing (Before, During, After, No Feature) across event categories. Errors clustered around event boundaries, with sighs showing more “Before” errors and throat-clearing showing more “During” errors.

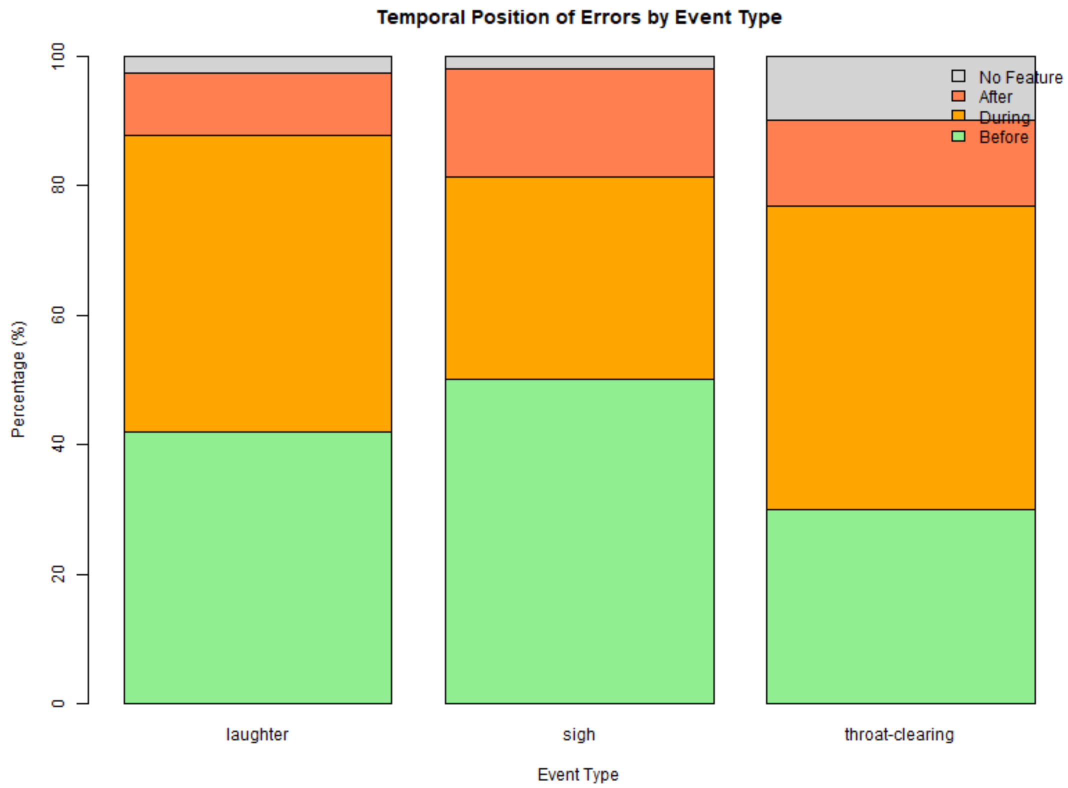

Figure 4 The temporal and positional distribution of identification errors in different event types (before the event, during the event, after the event, and no event). Errors tend to be concentrated near the boundaries of events.

Distribution of acoustic features by error occurrence

This paper further examined the relationship between certain acoustic feature values and the occurrence of recognition errors. Specifically, this paper compared the distribution differences of three acoustic indicators - HNR (Harmonic noise ratio), spectral tilt, and zero-crossing rate (ZCR) - between fragments with no recognition errors and those with at least one recognition error. As shown in Figure 4, the HNR, spectral tilt, and ZCR distributions of the error-free fragments and those containing the error-containing fragments are highly overlapped. HNR shows a slightly higher trend in error-free segments (with a median of approximately 4 dB), while in error-containing segments, the median is about 2 dB (see Figure 4), suggesting that segments with more harmonious speech quality (lower noise components) may be less prone to recognition errors. However, there was almost no difference in spectral tilt between the two groups (the median of both groups was approximately -6 dB, as shown in Figure 5). Similarly, the zero-crossing rate did not show a significant difference in terms of whether errors occurred: the median ZCR of both error-free and error-containing fragments was approximately 1.3-1.5 kHz, and the variability within each group was considerable

(Figure 6). Overall, these acoustic features did not show significant changes due to whether there were recognition errors or not.

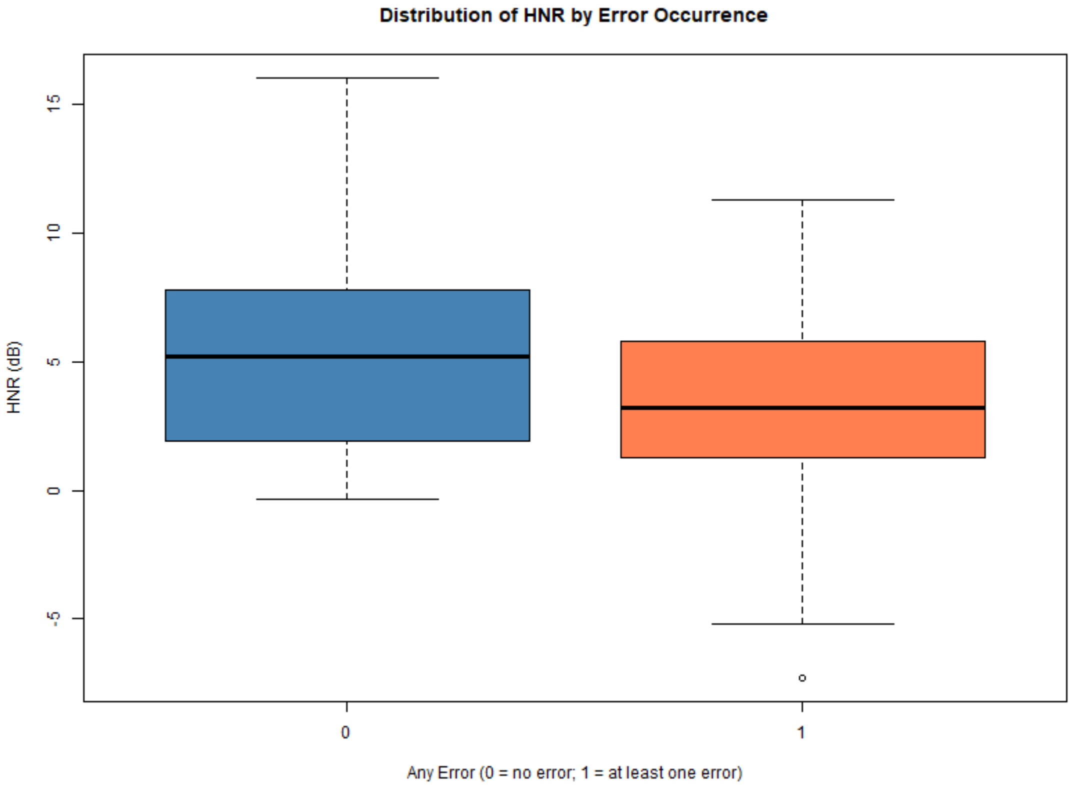

Figure 5 A box plot of HNR distribution classified by whether errors occur or not. The HNR of the error-free segments is slightly higher than that of the error-containing segments, indicating that more harmonious speech quality is more conducive to recognition

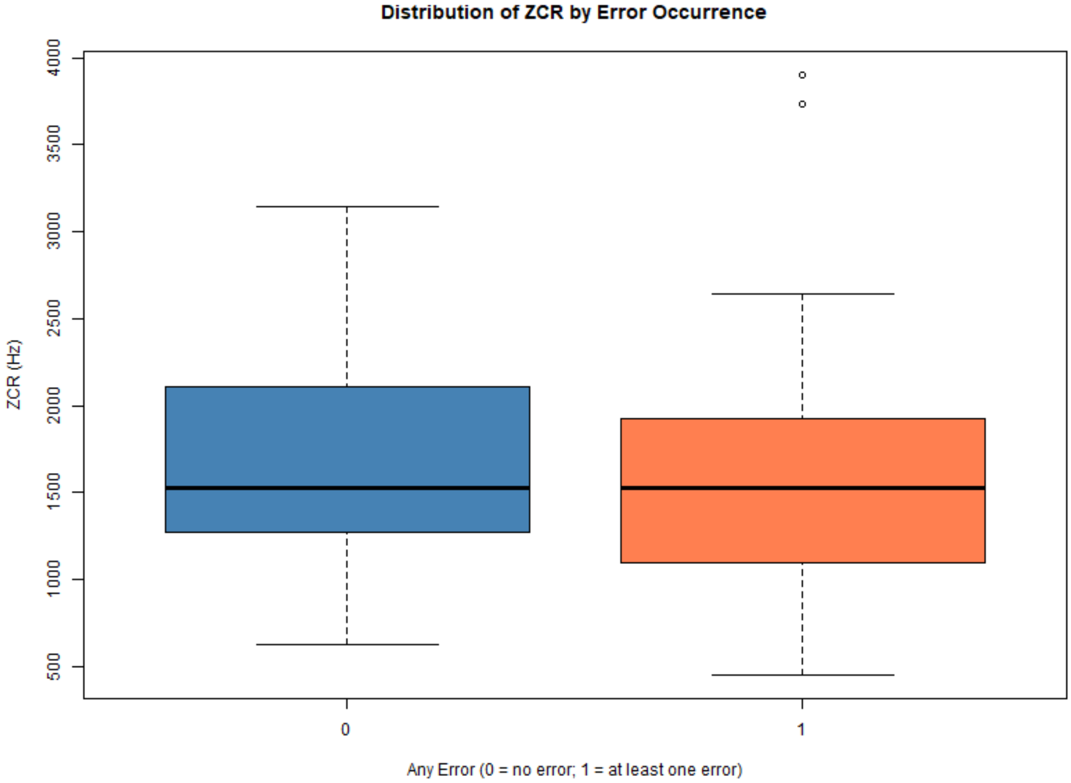

Figure 6 A box plot of ZCR distribution classified by whether errors occur or not. Both groups had significant internal variability, but the median difference was not significant.

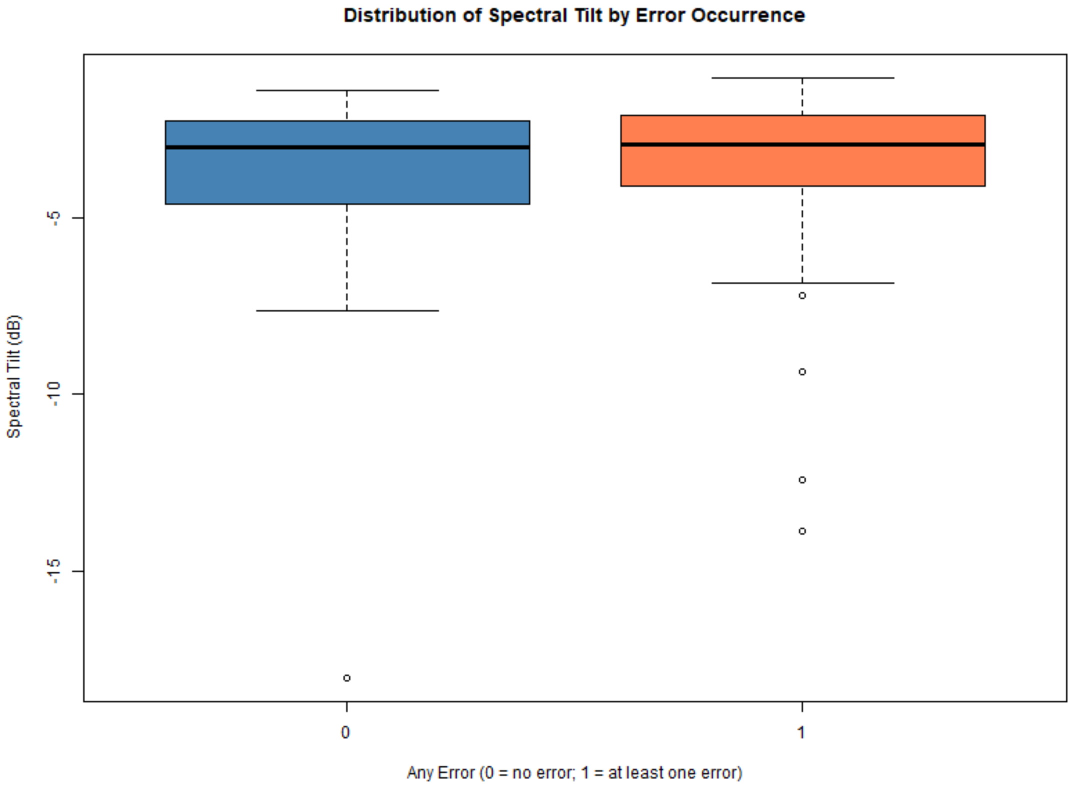

Figure 7 A box plot of spectral tilt distribution classified by whether errors occur or not. There was almost no significant difference between the two groups.

Spearman Correlation among acoustic feature

Finally, this paper calculated the Spearman rank correlation coefficients between each pair of acoustic features to evaluate the relationship among HNR, spectral tilt and ZCR (Table 4). The results show that the correlation between any pair of features has not reached a statistically significant level, and the values of the correlation coefficients are all close to zero. As shown in Table 4, the Spearman correlation coefficient ρ between HNR and spectral tilt is -0.02 ($p = 0.80$), indicating that there is almost no monotonic correlation between these two features. Similarly, the correlation between HNR and ZCR was also very low ($\rho = -0.076$, $p = 0.336$), and the correlation coefficient between spectral tilt and ZCR was 0.06 ($p = 0.452$). All these related p values are much greater than 0.05. The above results indicate that these three acoustic features are basically independent of each other in this dataset, and no significant linear or monotonic correlations have been observed.

Pair	rho	p
HNR vs. Spectral Tilt	0.047	0.582
HNR vs. ZCR	-0.107	0.214
Spectral Tilt vs. ZCR	0.131	0.127

Table 5 Pairwise Spearman rank correlation coefficients among acoustic features (HNR, spectral tilt, ZCR). None of the correlations was statistically significant, suggesting independence among features.

Inferential statistics

Logistic regression

To answer research question 2, this paper conducted a series of logistic regression analyses to examine whether the acoustic characteristics of paralinguistic events were statistically associated with a high error rate of ASR and whether they could be used to explain and predict the occurrence of errors. This study established four binary Logistic models respectively for four results: (M1) whether arbitrary identification errors occur, (M2) whether deletion errors occur, (M3) whether insertion errors occur, and (M4) whether replacement errors occur. Each model takes three acoustic features (HNR, spectral slope, and ZCR, all standardized) as continuous independent variables and event types (laughter, sighing, and throat clearing) as categorical independent variables (with laughter as the baseline category). The model adopts a robust standard error based on the speaker to consider the correlation among multiple observations of the same speaker. Table 6 summarizes the results of the four models (showing the odds ratios, their 95% confidence intervals and P-values).

Variable	M1_AnyError	M2_DelOccur	M3_InsOccur	M4_SubOccur
z-HNR	0.53 [0.31, 0.9] (0.02)	0.98 [0.78, 1.24] (0.876)	0.71 [0.34, 1.46] (0.351)	0.79 [0.5, 1.23] (0.294)
z-Spectral Tilt	1.24 [0.97, 1.59] (0.086)	1.18 [0.84, 1.66] (0.329)	1.07 [0.67, 1.71] (0.772)	1.23 [0.78, 1.94] (0.38)
z-ZCR	0.65 [0.46, 0.91] (0.013)	1.35 [0.83, 2.2] (0.229)	0.61 [0.48, 0.77] (<0.001)	0.81 [0.68, 0.98] (0.026)
Event: sigh	2.63 [0.95, 7.29] (0.062)	0.7 [0.19, 2.61] (0.593)	10.57 [3.87, 28.89] (<0.001)	0.95 [0.42, 2.16] (0.909)
Event: throat-clearing	1.59 [0.59, 4.25] (0.358)	0.81 [0.3, 2.13] (0.662)	5.86 [2.7, 12.7] (<0.001)	0.91 [0.56, 1.48] (0.719)

Table 6 Logistic regression results for acoustic predictors of ASR errors (odds ratio [95% CI] and p-value).

Model 1: any error occurrence

Model 1 examines whether any ASR error occurs. The results show that there are two acoustic features that are significant predictive factors. HNR was significantly negatively correlated with error occurrence: for every 1-standard deviation increase in HNR, the odds of error occurrence were approximately 0.53 times that of the original (odds ratio OR = 0.53, 95% CI [0.31, 0.90], p = 0.02). In other words, paragraphs with a lower HNR (i.e., fewer harmonic components and higher noise components in the speech) are more prone to recognition errors. Similarly, ZCR was also significantly negatively correlated with the occurrence of errors: for every 1-standard deviation increase in ZCR, the probability of errors was only 0.65 times that of the original (OR = 0.65, 95% CI [0.46, 0.91], p = 0.013), indicating that paragraphs with lower ZCR were significantly more prone to errors; A higher ZCR (more zeroing times, usually indicating more high-frequency components or silent noise) tends to reduce the possibility of errors occurring. The slope of the third characteristic spectrum did not reach a significant level in the model (OR = 1.24, p = 0.086), suggesting that its effect was relatively weak or there was redundancy with HNR. In terms of event types, the probability of errors caused by sighing events increased by approximately 2.6 times compared to laughter events (OR = 2.63, p = 0.062 compared to laughter), with a larger effect but not reaching a significant level. The throat-clearing event showed a smaller and less significant increase in the error rate compared to laughter (OR = 1.59, p = 0.358). Overall, noisier sounds with fewer harmonic components (low HNR) and sounds with a lower zero-crossing rate (low ZCR) are significantly associated with a higher incidence of recognition errors, which to some extent answers RQ2. After controlling for acoustic characteristics, the influence of event categories on the overall occurrence of errors is relatively weak, although the trend that sighs are more prone to errors than laughter is notable.

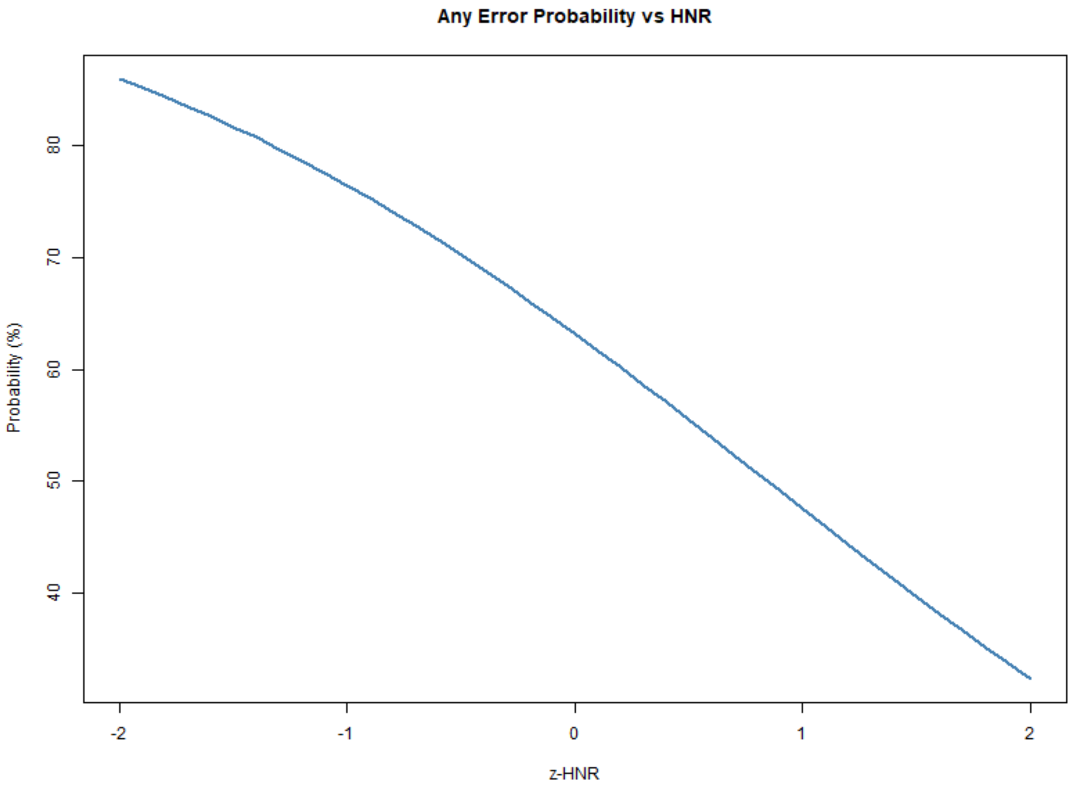

Figure 8 Prediction curve of arbitrary ASR error probability varying with HNR. The curve shows that when the HNR is low (low harmonic noise ratio), the possibility of recognition errors increases significantly.

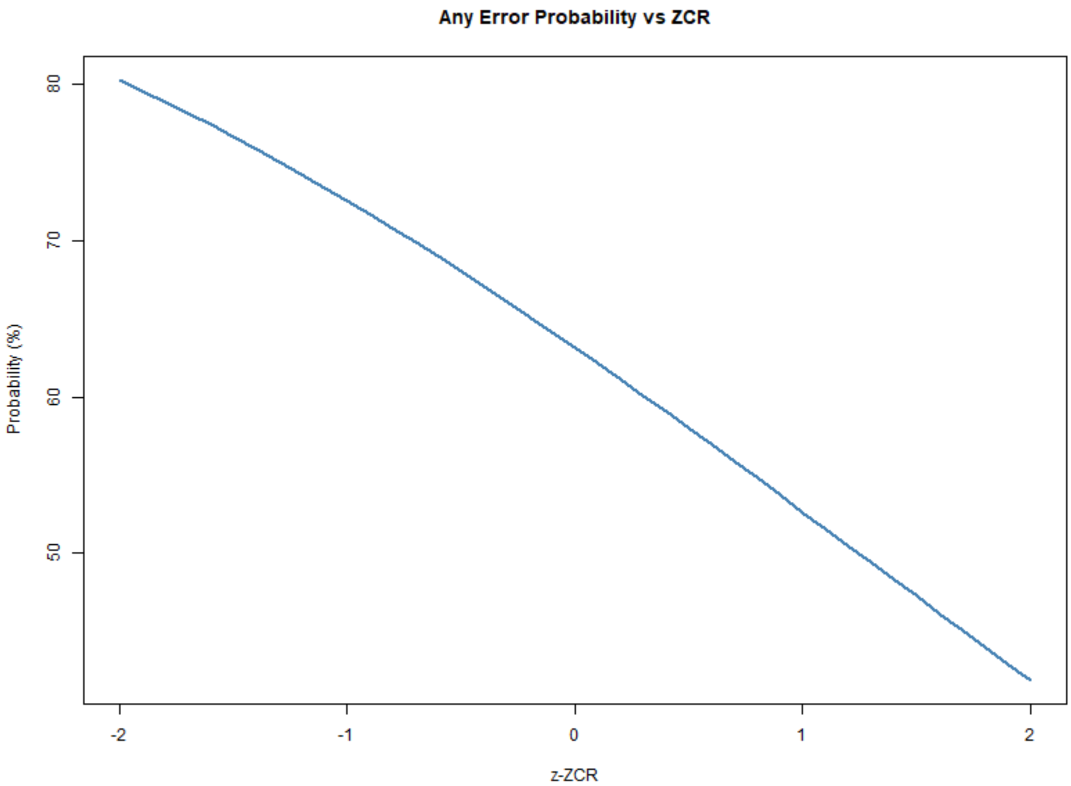

Figure 9 Prediction curve of arbitrary ASR error probability varying with ZCR. It can be seen that as ZCR increases, the probability of recognition errors decreases.

Model 2: deletion error occurrence

The second model analyzes word deletion errors (i.e., ASR misses words related to the sub-language event). Compared with the overall error model, none of the acoustic features showed a significant effect on the deletion of errors (all $p > 0.2$ in Model 2). As shown in Table 6, the odds ratios of HNR, spectral slope and ZCR for the occurrence of deletion errors are all close to 1.0, and they are not statistically significant. For instance, the OR of HNR was 0.98 ($p = 0.876$), and the OR of ZCR was 1.35 ($p = 0.229$), neither showing a reliable association. Similarly, the type of event had no significant impact on deletion errors: whether it was sighing OR clearing the voice, there was no significant difference in the occurrence rate of deletion errors compared to the laughter event (sighing OR = 0.70, $p = 0.593$; clearing the voice OR = 0.81, $p = 0.662$). This means that the occurrence of word deletion errors seems to have no obvious correlation with these acoustic indicators - under the existing data, whether the ASR misses a certain word does not systematically depend on the noise level (HNR) of the sublanguage sound, spectral tilt, or zero-crossing rate, nor does it depend on the specific type of event. One possible explanation is that the removal of errors depends more on the language environment or the behavior of the ASR's language model (such as skipping incomprehensible segments), rather than the objective acoustic features of the paralinguistics sound itself.

Model 3: insertion error occurrence

The third model focuses on insertion errors, that is, ASR hears out words in the corpus that are not actually spoken (usually caused by mistaking non-verbal sounds for verbal ones). In this model, this study identified significant and clear predictive factors. ZCR was highly significant: For every 1-standard deviation increase in ZCR, the probability of insertion errors was only 0.61 times that of the original (OR = 0.61, 95% CI [0.48, 0.77], $p < 0.001$). This indicates that paralinguistic events with a lower zero-crossing rate (fewer zero-crossing times) are more likely to induce false insertions in ASR, while events with a higher ZCR (more zero-crossing times, higher frequency and noise) are less likely to be transcribed into false words. From a practical perspective, sounds with more periodic or low-frequency components (low ZCR) are more likely to enable the ASR to "hear" non-existent words, while high-frequency noise events (high ZCR) are less likely to cause such misidentification.

It is worth noting that the event type itself has a very strong impact on insertion errors, even after controlling for each acoustic feature. While keeping HNR, spectral slope and ZCR the same, the possibility of sighing events causing insertion errors was more than ten times that of laughter events (OR = 10.57, 95% CI [3.87, 28.89], $p < 0.001$), and the possibility of throat clearing events causing insertions was approximately six times that of laughter events (OR = 5.86). 95% CI [2.70, 12.70], $p < 0.001$. These effects were statistically highly significant and consistent with the descriptive results of RQ1 - specifically, ASR often "inserted" speculative content (for example, recognizing sighs as additional syllables or words) during sighing and throat clearing, while the tendency to insert was much lower in laughter segments. Figure 8 shows the model predictions of the probability of insertion errors under different event types, clearly demonstrating the significant differences in insertion error rates between laughter and sighing, as well as throat clearing events. In conclusion, Model 3 demonstrates that both the acoustic property ZCR and the event category are powerful predictors of insertion errors. For this type of error, it also verifies RQ2: These metrics can indeed be used to explain and predict under what circumstances a higher insertion error rate will occur.

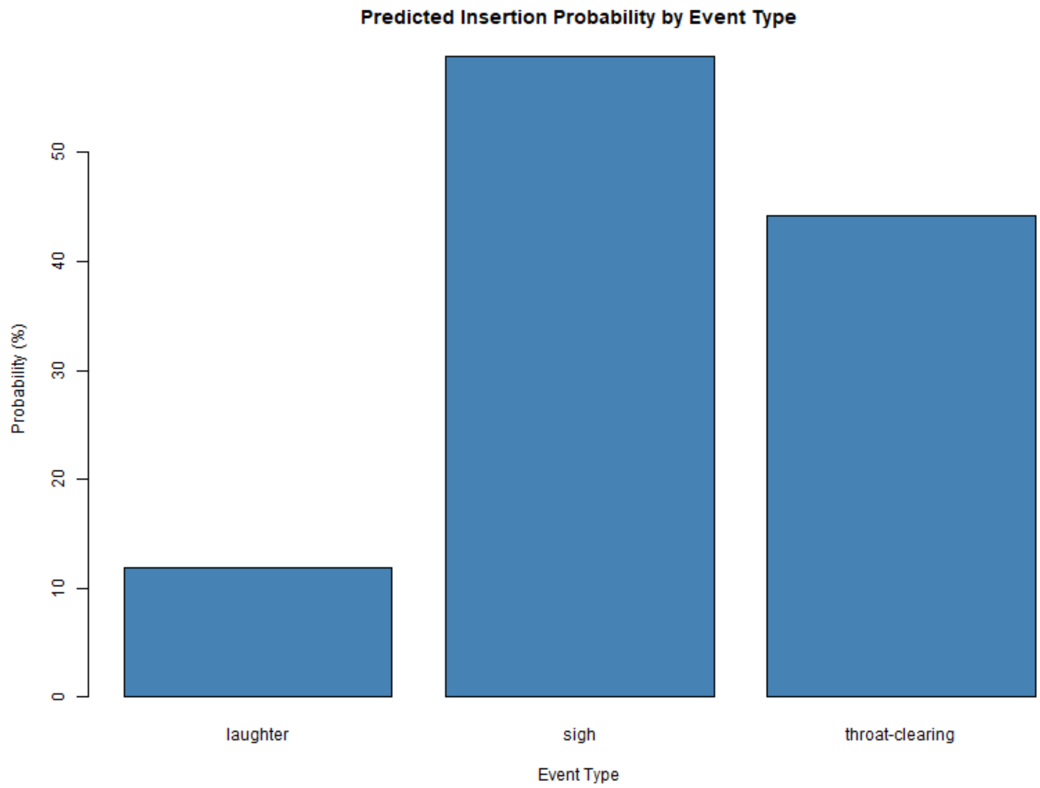

Figure 10 Probability of insertion errors predicted by paralinguistics event types.

Model 4: substitution error occurrence

The fourth model examines substitution errors, that is, ASR replaces the originally correct words with incorrect ones (usually due to incorrect recognition of the surrounding speech content when paralinguistics events occur). The discoveries of this model are relatively limited. Among the acoustic characteristics, ZCR demonstrated a moderate but significant effect: OR = 0.81 (95% CI [0.68, 0.98], $p = 0.026$), indicating that a higher ZCR was associated with a lower probability of replacement errors. From a practical perspective, for every one standard deviation increase in ZCR, the probability of replacement errors decreases by approximately 19%. Neither HNR nor spectral slope had a significant effect on replacement errors (p values were 0.294 and 0.380, respectively). Although the direction of HNR's effect remained negative (OR = 0.79), which is consistent with the view that "a lower HNR may increase the risk of errors", it did not reach statistical significance in this model. On the other hand, the type of event had no significant impact on the occurrence of replacement errors - after controlling for acoustic characteristics, there was no significant difference in the probability of replacement errors between sighing and throbbing events compared to laughter events (OR approximately 0.9, $p > 0.7$, see Table 6). This indicates that the replacement error is mainly related to certain acoustic conditions (especially low ZCR), but it does not depend much on the specific type of the event itself. When the acoustic environment is prone to confusion (for example, a very low ZCR may indicate a voice component that ASR attempts to interpret as speech), replacement errors are more likely to occur; However, once acoustic factors are taken into account, whether the sound is laughter, a sigh or a throbbing does not significantly change the probability of replacement errors occurring.

Summary

In conclusion, these inferential analyses clearly answer RQ2: There is a statistical correlation between the acoustic characteristics of specific paralural events and the possibility of ASR errors occurring, and thus can be used to a considerable extent to predict situations with high error rates. Specifically, events with low HNR (higher noise) and low ZCR (fewer zero-crossing) are associated with a higher probability of recognition errors - whether in terms of overall errors or specific error types - and this result confirms the role of these acoustic indicators as indicators of ASR ease. In addition, the types of parapultural events also play a certain role, especially in terms of insertion errors: Events such as sighing and clearing the throat essentially pose a greater challenge to ASR than laughter (leading to more insertion errors), even beyond the scope that can be purely explained by their acoustic parameters. In conclusion, RQ2's response is affirmative: There is indeed a statistical correlation between the systematic error patterns of iFLYTEK's ASR and the acoustic attributes of events such as laughter, voice clearing, and sighing. These acoustic indicators (such as HNR, spectral slope, and ZCR), as well as the event types themselves, all help explain and predict in which situations the ASR is prone to errors.

Discussion.

This study theoretically deepens the understanding of the interaction between automatic speech recognition (ASR) and paralanguage phenomena. Previous ASR studies mainly focused on vocabulary and phonetic content in spoken language, while non-verbal sounds such as laughter, sighing, and throat clearing were often regarded as background noise and ignored. Our results clearly indicate that these paralinguinal events are by no means dispensable "noises", but will have a significant impact on recognition performance.This discovery, from a new perspective, confirms the understanding of the function of paralanguage signals in the fields of linguistics and phonetics: these non-verbal acoustic signals carry emotional and interactive information in conversations and also significantly change the way machines process speech. Therefore, this study has, in a theoretical sense, built a bridge between ASR technology and paralanguage research. On the one hand, we have demonstrated that it is necessary to introduce attention to paralanguage events in ASR systems, which broadens the perspective of traditional speech recognition theories. On the other hand, we also provide a new perspective for the study of paralanguage phenomena, that is, by examining the success or failure of machine recognition, we can, in turn, gain insights into the acoustic essence and communicative function of these sounds.

In terms of technical implementation, our approach has good replicability and expansion potential. From the perspective of corpus processing, we used the Buckeye natural corpus as the basis for analysis. The Buckeye corpus contains a wealth of pronunciation variations and non-verbal events from real conversations. We made meticulous annotations on it, aligning events such as laughter, sighing, and throat clearing with adjacent words one by one, and calculated the error types and acoustic features of the corresponding segments. This strict alignment and annotation method ensures the credibility of the analysis results and also provides a template for others to reproduce the research. In terms of acoustic feature extraction, this study selected classic indicators such as HNR (Harmonic Noise ratio), spectral skew, and zero-crossing rate (ZCR) to quantify the sound characteristics of sub-language events. The extraction of these features relies on mature tools and algorithms, and thus is universal. Any researcher using a similar speech database can repeat our process: first, label the paralingual events, then extract the above acoustic indicators, and finally conduct a correlation analysis with the recognition errors. In addition, this study adopted a method combining statistical description and Logistic regression model to explore the correlation between features and errors. This methodology is transparent and easy to promote. The results of Logistic regression not only provide interpretable statistical associations but also serve as a basis for comparison with more complex machine learning models in the future. Overall, our technical route is clear and the steps are well-defined, providing a referenceable paradigm for research in related fields.

The findings of this study have significant implications for enhancing the robustness of actual ASR systems. Firstly, this study confirmed that paralingual events can significantly reduce the recognition accuracy of ASR, and the impact of different types of events on error patterns varies. For instance, this study observed that although laughter occurred most frequently in the corpus, it caused relatively less harm to ASR. It is speculated that the reason lies in the fact that laughter has a certain harmonic structure, making it easier for ASR to identify that it is not normal speech and thus less likely to misrecognize it as lexical content. Events like sighing and throat clearing often have irregular sounds and abnormal spectral energy distribution, which makes them more likely to confuse the recognition model, leading to the insertion of additional false words or the omission of the real words that follow immediately. Through error type analysis, this study found that when ASR encounters sighing and throat clearing, it tends to generate a relatively high proportion of insertion errors - the system will mistake these noises for some kind of pronunciation and "mishear" non-existent words. On the contrary, in the laughter section, there are significantly fewer insertion errors, indicating that the system is more likely to treat laughter as muted. This difference indicates that enhancing the ASR's ability to distinguish different sub-language acoustic patterns is the key to improving the system's accuracy. From an application perspective, ASR developers should consider explicitly handling these non-verbal events in the model. For instance, in actual voice assistants or transcription tools, a preprocessing module can be added. When events such as laughter or sighing are detected, special markers can be used to replace or filter the audio segment, thereby preventing the misidentification of incorrect content. This approach is consistent with the "paralinguinal perception" recognition concept proposed in the latest research: integrating paralinguinal cues as decodable special labels into the recognition output, enabling the system to simultaneously transcribe both lexical and non-verbal information.With such improvements, ASR will no longer simply ignore or mishear situations like laughter in natural conversations, but can handle them more robustably, enhancing the user experience and transcription quality in practical applications.

It is worth further discussion that our analysis also reveals that the mechanism by which paralanguage events cause errors is closely related to their acoustic properties. The Logistic regression results show that changes in specific acoustic features significantly affect the probability of error occurrence. Among them, a low HNR value (i.e., high noise component) and a low ZCR value (i.e., low waveform zero crossover rate, suggesting a stronger periodic component) are both statistically correlated with an increase in ASR error rate. This indicates that when the paralingual sounds are characterized by being noisy or dominated by low frequencies, the recognition model is more prone to confusion. This conclusion is in line with intuition: highly noisy and irregular sounds can interfere with the model's matching of normal speech patterns, thereby increasing the possibility of recognition failure. Furthermore, this study found that even after controlling for the above-mentioned acoustic features, different event types themselves still have differences in influence. For instance, the possibility of an sighing event leading to insertion misidentification is much higher than that of laughter, even if the HNR, spectral tilt and other values of the two are similar. This implies that apart from simple acoustic parameters, there are more complex differences in signal morphology between laughter and sighing (for instance, laughter is often accompanied by vowel pitch fluctuations, while sighing is more of a continuous airflow sound).

These differences make ASR even more helpless when it comes to sighing. This discovery emphasizes at a deeper level the importance of incorporating event type information into the recognition process: perhaps future models can handle and model such specific events differently to reduce the errors caused by them.

Finally, this study provides some valuable lessons and experiences for the design and training of future ASR systems. Our analysis of the time distribution of errors shows that recognition errors tend to occur in the period near the occurrence of paralanguage events (approximately within one second before and after). This means that the interference of paralanguage events on ASR is mainly local and immediate, and will not have a continuous impact on distant speech segments. Therefore, the ASR system can focus on optimizing these critical moments. For instance, when laughter is detected or throat clearing has just ended, the system can temporarily reduce the language model's trust in the output of the voice content or increase the tolerance for silence/noise to avoid the trap of mistaking instantaneous abnormal sounds for words. Similarly, during the model training phase, training data containing para-language events should be purposefully added (and the correct event positions should be marked), enabling the model to learn to "skip" or "go blank" during these brief interludes, rather than forcing words to match. In addition, our research has demonstrated that simple acoustic features can already effectively predict high-error scenarios, suggesting that future ASRs can combine these easily extracted indicators to achieve online error early warning or adaptive adjustment. For instance, the HNR, ZCR and other values of the input voice are monitored in real time. Once an abnormal range is detected (which may correspond to the occurrence of laughter, etc.), the system can adjust the decoding strategy or activate a dedicated sub-language processing module. In conclusion, the techniques and experiences of this study point out the direction for building more robust ASR systems for natural dialogue: integrating the detection and processing of paralingual events will help significantly reduce the recognition error rate in real-world applications.

Conclusion

This paper systematically studies the impact of paralanguage events (such as laughter, sighing, and throat clearing) in natural conversations on automatic speech recognition systems, and has achieved the following main results and contributions. Firstly, based on the Buckeye corpus and iFLYTEK's commercial ASR system, this study quantitatively demonstrated that para-language events can lead to a decline in recognition performance and detailedly revealed the differences in error patterns triggered by different events. This fills a gap in existing research - few previous works have explored paralanguage phenomena so deeply from the perspective of recognition errors. This study not only reported the changing trend of the overall recognition error rate when paralingual events occurred, but also for the first time associated specific error types (insertion, deletion, replacement) with specific events, depicting a unique map of the errors that ASR is prone to under different paralingual events. Secondly, this study analyzed the internal factors causing the errors in combination with acoustic characteristics and found that parameters such as harmonic-noise ratio, spectral tilt, and zero-crossing rate were significantly associated with the identification errors. This discovery provides an empirical basis for explaining the easily confused signal characteristics of ASR and verifies that certain acoustic indicators can serve as effective signals for predicting and identifying difficulties. Overall, the research work of this paper has made new progress in the intersection of ASR robustness and paralingual signal processing: our conclusions emphasize the importance of taking paralingual events into account for improving recognition accuracy and provide empirical support for future improvements in ASR.

Despite the above achievements, this study still has some limitations that need to be overcome in future work. Firstly, in terms of the corpus, the Buckeye dialogue library used in the research is relatively limited in scale and the language is English. Although this corpus covers a wealth of spontaneous oral phenomena, its representativeness is still limited. Subsequent research can introduce larger-scale, multilingual natural dialogue data to verify the universality of the findings of this study. Secondly, the ASR system this study selected is a single commercial model (iFLYTEK), and its architecture and training data are specific. Therefore, the application of this result on different recognition engines (such as other commercial systems or open-source models) still needs further investigation. Secondly, the types of paralingual events focused on in this article mainly include laughter, sighing, and throat clearing, and do not include other common non-verbal sounds such as crying, panting, and filler words. These unexplored factors may also have an impact on recognition and are worthy of being taken into account in future research. Furthermore, in terms of methods, the statistical modeling this study adopt (such as Logistic regression) assumes a linear feature interaction relationship and may not be able to capture more complex nonlinear influences. In the future, more complex error prediction models can be constructed by means of deep learning and other methods, or directly used for real-time detection of paralanguage events. Finally, regarding the improvement of the ASR system itself, our research only put forward directional suggestions and did not implement countermeasure verification in this paper. For instance, integrating paralingual event markers into ASR decoding or adding dedicated pre-detection modules, the practical effects of these schemes remain to be evaluated through new experiments.

In conclusion, enhancing the robustness of ASR in natural conversation scenarios is a challenging yet significant task. Our research reveals that paralanguage events are one of the key factors affecting recognition performance, highlighting the shortcomings of traditional ASR systems in human-computer interaction environments. Future research should verify and expand the conclusions of this study on a broader range of data and models, and explore innovative methods to enable the ASR recognition engine to handle non-verbal sounds such as laughter more "intelligently". For instance, develop models capable of jointly transcribing speech and paralinguinal signals, enabling machines not only to "understand" what is said but also to mark the speaker's laughter, sighs and other behaviors; Or design a multimodal interaction system that combines voice recognition with signals such as expressions and postures to reduce the interference of non-verbal events in pure audio. This study believe that as these directions are further advanced, future automatic speech recognition will be closer to human auditory capabilities, maintaining high accuracy even in noisy and ever-changing conversations, laying the foundation for more natural voice interaction.

Reference

Argyle, M., Alkema, F., & Gilmour, R. (1971). The communication of friendly and hostile attitudes by verbal and non-verbal signals. *European Journal of Social Psychology*, *1*(3), 385–402. https://doi.org/10.1002/ejsp.2420010307

Fukuda, T., Ichikawa, O., & Nishimura, M. (2018). Detecting breathing sounds in realistic Japanese telephone conversations and its application to automatic speech recognition. *Speech Communication*, *98*, 95–103. https://doi.org/10.1016/j.specom.2018.01.008

Gupta, R., Audhkhasi, K., Lee, S., & Narayanan, S. (2016). Detecting paralinguistic events in audio stream using context in features and probabilistic decisions. *Computer Speech & Language*, *36*, 72–92. https://doi.org/10.1016/j.csl.2015.08.003

ISBISTER, K., & NASS, C. (2000). Consistency of personality in interactive characters: verbal cues, non-verbal cues, and user characteristics. *International Journal of Human-Computer Studies*, *53*(2), 251–267. https://doi.org/10.1006/ijhc.2000.0368

Laskowski, K. (2009). Contrasting emotion-bearing laughter types in multiparticipant vocal activity detection for meetings. *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, 4765–4768. https://doi.org/10.1109/ICASSP.2009.4960696

Ludusan, B. (2023). The usefulness of phonetically-motivated features for automatic laughter detection. *Disfluency in Spontaneous Speech (DiSS) Workshop 2023*, 33–37. https://doi.org/10.21437/DiSS.2023-7

Ludusan, B., & Wagner, P. (2022). ha-HA-hha? Intensity and voice quality characteristics of laughter. *Speech Prosody 2022*, 560–564. https://doi.org/10.21437/SpeechProsody.2022-114

Mazzocconi, C., Tian, Y., & Ginzburg, J. (2022). What's Your Laughter Doing There? A Taxonomy of the Pragmatic Functions of Laughter. *IEEE Transactions on Affective Computing*, *13*(3), 1302–1321. https://doi.org/10.1109/TAFFC.2020.2994533

Schuller, B., Steidl, S., Batliner, A., Burkhardt, F., Devillers, L., Müller, C., & Narayanan, S. (2013). Paralinguistics in speech and language—State-of-the-art and the challenge. *Computer Speech & Language*, *27*(1), 4–39. https://doi.org/10.1016/j.csl.2012.02.005

Truong, K. P., & Van Leeuwen, D. A. (2007). Automatic discrimination between laughter and speech. *Speech Communication*, *49*(2), 144–158. https://doi.org/10.1016/j.specom.2007.01.001

Appendix

Appendix 1: Adjacent.praat

```
form: "Report on Buckeye corpus"
	folder: "Buckeye folder", "/Volumes/Buckeye"
endform

corpusFolderPath$ = buckeye_folder$
writeInfoLine: "Reporting on the Buckeye folder “", corpusFolderPath$, "”..."

stopwatch
folderNames$# = folderNames$#: corpusFolderPath$ + "/s*"
numberOfFolders = size (folderNames$#)

for ifolder to numberOfFolders
	folderName$ = folderNames$# [ifolder]
	folderPath$ = corpusFolderPath$ + "/" + folderName$
	appendInfoLine: folderPath$
	subfolderNames$# = folderNames$#: folderPath$ + "/*"
	numberOfSubfolders = size (subfolderNames$#)

	for isubfolder to numberOfSubfolders
		subfolderName$ = subfolderNames$# [isubfolder]
		subfolderPath$ = folderPath$ + "/" + subfolderName$
		appendInfo: subfolderPath$

		#
		# Read all the data.
		#
		soundFilePath$ = subfolderPath$ + "/" + subfolderName$ + ".wav"
		Read Sound with adjacent annotation files (Buckeye): soundFilePath$
				selectObject: "Sound untitled"
				Rename: subfolderName$
				selectObject: "TextGrid untitled"
				Rename: subfolderName$
	endfor
endfor
```

Appendix 2: tierlabelcheck.praat

```
# Script to check tier 2 and tier 3 names in all TextGrid files in the script folder

folder$ = "scripts/"

# Create output file to save the results
output_file$ = "output_data/tier_names_report.txt"
writeFile: output_file$, "TextGrid Tier Names Report", newline$
appendFile: output_file$, "Generated on: ", date$(), newline$
appendFile: output_file$, "=================================", newline$,
newline$

# Get all TextGrid files in the folder
Create Strings as file list: "textgrid_list", folder$ + "*.TextGrid"
numFiles = Get number of strings

# Check if any TextGrid files were found
if numFiles = 0
	appendFile: output_file$, "No TextGrid files found in folder: ", folder$, newline$
	writeInfoLine: "No TextGrid files found in folder: ", folder$
else
	writeInfoLine: "Found ", numFiles, " TextGrid files. Processing..."
	appendFile: output_file$, "Found ", numFiles, " TextGrid files:", newline$,
newline$

	# Process each TextGrid file
	for i from 1 to numFiles
		selectObject: "Strings textgrid_list"
		fileName$ = Get string: i
		fullPath$ = folder$ + fileName$

		# Try to read the TextGrid file

			textgrid_id = Read from file: fullPath$
			selectObject: textgrid_id

			# Get basic information about the TextGrid
			numTiers = Get number of tiers

			# Write file name to output
			appendFile: output_file$, "File: ", fileName$, newline$
			appendFile: output_file$, "Total tiers: ", numTiers, newline$
```

```
        # Check tier 2
        if numTiers >= 2
            tier2_name$ = Get tier name: 2
            appendFile: output_file$, "Tier 2 name: ", tier2_name$, newline$
            writeInfoLine: fileName$, " - Tier 2: ", tier2_name$
        else
            appendFile: output_file$, "Tier 2: NOT FOUND (file has only ",
numTiers, " tier(s))", newline$
            writeInfoLine: fileName$, " - Tier 2: NOT FOUND"
        endif

        # Check tier 3
        if numTiers >= 3
            tier3_name$ = Get tier name: 3
            appendFile: output_file$, "Tier 3 name: ", tier3_name$, newline$
            writeInfoLine: fileName$, " - Tier 3: ", tier3_name$
        else
            appendFile: output_file$, "Tier 3: NOT FOUND (file has only ",
numTiers, " tier(s))", newline$
            writeInfoLine: fileName$, " - Tier 3: NOT FOUND"
        endif

        appendFile: output_file$, newline$

        # Clean up
        Remove
    endfor
endif

# Clean up
removeObject: "Strings textgrid_list"

# Final message
appendFile: output_file$, "===================================", newline$
appendFile: output_file$, "Report completed on: ", date$(), newline$
writeInfoLine: "Processing complete! Results saved to: ", output_file$
```

Appendix 3: extract_tier2.praat

```
# Extract all tier 2 text to check if there's annotation mistake between tiers

folder$ = "scripts/"
outputFile$ = "output_data/tier2_content_analysis.txt"

# get all TextGrid
Create Strings as file list: "textgrid_list", folder$ + "*.TextGrid"
numFiles = Get number of strings


writeInfoLine: "find ", numFiles, " textgrid， extracting tier 2"


# output
output$ = "tier 2 content analysis" + newline$
output$ = output$ + "time: " + date$() + newline$
output$ = output$ + "folder: " + folder$ + newline$
output$ = output$ + "==================================================" +
newline$ + newline$

allUniqueLabels$ = "|"
specialCharacters$ = ""
totalIntervals = 0
filesProcessed = 0

# process every TextGrid
for i from 1 to numFiles
	selectObject: "Strings textgrid_list"
	fileName$ = Get string: i
	fullPath$ = folder$ + fileName$

	writeInfoLine: "processing ", i, "/", numFiles, ": ", fileName$

	# read TextGrid
	textgrid_id = Read from file: fullPath$
	selectObject: textgrid_id

	# check tier number
	numTiers = Get number of tiers
	if numTiers < 2
		output$ = output$ + "文件: " + fileName$ + " warning: only " +

string$(numTiers) + " 个 tier， skip" + newline$
```

```
        Remove
        goto NEXT_FILE
    endif

    # add filename
    output$ = output$ + "file: " + fileName$ + " ===" + newline$

    # get tier 2
    tierName$ = Get tier name: 2
    output$ = output$ + "Tier2name: " + tierName$ + newline$

    # get all intervals in tier 2
    numIntervals = Get number of intervals: 2
    output$ = output$ + "Intervals number: " + string$(numIntervals) + newline$

    fileIntervalCount = 0

    for j from 1 to numIntervals
        intervalText$ = Get label of interval: 2, j
        startTime = Get start time of interval: 2, j
        endTime = Get end time of interval: 2, j

        # only non empty intervals
        if intervalText$ <> ""
            fileIntervalCount = fileIntervalCount + 1

            output$ = output$ + "    [" + fixed$(startTime, 3) + "-" +
fixed$(endTime, 3) + "]: " + intervalText$ + newline$

            # collect labels
            searchPattern$ = "|" + intervalText$ + "|"
            if index(allUniqueLabels$, searchPattern$) = 0
                allUniqueLabels$ = allUniqueLabels$ + intervalText$ + "|"
            endif

            call analyzeSpecialCharacters: intervalText$
        endif
    endfor

    output$ = output$ + "non empty intervals number: " + string$(fileIntervalCount)
+ newline$ + newline$

    totalIntervals = totalIntervals + fileIntervalCount
    filesProcessed = filesProcessed + 1
```

```
	Remove

	label NEXT_FILE
endfor

# output
output$ = output$ + "==================================================" +
newline$
output$ = output$ + "output:" + newline$
output$ = output$ + "file number: " + string$(filesProcessed) + newline$
output$ = output$ + "interval number: " + string$(totalIntervals) + newline$ +
newline$

# analyse labels
output$ = output$ + "label list" + newline$
call extractUniqueLabels: allUniqueLabels$
output$ = output$ + uniqueLabelsReport$ + newline$

# analysis
output$ = output$ + "--- 特殊字符和模式分析 ---" + newline$
call analyzePatterns: allUniqueLabels$
output$ = output$ + patternsReport$ + newline$

# 保存结果到文件
writeFile: outputFile$, output$

removeObject: "Strings textgrid_list"

writeInfoLine: "completed！"



# analysis characters
procedure analyzeSpecialCharacters: text$
	textLength = length(text$)
	normalChars$ =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012345678
9 -"

	for k from 1 to textLength
		char$ = mid$(text$, k, 1)
```

```
        # collect special characters
        if index(normalChars$, char$) = 0
            if index(specialCharacters$, char$) = 0
                specialCharacters$ = specialCharacters$ + char$
            endif
        endif
    endfor
endproc

# extract uniquelabels
procedure extractUniqueLabels: labelString$
    uniqueLabelsReport$ = ""
    labelCount = 0

    remainingString$ = labelString$

    if left$(remainingString$, 1) = "|"
        remainingString$ = right$(remainingString$, length(remainingString$) - 1)
    endif

    while length(remainingString$) > 0
        separatorPos = index(remainingString$, "|")

        if separatorPos = 0
            if remainingString$ <> ""
                labelCount = labelCount + 1
                uniqueLabelsReport$ = uniqueLabelsReport$ +
string$(labelCount) + ". " + remainingString$ + newline$
            endif
            remainingString$ = ""
        else
            currentLabel$ = left$(remainingString$, separatorPos - 1)
            if currentLabel$ <> ""
                labelCount = labelCount + 1
                uniqueLabelsReport$ = uniqueLabelsReport$ +
string$(labelCount) + ". " + currentLabel$ + newline$
            endif
            remainingString$ = right$(remainingString$, length(remainingString$)
- separatorPos)
        endif
    endwhile

    uniqueLabelsReport$ = "in total: " + string$(labelCount) + " unique labels:" +
newline$ + uniqueLabelsReport$
```

```
endproc

# analyzePatterns
procedure analyzePatterns: labelString$
	patternsReport$ = ""

	bracketCount = 0
	laughCount = 0
	cutCount = 0
	noiseCount = 0
	errorCount = 0
	dashCount = 0

	remainingString$ = labelString$

	if left$(remainingString$, 1) = "|"
		remainingString$ = right$(remainingString$, length(remainingString$) - 1)
	endif

	while length(remainingString$) > 0
		separatorPos = index(remainingString$, "|")

		if separatorPos = 0

			currentLabel$ = remainingString$
			remainingString$ = ""
		else

			currentLabel$ = left$(remainingString$, separatorPos - 1)
			remainingString$ = right$(remainingString$, length(remainingString$)
- separatorPos)
		endif

		if currentLabel$ <> ""

			if index(currentLabel$, "<") > 0 and index(currentLabel$, ">") > 0
				bracketCount = bracketCount + 1
			endif

			if index(currentLabel$, "laugh") > 0 or index(currentLabel$,
"LAUGH") > 0 or index(currentLabel$, "Laugh") > 0
				laughCount = laughCount + 1
			endif
```

```
            if index(currentLabel$, "cut") > 0 or index(currentLabel$, "CUT") > 0
or index(currentLabel$, "Cut") > 0
                cutCount = cutCount + 1
            endif
            if index(currentLabel$, "voc") > 0 or index(currentLabel$, "VOC") > 0
or index(currentLabel$, "noise") > 0 or index(currentLabel$, "NOISE") > 0
                noiseCount = noiseCount + 1
            endif
            if index(currentLabel$, "error") > 0 or index(currentLabel$,
"ERROR") > 0 or index(currentLabel$, "Error") > 0
                errorCount = errorCount + 1
            endif
            if index(currentLabel$, "-") > 0
                dashCount = dashCount + 1
            endif
        endif
    endwhile

    patternsReport$ = "尖括号标签 (<...>): " + string$(bracketCount) + " 个" +
newline$
    patternsReport$ = patternsReport$ + "laugh: " + string$(laughCount) + " 个" +
newline$
    patternsReport$ = patternsReport$ + "cut: " + string$(cutCount) + " 个" +
newline$
    patternsReport$ = patternsReport$ + "voc/noise: " + string$(noiseCount) + " 个"
+ newline$
    patternsReport$ = patternsReport$ + "error: " + string$(errorCount) + " 个" +
newline$
    patternsReport$ = patternsReport$ + "include -: " + string$(dashCount) + " 个"
+ newline$
    patternsReport$ = patternsReport$ + "special characters: " +
specialCharacters$ + newline$
endproc
```

Appendix 4:clean_stan_collection.praat

```
# Clean stan tier from Collection

folderPath$ = "scripts/"
collectionFile$ = folderPath$ + "praat.Collection"

# read Collection
Read from file: collectionFile$

# get all objects
select all
numberOfObjects = numberOfSelected()
writeInfoLine: "Collection include ", numberOfObjects, " objects"

# get object ID
for i from 1 to numberOfObjects
	objectIDs[i] = selected(i)
endfor

# read TextGrid
textGridCount = 0
for objectIndex from 1 to numberOfObjects
	selectObject: objectIDs[objectIndex]
	objectType$ = extractWord$(selected$(), "")

	if objectType$ = "TextGrid"
		textGridCount = textGridCount + 1
		objectName$ = selected$()

		writeInfoLine: ""
		writeInfoLine: "process TextGrid ", textGridCount, ": ", objectName$

		# find tier9 (stan)
		numTiers = Get number of tiers
		stanTier = 0

		for tierNum from 1 to numTiers
			tierName$ = Get tier name: tierNum
			if (tierNum = 9) or (tierName$ = "stan")
				stanTier = tierNum
				writeInfoLine: "    find stan tier: ", tierName$, " (tier ", tierNum,
")"
				goto foundStanTier
			endif
```

```
endfor

writeInfoLine: "    skip: no tier9 stan found"
goto nextTextGrid

label foundStanTier

# create segment version
numIntervals = Get number of intervals: stanTier

# clean text, keep interval
for intNum from 1 to numIntervals
	originalText$ = Get label of interval: stanTier, intNum

	if originalText$ <> ""
		cleanedText$ = originalText$

		# clean labels
		while index(cleanedText$, "<") > 0 and index(cleanedText$, ">") > 0
			startPos = index(cleanedText$, "<")
			endPos = index(cleanedText$, ">")

			if endPos > startPos
				bracketContent$ = mid$(cleanedText$, startPos + 1, endPos - startPos - 1)

				# check transcript within label
				extractedText$ = ""
				if index(bracketContent$, "-") > 0
					# process LAUGH-i-love-it
					dashPos = index(bracketContent$, "-")
					afterDash$ = right$(bracketContent$, length(bracketContent$) - dashPos)

					if index(afterDash$, "=") > 0
						# process f=for
						equalPos = index(afterDash$, "=")
						extractedText$ = right$(afterDash$, length(afterDash$) - equalPos)
					else
						# process"_", it's_hilarious
						if index(afterDash$, "_") > 0
							# replace with space
```

```
                        extractedText$ = replace$(afterDash$,
"_", " ", 0)
                        extractedText$ =
replace$(extractedText$, "-", " ", 0)
                    else
                        extractedText$ = replace$(afterDash$, "-
", " ", 0)
                    endif
                endif
            endif

            # replace"<>"
            beforeBracket$ = left$(cleanedText$, startPos - 1)
            afterBracket$ = right$(cleanedText$,
length(cleanedText$) - endPos)
            cleanedText$ = beforeBracket$ + " " + extractedText$ +
" " + afterBracket$
        else
            goto endBracketLoop
        endif
    endwhile

    label endBracketLoop

    # delect other"<>"
    cleanedText$ = replace$(cleanedText$, "<", " ", 0)
    cleanedText$ = replace$(cleanedText$, ">", " ", 0)

    # delete other symbol
    cleanedText$ = replace$(cleanedText$, ".", " ", 0)
    cleanedText$ = replace$(cleanedText$, ",", " ", 0)
    cleanedText$ = replace$(cleanedText$, "!", " ", 0)
    cleanedText$ = replace$(cleanedText$, "?", " ", 0)
    cleanedText$ = replace$(cleanedText$, ":", " ", 0)
    cleanedText$ = replace$(cleanedText$, ";", " ", 0)
    cleanedText$ = replace$(cleanedText$, "(", " ", 0)
    cleanedText$ = replace$(cleanedText$, ")", " ", 0)
    cleanedText$ = replace$(cleanedText$, "[", " ", 0)
    cleanedText$ = replace$(cleanedText$, "]", " ", 0)
    cleanedText$ = replace$(cleanedText$, "*", " ", 0)
    cleanedText$ = replace$(cleanedText$, "&", " ", 0)
    cleanedText$ = replace$(cleanedText$, "#", " ", 0)
    cleanedText$ = replace$(cleanedText$, "@", " ", 0)
```

```
            # clean redundant space
            while index(cleanedText$, "    ") > 0
                cleanedText$ = replace$(cleanedText$, "    ", " ", 0)
            endwhile

            while left$(cleanedText$, 1) = " " and length(cleanedText$) > 0
                cleanedText$ = right$(cleanedText$, length(cleanedText$) -
1)
            endwhile
            while right$(cleanedText$, 1) = " " and length(cleanedText$) > 0
                cleanedText$ = left$(cleanedText$, length(cleanedText$) - 1)
            endwhile

            # set text after cleaning
            Set interval text: stanTier, intNum, cleanedText$

            if originalText$ <> cleanedText$
                writeInfoLine: "        Interval ", intNum, ": """, originalText$,
""" → """, cleanedText$, """"
            endif
        endif
    endfor

    # rename tier 9 to stanSeg
    Set tier name: stanTier, "stanSeg"

    # create unsegment version
    # collect text after cleaning
    allCleanText$ = ""
    for intNum from 1 to numIntervals
        intervalText$ = Get label of interval: stanTier, intNum

        if intervalText$ <> ""
            if allCleanText$ = ""
                allCleanText$ = intervalText$
            else
                allCleanText$ = allCleanText$ + " " + intervalText$
            endif
        endif
    endfor

    # create unseg tier
    totalStartTime = Get start time
    totalEndTime = Get end time
```

```
        Insert interval tier: stanTier + 1, "stanUnseg"
        unsegmentedTier = stanTier + 1
        Set interval text: unsegmentedTier, 1, allCleanText$

        label nextTextGrid
    endif
endfor

# save Collection
writeInfoLine: "saving Collection..."
Write to binary file: collectionFile$

writeInfoLine: "completed"
```

Appendix 5: werstep1.praat

```
# WER Analysis Script
# Compare tier8 (ASR) vs tier11 (stanUnseg)

folderPath$ = "scripts/"
collectionFile$ = folderPath$ + "praat.Collection"
outputFile$ = "output_data/wer_detailed_results.txt"

# read Collection
Read from file: collectionFile$

# get all objects
select all
numberOfObjects = numberOfSelected()
writeInfoLine: "Collection contains ", numberOfObjects, " objects"

# get all objects ID
for i from 1 to numberOfObjects
	objectIDs[i] = selected(i)
endfor

# initialise
totalFiles = 0
totalWords = 0
totalErrors = 0
totalDeletions = 0
totalInsertions = 0
totalSubstitutions = 0

# creat output file
deleteFile: outputFile$
fileappend "'outputFile$'" WER analysis results with word details'newline$'
fileappend "'outputFile$'" Time: 'date$()''newline$'
fileappend "'outputFile$'" 'newline$'

# process every TextGrid
writeInfoLine: "Starting to process ", numberOfObjects, " objects..."
for objectIndex from 1 to numberOfObjects
	selectObject: objectIDs[objectIndex]
	objectType$ = extractWord$(selected$(), "")
	writeInfoLine: "Object ", objectIndex, " type: ", objectType$

	if objectType$ = "TextGrid"
		totalFiles = totalFiles + 1
```

```
	objectName$ = selected$()

	writeInfoLine: ""
	writeInfoLine: "Processing file ", totalFiles, ": ", objectName$
	fileappend "'outputFile$'" File: 'objectName$''newline$'

	# get tier number
	numTiers = Get number of tiers

	# find tier
	asrTier = 0
	stanTier = 0

	for tierNum from 1 to numTiers
		tierName$ = Get tier name: tierNum
		if tierNum = 8 or tierName$ = "ASR"
			asrTier = tierNum
			writeInfoLine: "    Found ASR tier: ", tierName$, " (tier ", tierNum, ")"
		endif
		if tierNum = 11 or tierName$ = "stanUnseg"
			stanTier = tierNum
			writeInfoLine: "    Found stanUnseg tier: ", tierName$, " (tier ", tierNum, ")"
		endif
	endfor

	if asrTier = 0 or stanTier = 0
		writeInfoLine: "    Skipped: Missing required tiers"
		fileappend "'outputFile$'"      Error: Missing ASR tier or stanUnseg tier'newline$'
		fileappend "'outputFile$'" 'newline$'
		goto nextTextGrid
	endif

	# get text
	asrIntervals = Get number of intervals: asrTier
	stanIntervals = Get number of intervals: stanTier

	# merge all text
	asrText$ = ""
	stanText$ = ""

	# merge ASR tier text
```

```
for intNum from 1 to asrIntervals
	intervalText$ = Get label of interval: asrTier, intNum
	if intervalText$ <> ""
		if asrText$ <> ""
			asrText$ = asrText$ + " " + intervalText$
		else
			asrText$ = intervalText$
		endif
	endif
endfor

# merge stan tier text
for intNum from 1 to stanIntervals
	intervalText$ = Get label of interval: stanTier, intNum
	if intervalText$ <> ""
		if stanText$ <> ""
			stanText$ = stanText$ + " " + intervalText$
		else
			stanText$ = intervalText$
		endif
	endif
endfor

# output comprision
fileappend "'outputFile$'"	ASR Text:		"'asrText$'"'newline$'
fileappend "'outputFile$'"	Standard Text: "'stanText$'"'newline$'
fileappend "'outputFile$'" 'newline$'

# segment by space
asrWords = 0
stanWords = 0

# segment(ASR)
if asrText$ <> ""
	# by space
	asrTextCopy$ = asrText$
	while index(asrTextCopy$, " ") > 0
		spacePos = index(asrTextCopy$, " ")
		asrWords = asrWords + 1
		asrWord$[asrWords] = left$(asrTextCopy$, spacePos - 1)
		asrTextCopy$ = right$(asrTextCopy$, length(asrTextCopy$) -
spacePos)
	endwhile
	# last word
```

```
		if asrTextCopy$ <> ""
			asrWords = asrWords + 1
			asrWord$[asrWords] = asrTextCopy$
		endif
	endif

	# segment(stan)
	if stanText$ <> ""
		# by space
		stanTextCopy$ = stanText$
		while index(stanTextCopy$, " ") > 0
			spacePos = index(stanTextCopy$, " ")
			stanWords = stanWords + 1
			stanWord$[stanWords] = left$(stanTextCopy$, spacePos - 1)
			stanTextCopy$ = right$(stanTextCopy$, length(stanTextCopy$) - 
spacePos)
		endwhile
		# last word
		if stanTextCopy$ <> ""
			stanWords = stanWords + 1
			stanWord$[stanWords] = stanTextCopy$
		endif
	endif

	writeInfoLine: "    ASR word count: ", asrWords
	writeInfoLine: "    Standard word count: ", stanWords
	fileappend "'outputFile$'      ASR word count: 'asrWords''newline$'
	fileappend "'outputFile$'      Standard word count: 'stanWords''newline$'

	# output comprision
	asrWordList$ = ""
	stanWordList$ = ""

	for w from 1 to asrWords
		if w > 1
			asrWordList$ = asrWordList$ + " | "
		endif
		asrWordList$ = asrWordList$ + asrWord$[w]
	endfor

	for w from 1 to stanWords
		if w > 1
			stanWordList$ = stanWordList$ + " | "
		endif
```

```
	stanWordList$ = stanWordList$ + stanWord$[w]
endfor

fileappend "'outputFile$'"	ASR Words:	['asrWordList$']'newline$'
fileappend "'outputFile$'"	Standard Words:['stanWordList$']'newline$'
fileappend "'outputFile$'" 'newline$'

# calculate edit distance(word level)
asrLen = asrWords
stanLen = stanWords

if asrLen = 0 and stanLen = 0
	writeInfoLine: "	Skipped: Both tiers are empty"
	fileappend "'outputFile$'"	Skipped: Both tiers are empty'newline$'
	fileappend "'outputFile$'" 'newline$'
	goto nextTextGrid
endif

for i from 0 to asrLen
	for j from 0 to stanLen
		dp[i, j] = 0
	endfor
endfor

for i from 0 to asrLen
	dp[i, 0] = i
endfor
for j from 0 to stanLen
	dp[0, j] = j
endfor

for i from 1 to asrLen
	asrCurrentWord$ = asrWord$[i]
	for j from 1 to stanLen
		stanCurrentWord$ = stanWord$[j]

		if asrCurrentWord$ = stanCurrentWord$
			dp[i, j] = dp[i-1, j-1]
		else

			deletion = dp[i-1, j] + 1
			insertion = dp[i, j-1] + 1
			substitution = dp[i-1, j-1] + 1
```

```
            if deletion <= insertion and deletion <= substitution
                dp[i, j] = deletion
            elsif insertion <= substitution
                dp[i, j] = insertion
            else
                dp[i, j] = substitution
            endif
        endif
    endfor
endfor

editDistance = dp[asrLen, stanLen]
referenceLength = stanLen
if referenceLength > 0
    werPercent = (editDistance / referenceLength) * 100
else
    werPercent = 0
endif

# error types and specific error details
i = asrLen
j = stanLen
fileDeletions = 0
fileInsertions = 0
fileSubstitutions = 0

# details error information
errorCount = 0

fileappend "'outputFile$'"      Error analysis'newline$'

while i > 0 or j > 0
    if i > 0 and j > 0
        asrCurrentWord$ = asrWord$[i]
        stanCurrentWord$ = stanWord$[j]

        if asrCurrentWord$ = stanCurrentWord$
            i = i - 1
            j = j - 1
        elsif dp[i, j] = dp[i-1, j-1] + 1
            # Substitution
            fileSubstitutions = fileSubstitutions + 1
            errorCount = errorCount + 1
```

```
					fileappend "'outputFile$'"	Error 'errorCount': 
SUBSTITUTION - ASR:"'asrCurrentWord$'" → Standard:"'stanCurrentWord$'" 
(position 'j')'newline$'
					i = i - 1
					j = j - 1
				elsif dp[i, j] = dp[i-1, j] + 1
					# Insertion (ASR has extra word)
					fileInsertions = fileInsertions + 1
					errorCount = errorCount + 1
					fileappend "'outputFile$'"	Error 'errorCount': 
INSERTION - ASR extra word:"'asrCurrentWord$'" (after position 'j')'newline$'
					i = i - 1
				else
					# Deletion (ASR missing word from standard)
					fileDeletions = fileDeletions + 1
					errorCount = errorCount + 1
					fileappend "'outputFile$'"	Error 'errorCount': DELETION 
- Missing standard word:"'stanCurrentWord$'" (position 'j')'newline$'
					j = j - 1
				endif
			elsif i > 0
				# Insertion (remaining ASR words)
				fileInsertions = fileInsertions + 1
				errorCount = errorCount + 1
				asrCurrentWord$ = asrWord$[i]
				fileappend "'outputFile$'"	Error 'errorCount': INSERTION - 
ASR extra word:"'asrCurrentWord$'" (at end)'newline$'
				i = i - 1
			else
				# Deletion (remaining standard words)
				fileDeletions = fileDeletions + 1
				errorCount = errorCount + 1
				stanCurrentWord$ = stanWord$[j]
				fileappend "'outputFile$'"	Error 'errorCount': DELETION - 
Missing standard word:"'stanCurrentWord$'" (at beginning)'newline$'
				j = j - 1
			endif
		endwhile

		if errorCount = 0
			fileappend "'outputFile$'"	No errors detected - Perfect 
match!'newline$'
		endif
```

```
        # output
        writeInfoLine: "    WER: ", fixed$(werPercent, 2), "%"
        writeInfoLine: "    Error count: ", editDistance, "/", referenceLength
        writeInfoLine: "    Deletions: ", fileDeletions, ", Insertions: ", fileInsertions,
", Substitutions: ", fileSubstitutions

        # culculate
        werString$ = fixed$(werPercent, 2)
        fileappend "'outputFile$'" 'newline$'
        fileappend "'outputFile$'"    Summary 'newline$'
        fileappend "'outputFile$'"    WER: 'werString$'%'newline$'
        fileappend "'outputFile$'"    Edit distance: 'editDistance' / Reference
length: 'referenceLength''newline$'
        fileappend "'outputFile$'"    Error type distribution:'newline$'
        fileappend "'outputFile$'"      Deletions: 'fileDeletions''newline$'
        fileappend "'outputFile$'"      Insertions: 'fileInsertions''newline$'
        fileappend "'outputFile$'"      Substitutions: 'fileSubstitutions''newline$'
        fileappend "'outputFile$'" 'newline$'
        fileappend "'outputFile$'"
========================================'newline$'
        fileappend "'outputFile$'" 'newline$'

        totalWords = totalWords + referenceLength
        totalErrors = totalErrors + editDistance
        totalDeletions = totalDeletions + fileDeletions
        totalInsertions = totalInsertions + fileInsertions
        totalSubstitutions = totalSubstitutions + fileSubstitutions

        label nextTextGrid
    endif
endfor

if totalWords > 0
    overallWER = (totalErrors / totalWords) * 100
else
    overallWER = 0
endif

# Output overall results
writeInfoLine: ""
writeInfoLine: "Overall Statistics"
writeInfoLine: "Files processed: ", totalFiles
writeInfoLine: "Total words: ", totalWords
writeInfoLine: "Total errors: ", totalErrors
```

```
writeInfoLine: "Overall WER: ", fixed$(overallWER, 2), "%"
writeInfoLine: "Error type distribution:"
writeInfoLine: "    Deletions: ", totalDeletions, " (",
fixed$(totalDeletions/totalErrors*100, 1), "%)"
writeInfoLine: "    Insertions: ", totalInsertions, " (",
fixed$(totalInsertions/totalErrors*100, 1), "%)"
writeInfoLine: "    Substitutions: ", totalSubstitutions, " (",
fixed$(totalSubstitutions/totalErrors*100, 1), "%)"

deletionPercent$ = fixed$(totalDeletions/totalErrors*100, 1)
insertionPercent$ = fixed$(totalInsertions/totalErrors*100, 1)
substitutionPercent$ = fixed$(totalSubstitutions/totalErrors*100, 1)
overallWERString$ = fixed$(overallWER, 2)

fileappend "'outputFile$'" overall statistics 'newline$'
fileappend "'outputFile$'" Files processed: 'totalFiles''newline$'
fileappend "'outputFile$'" Total words: 'totalWords''newline$'
fileappend "'outputFile$'" Total errors: 'totalErrors''newline$'
fileappend "'outputFile$'" Overall WER: 'overallWERString$'%'newline$'
fileappend "'outputFile$'" Error type distribution:'newline$'
fileappend "'outputFile$'"      Deletions: 'totalDeletions'
('deletionPercent$'%)'newline$'
fileappend "'outputFile$'"      Insertions: 'totalInsertions'
('insertionPercent$'%)'newline$'
fileappend "'outputFile$'"      Substitutions: 'totalSubstitutions'
('substitutionPercent$'%)'newline$'

appendFileLine: outputFile$, ""
appendFileLine: outputFile$, "completed"

select all
Remove

writeInfoLine: ""
writeInfoLine: "Results saved to: ", outputFile$
writeInfoLine: "WER Analysis completed"
```

Appendix 6: parastep2.praat

```
# Paralinguistic features analysis - Step 2: Error-feature relationship analysis

folderPath$ = "scripts/"
collectionFile$ = folderPath$ + "praat.Collection"
outputFile$ = "output_data/paralinguistic_analysis_results.txt"

# Time window settings (seconds)
beforeWindow = 2.0
afterWindow = 2.0

writeInfoLine: "Paralinguistic features analysis"
writeInfoLine: "Collection file: ", collectionFile$

# Read Collection
Read from file: collectionFile$

# Get all objects
select all
numberOfObjects = numberOfSelected()
writeInfoLine: "Collection contains ", numberOfObjects, " objects"

# Get object ID list
for i from 1 to numberOfObjects
	objectIDs[i] = selected(i)
endfor

# Initialize statistics
totalFiles = 0
totalErrorIntervals = 0
featureBefore = 0
featureDuring = 0
featureAfter = 0
noFeature = 0

# Create results file
deleteFile: outputFile$
fileappend "'outputFile$'" Paralinguistic features analysis results'newline$'
fileappend "'outputFile$'" Time: 'date$()''newline$'
fileappend "'outputFile$'" Before window: 'beforeWindow's, After window:
'afterWindow's'newline$'
fileappend "'outputFile$'" 'newline$'

# Process each TextGrid
```

```
for objectIndex from 1 to numberOfObjects
	selectObject: objectIDs[objectIndex]
	objectType$ = extractWord$(selected$(), "")

	if objectType$ = "TextGrid"
		totalFiles = totalFiles + 1
		objectName$ = selected$("TextGrid")

		writeInfoLine: "Processing file ", totalFiles, ": ", objectName$
		fileappend "'outputFile$'" File: TextGrid 'objectName$''newline$'

		numTiers = Get number of tiers

		# Find target tiers
		paraTier = 0
		asrSegTier = 0
		stanSegTier = 0

		for tierNum from 1 to numTiers
			tierName$ = Get tier name: tierNum
			if tierNum = 7
				paraTier = tierNum
			elif tierNum = 9
				asrSegTier = tierNum
			elif tierNum = 10
				stanSegTier = tierNum
			endif
		endfor

		if asrSegTier = 0 or stanSegTier = 0
			writeInfoLine: "    Skipped: Missing required segmented tiers"
			fileappend "'outputFile$'"      Skipped: Missing required tiers'newline$'
			goto nextFile
		endif

		# Get paralinguistic features from tier 7
		paraFeatures = 0
		if paraTier > 0
			paraIntervals = Get number of intervals: paraTier
			writeInfoLine: "    Found paralinguistic tier with ", paraIntervals, "
intervals"
			for intNum from 1 to paraIntervals
				intervalText$ = Get label of interval: paraTier, intNum
				if intervalText$ <> ""
```

```
					paraFeatures = paraFeatures + 1
					paraStart[paraFeatures] = Get start time of interval: paraTier,
intNum
					paraEnd[paraFeatures] = Get end time of interval: paraTier,
intNum
					paraLabel$[paraFeatures] = intervalText$
					writeInfoLine: "        Feature ", paraFeatures, ": '",
intervalText$, "' from ", paraStart[paraFeatures], " to ", paraEnd[paraFeatures]
				endif
			endfor
		else
			writeInfoLine: "    No paralinguistic tier found"
		endif
		writeInfoLine: "    Total paralinguistic features: ", paraFeatures

		# Compare segmented tiers interval by interval
		asrSegIntervals = Get number of intervals: asrSegTier
		stanSegIntervals = Get number of intervals: stanSegTier

		# Check if tiers have same number of intervals (should be aligned)
		if asrSegIntervals <> stanSegIntervals
			writeInfoLine: "    Warning: Different number of intervals - ASR: ",
asrSegIntervals, ", Standard: ", stanSegIntervals
		endif

		minIntervals = min(asrSegIntervals, stanSegIntervals)

		# Initialize file counters
		fileErrorIntervals = 0
		fileFeatureBefore = 0
		fileFeatureDuring = 0
		fileFeatureAfter = 0
		fileNoFeature = 0

		fileappend "'outputFile$'"      Comparing 'minIntervals' aligned
intervals:'newline$'

		# Compare each aligned interval
		for intervalNum from 1 to minIntervals
			asrText$ = Get label of interval: asrSegTier, intervalNum
			stanText$ = Get label of interval: stanSegTier, intervalNum

			# Skip empty intervals (unless both are different types of empty)
			if asrText$ <> stanText$
```

```
                    # Found an error interval
                    fileErrorIntervals = fileErrorIntervals + 1

                    # Get timing of this interval
                    errorStart = Get start time of interval: stanSegTier, intervalNum
                    errorEnd = Get end time of interval: stanSegTier, intervalNum

                    writeInfoLine: "    Error interval ", intervalNum, ": ASR='", 
asrText$, "' vs Standard='", stanText$, "' (", errorStart, "-", errorEnd, ")"
                    fileappend "'outputFile$'"      Error 'fileErrorIntervals': Interval 
'intervalNum' | ASR:"'asrText$'" vs Standard:"'stanText$'" | Time: 'errorStart:3'-
'errorEnd:3's'newline$'

                    # Analyze relationship with paralinguistic features
                    call classifyErrorRelationship errorStart errorEnd

                endif
            endfor

            if fileErrorIntervals = 0
                fileappend "'outputFile$'"      No errors detected - Perfect 
alignment!'newline$'
            endif

            # Output file statistics
            writeInfoLine: "    File results: Errors=", fileErrorIntervals, " | Before=", 
fileFeatureBefore, " | During=", fileFeatureDuring, " | After=", fileFeatureAfter, " | 
None=", fileNoFeature
            fileappend "'outputFile$'"      Errors: 'fileErrorIntervals''newline$'
            fileappend "'outputFile$'"      Before: 'fileFeatureBefore''newline$'
            fileappend "'outputFile$'"      During: 'fileFeatureDuring''newline$'
            fileappend "'outputFile$'"      After: 'fileFeatureAfter''newline$'
            fileappend "'outputFile$'"      None: 'fileNoFeature''newline$'
            fileappend "'outputFile$'" 'newline$'

            # Update overall statistics
            totalErrorIntervals = totalErrorIntervals + fileErrorIntervals
            featureBefore = featureBefore + fileFeatureBefore
            featureDuring = featureDuring + fileFeatureDuring
            featureAfter = featureAfter + fileFeatureAfter
            noFeature = noFeature + fileNoFeature

            label nextFile
        endif
```

```
endfor

# Calculate overall percentages
if totalErrorIntervals > 0
	beforePercent = (featureBefore / totalErrorIntervals) * 100
	duringPercent = (featureDuring / totalErrorIntervals) * 100
	afterPercent = (featureAfter / totalErrorIntervals) * 100
	noFeaturePercent = (noFeature / totalErrorIntervals) * 100
else
	beforePercent = 0
	duringPercent = 0
	afterPercent = 0
	noFeaturePercent = 0
endif

# Output overall results
writeInfoLine: ""
writeInfoLine: "final results"
writeInfoLine: "Files processed: ", totalFiles
writeInfoLine: "Total error intervals: ", totalErrorIntervals
writeInfoLine: "Before: ", featureBefore, " (", fixed$(beforePercent, 1), "%)"
writeInfoLine: "During: ", featureDuring, " (", fixed$(duringPercent, 1), "%)"
writeInfoLine: "After: ", featureAfter, " (", fixed$(afterPercent, 1), "%)"
writeInfoLine: "None: ", noFeature, " (", fixed$(noFeaturePercent, 1), "%)"

fileappend "'outputFile$'" final results'newline$'
fileappend "'outputFile$'" Files processed: 'totalFiles''newline$'
fileappend "'outputFile$'" Total error intervals: 'totalErrorIntervals''newline$'
fileappend "'outputFile$'" Before: 'featureBefore' ('beforePercent:1'%)'newline$'
fileappend "'outputFile$'" During: 'featureDuring' ('duringPercent:1'%)'newline$'
fileappend "'outputFile$'" After: 'featureAfter' ('afterPercent:1'%)'newline$'
fileappend "'outputFile$'" None: 'noFeature' ('noFeaturePercent:1'%)'newline$'

writeInfoLine: ""
writeInfoLine: "Results saved to: ", outputFile$

# Clean up all objects
writeInfoLine: "Cleaning up objects..."
select all
Remove

writeInfoLine: "completed"

# start process
```

```
procedure classifyErrorRelationship .errorStart .errorEnd
    relationship$ = "no_feature"

    writeInfoLine: "        Analyzing error at ", .errorStart, "-", .errorEnd, " with ", paraFeatures, " features"

    if paraFeatures > 0
        for pf from 1 to paraFeatures
            writeInfoLine: "            Feature ", pf, ": '", paraLabel$[pf], "' (", paraStart[pf], "-", paraEnd[pf], ")"

            # Check overlap (during)
            if .errorStart < paraEnd[pf] and .errorEnd > paraStart[pf]
                relationship$ = "feature_during"
                writeInfoLine: "            -> DURING: Time overlap detected"
                goto endFeatureCheck
            endif

            # Check before window
            if .errorEnd <= paraStart[pf] and paraStart[pf] - .errorEnd <= beforeWindow
                relationship$ = "feature_before"
                gap = paraStart[pf] - .errorEnd
                writeInfoLine: "            -> BEFORE: Gap = ", gap, "s (within ", beforeWindow, "s window)"
                goto endFeatureCheck
            endif

            # Check after window
            if .errorStart >= paraEnd[pf] and .errorStart - paraEnd[pf] <= afterWindow
                relationship$ = "feature_after"
                gap = .errorStart - paraEnd[pf]
                writeInfoLine: "            -> AFTER: Gap = ", gap, "s (within ", afterWindow, "s window)"
                goto endFeatureCheck
            endif
        endfor
        label endFeatureCheck
    else
        writeInfoLine: "        -> No paralinguistic features to compare"
    endif
```

```
	writeInfoLine: "	Final classification: ", relationship$

	# Update file counters
	if relationship$ = "feature_before"
		fileFeatureBefore = fileFeatureBefore + 1
	elsif relationship$ = "feature_during"
		fileFeatureDuring = fileFeatureDuring + 1
	elsif relationship$ = "feature_after"
		fileFeatureAfter = fileFeatureAfter + 1
	else
		fileNoFeature = fileNoFeature + 1
	endif

	fileappend "'outputFile$'"	-> Relationship: 'relationship$''newline$'
endproc


writeInfoLine: ""
writeInfoLine: "Results saved to: ", outputFile$
writeInfoLine: "para relationship completed"
```

Appendix 7: aacoustic feature.praat

```
# Script for analyzing paralinguistic features(HNR, Spectral Tilt, and ZCR) for
specific labeled intervals in tier 7 (“para”).

form Analyze paralinguistic features
	text input_directory scripts
	text output_file	output_data/acoustic_feature_results.txt
endform

#	Create output file and write header
writeFileLine: output_file$, "File | Label | Start | End | Mean_HNR | Spectral_Tilt |
ZCR"

# get all wav
Create Strings as file list: "fileList", input_directory$ + "/*.wav"
numberOfFiles = Get number of strings

for i from 1 to numberOfFiles
	select Strings fileList
	filename$ = Get string: i
	basename$ = filename$ - ".wav"

	# read sound and textgrid
	Read from file: input_directory$ + "/" + filename$
	sound = selected("Sound")
	Read from file: input_directory$ + "/" + basename$ + ".TextGrid"
	textgrid = selected("TextGrid")

	# get number of interval in tier 7
	select textgrid
	numberOfIntervals = Get number of intervals: 7
	  # Process each interval in tier 7
	for interval to numberOfIntervals
		select textgrid
		label$ = Get label of interval: 7, interval

		if label$ = "laughter" or label$ = "throat-clearing" or label$ = "sigh"
			start = Get start time of interval: 7, interval
			end = Get end time of interval: 7, interval
			duration = end - start

			# Extract sound segment
			select sound
			Extract part: start, end, "rectangular", 1, "no"
```

```
            segment = selected("Sound")

            # Calculate HNR
            To Harmonicity (cc): 0.01, 75, 0.1, 4.5
            meanHNR = Get mean: 0, 0
            Remove

            # Calculate Spectral Tilt
            select segment
            To Spectrum: "yes"
            spectrum = selected("Spectrum")
            select spectrum
            To Ltas (1-to-1)
            ltas = selected("Ltas")
            select ltas
            # Get slope between low band (0-1000 Hz) and high band (1000-4000
Hz)
            lowBand = Get mean: 0, 1000, "energy"
            highBand = Get mean: 1000, 4000, "energy"
            spectralTilt = 10 * log10(highBand / lowBand)
        removeObject: spectrum, ltas


            # Calculate ZCR
            select segment
            To PointProcess (zeroes): 1, "yes", "yes"
            points = Get number of points
            zcr = points / duration
            Remove

             # Write results to file
            resultLine$ = basename$ + tab$ + label$ + tab$ + fixed$(start, 3) +
tab$ +
            ... fixed$(end, 3) + tab$ + fixed$(meanHNR, 3) + tab$ +
            ... fixed$(spectralTilt, 3) + tab$ + fixed$(zcr, 3)
            appendFileLine: output_file$, resultLine$

            select segment
            removeObject: segment
        endif
    endfor
    removeObject: sound, textgrid
endfor
```

```
select Strings fileList
Remove

# Clean up any remaining objects
writeInfoLine: "Cleaning up objects..."
select all
numberOfRemainingObjects = numberOfSelected()
if numberOfRemainingObjects > 0
	writeInfoLine: "Removing ", numberOfRemainingObjects, " remaining objects"
	Remove
endif

writeInfoLine: "complete"
```

Appendix 8: master_table.praat

```
# Master analysis table
# Combines WER + Paralinguistic + Acoustic data

folderPath$ = "praat_analysis/output_data/"
outputFile$ = "descriptive_stat/master_analysis_table.tsv"

# Arrays for master data
fileCount = 0

#WER
werLines = Read Strings from raw text file: folderPath$ + "wer_detailed_results.txt"
numberOfLines = Get number of strings

for line from 1 to numberOfLines
	text$ = Get string: line

	# Get file name
	if index(text$, "File: TextGrid ") > 0
		fileCount = fileCount + 1
		start = index(text$, "TextGrid ") + 9
		fileName$[fileCount] = right$(text$, length(text$) - start + 1)
	endif

	# Get WER percentage
	if index(text$, "WER: ") > 0 and fileCount > 0
		start = index(text$, "WER: ") + 5
		end = index(text$, "%")
		werPercent[fileCount] = number(mid$(text$, start, end - start))
	endif

	# Get error counts in one go
	if index(text$, "Deletions: ") > 0 and fileCount > 0
		# Extract all three numbers from this section
		deletions[fileCount] = number(mid$(text$, index(text$, "Deletions: ") + 11,
3))
	endif
	if index(text$, "Insertions: ") > 0 and fileCount > 0
		insertions[fileCount] = number(mid$(text$, index(text$, "Insertions: ") +
12, 3))
	endif
	if index(text$, "Substitutions: ") > 0 and fileCount > 0
		substitutions[fileCount] = number(mid$(text$, index(text$, "Substitutions:
") + 15, 3))
```

```
		totalErrors[fileCount] = deletions[fileCount] + insertions[fileCount] +
substitutions[fileCount]
	endif
endfor

select werLines
Remove

#Paralinguistic
paraLines = Read Strings from raw text file: folderPath$ +
"paralinguistic_analysis_results.txt"
numberOfLines = Get number of strings

# Initialize para arrays
for i from 1 to fileCount
	before[i] = 0
	during[i] = 0
	after[i] = 0
	none[i] = 0
endfor

currentFile = 0
for line from 1 to numberOfLines
	text$ = Get string: line

	# Find current file
	if index(text$, "File: TextGrid ") > 0
		start = index(text$, "TextGrid ") + 9
		currentFileName$ = right$(text$, length(text$) - start + 1)

		# Match with existing files
		currentFile = 0
		for i from 1 to fileCount
			if fileName$[i] = currentFileName$
				currentFile = i
				goto foundFile
			endif
		endfor
		label foundFile
	endif

	# Extract individual values
	if currentFile > 0
		if index(text$, "    Before: ") > 0
```

```
            start = index(text$, "Before: ") + 8
            before[currentFile] = number(right$(text$, length(text$) - start + 1))
        endif
        if index(text$, "    During: ") > 0
            start = index(text$, "During: ") + 8
            during[currentFile] = number(right$(text$, length(text$) - start + 1))
        endif
        if index(text$, "    After: ") > 0
            start = index(text$, "After: ") + 7
            after[currentFile] = number(right$(text$, length(text$) - start + 1))
        endif
        if index(text$, "    None: ") > 0
            start = index(text$, "None: ") + 6
            none[currentFile] = number(right$(text$, length(text$) - start + 1))
        endif
    endif
endfor

select paraLines
Remove

#Acoustic
acousticLines = Read Strings from raw text file: folderPath$ +
"acoustic_feature_results.txt"
numberOfLines = Get number of strings

# Initialize acoustic arrays
for i from 1 to fileCount
    label$[i] = "NA"
    startTime$[i] = "NA"
    endTime$[i] = "NA"
    hnr$[i] = "NA"
    tilt$[i] = "NA"
    zcr$[i] = "NA"
endfor

# Skip header, process data
for line from 2 to numberOfLines
    text$ = Get string: line
    if text$ = ""
        goto nextLine
    endif

    # Simple tab parsing
```

```
	parts = 0
	remaining$ = text$
	while index(remaining$, tab$) > 0
		parts = parts + 1
		pos = index(remaining$, tab$)
		part$[parts] = left$(remaining$, pos - 1)
		remaining$ = right$(remaining$, length(remaining$) - pos)
	endwhile
	parts = parts + 1
	part$[parts] = remaining$

	if parts >= 6
		# Match file name (remove extensions)
		acousticFile$ = replace$(part$[1], ".wav", "", 0)
		acousticFile$ = replace$(acousticFile$, ".TextGrid", "", 0)

		for i from 1 to fileCount
			baseFileName$ = replace$(fileName$[i], ".TextGrid", "", 0)
			if baseFileName$ = acousticFile$ and label$[i] = "NA"
				label$[i] = part$[2]
				startTime$[i] = part$[3]
				endTime$[i] = part$[4]
				hnr$[i] = part$[5]
				tilt$[i] = part$[6]
				if parts >= 7
					zcr$[i] = part$[7]
				endif
				goto nextLine
			endif
		endfor
	endif

	label nextLine
endfor

select acousticLines
Remove


#Write master table
deleteFile: outputFile$

# Header
```

```
appendFileLine: outputFile$, "FileName" + tab$ + "WER_Percent" + tab$ +
"Total_WER_Errors" + tab$ +
... "Deletions" + tab$ + "Insertions" + tab$ + "Substitutions" + tab$ +
... "Feature_Before" + tab$ + "Feature_During" + tab$ + "Feature_After" + tab$ +
"No_Feature" + tab$ +
... "Total_Feature_Errors" + tab$ + "Label" + tab$ + "Start" + tab$ + "End" + tab$ +
... "Mean_HNR" + tab$ + "Spectral_Tilt" + tab$ + "ZCR"

# Data rows
for i from 1 to fileCount
	totalFeature = before[i] + during[i] + after[i] + none[i]

	line$ = fileName$[i] + tab$ +
			... string$(werPercent[i]) + tab$ + string$(totalErrors[i]) + tab$ +
			... string$(deletions[i]) + tab$ + string$(insertions[i]) + tab$ +
string$(substitutions[i]) + tab$ +
			... string$(before[i]) + tab$ + string$(during[i]) + tab$ + string$(after[i])
+ tab$ + string$(none[i]) + tab$ +
			... string$(totalFeature) + tab$ + label$[i] + tab$ + startTime$[i] +
tab$ + endTime$[i] + tab$ +
			... hnr$[i] + tab$ + tilt$[i] + tab$ + zcr$[i]

	appendFileLine: outputFile$, line$
   endfor

writeInfoLine: "complete"
```

Appendix 9: descriptive.stat,rmd

```
---
title: "descriptive_stat"
author: "Xuefeiyang Zhang"
date: "2025-08-12"
output: html_document
---

# Step 1: Event distribution analysis

# Set path
data_dir <- "../descriptive_stat"
output_dir <- "../descriptive_stat/output"

cat("=== DISTRIBUTION OF NUMBER OF EVENTS ===\n")

# Read data
data_file <- file.path(data_dir, "master_analysis_table.tsv")
data <- read.delim(data_file, sep = "\t", header = TRUE, stringsAsFactors = FALSE)

cat("Data loaded:", nrow(data), "observations\n")

# Create frequency table
label_counts <- table(data$Label)

# Convert to data frame and calculate percentages
event_dist <- data.frame(
   Label = names(label_counts),
   n = as.numeric(label_counts),
   stringsAsFactors = FALSE
)

# Calculate percentages
event_dist$percent <- paste0(round(event_dist$n / sum(event_dist$n) * 100, 1), "%")

# Sort by frequency
event_dist <- event_dist[order(event_dist$n, decreasing = TRUE), ]

# Add total row
total_row <- data.frame(
   Label = "Total",
   n = sum(event_dist$n),
   percent = "100%",
   stringsAsFactors = FALSE
```

```
)

# Combine table
table_event_distribution <- rbind(event_dist, total_row)

# Reset row names
rownames(table_event_distribution) <- NULL

# Print and save table
cat("\nEvent Distribution Table:\n")
print(table_event_distribution)
output_file <- file.path(output_dir, "table1_event_distribution.tsv")
write.table(table_event_distribution, output_file, sep = "\t", row.names = FALSE,
quote = FALSE)

cat("\nTable saved to:", output_file, "\n")

# Create bar chart
cat("\nCreating bar chart...\n")

chart_data <- event_dist[event_dist$Label != "Total", ]

#save bar chart
png_file <- file.path(output_dir, "figure1_event_distribution.png")
png(png_file, width = 800, height = 600)

# Set bar chart
max_count <- max(chart_data$n)
y_limit <- max_count * 1.1

barplot(chart_data$n,
        names.arg = chart_data$Label,
        main = "Distribution of Paralinguistic Events",
        xlab = "Event Type",
        ylab = "Count",
        col = "steelblue",
        ylim = c(0,y_limit))

#Add text and labels
text(x = seq_along(chart_data$n) * 1.2 - 0.5,
     y = chart_data$n + max(chart_data$n) * 0.02,
     labels = chart_data$n,
     pos = 3)
```

```r
dev.off()

cat("Bar chart saved to:", png_file, "\n")

# Step 2: WER overview

# Set path
data_dir <- "../descriptive_stat"
output_dir <- "../descriptive_stat/output"

# Read data
data <- read.delim(file.path(data_dir, "master_analysis_table.tsv"), sep = "\t")
data$WER_Percent <- as.numeric(data$WER_Percent)
data <- data[!is.na(data$WER_Percent), ]

# Calculate WER statistics
event_types <- c("laughter", "sigh", "throat-clearing")
wer_stats <- data.frame()

for(label in event_types) {
  wer_values <- data[data$Label == label, "WER_Percent"]
  if(length(wer_values) > 0) {
    stats_row <- data.frame(
      Label = label,
      n = length(wer_values),
      Mean = round(mean(wer_values), 2),
      SD = round(sd(wer_values), 2),
      Median = round(median(wer_values), 2),
      Q1 = round(quantile(wer_values, 0.25), 2),
      Q3 = round(quantile(wer_values, 0.75), 2),
      IQR = round(quantile(wer_values, 0.75) - quantile(wer_values, 0.25), 2)
    )
    wer_stats <- rbind(wer_stats, stats_row)
  }
}

# Print and save table
print(wer_stats)
write.table(wer_stats, file.path(output_dir, "table2_wer_statistics.tsv"),
            sep = "\t", row.names = FALSE, quote = FALSE)

# Create boxplot
boxplot_data <- list()
sample_sizes <- c()
```

```
for(label in event_types) {
  boxplot_data[[label]] <- data[data$Label == label, "WER_Percent"]
  sample_sizes <- c(sample_sizes, sum(data$Label == label))
}

label_with_n <- paste0(event_types, "\n(n=", sample_sizes, ")")

# Save boxplot
png(file.path(output_dir, "figure2_wer_boxplot.png"), width = 800, height = 600)

#set boxplot
boxplot(boxplot_data,
        names = label_with_n,
        main = "WER Distribution by Event Type",
        xlab = "Event Type",
        ylab = "WER (%)",
        col = "steelblue",
        ylim = c(0, 120))
dev.off()

cat("Files saved:\n- table2_wer_statistics.tsv\n- figure2_wer_boxplot.png\n")


# Step 3: error type composition
# Set paths
data_dir <- "../descriptive_stat"
output_dir <- "../descriptive_stat/output"

# Read data
data <- read.delim(file.path(data_dir, "master_analysis_table.tsv"), sep = "\t")
data$Deletions <- as.numeric(data$Deletions)
data$Insertions <- as.numeric(data$Insertions)
data$Substitutions <- as.numeric(data$Substitutions)
data <- data[!is.na(data$Deletions) & !is.na(data$Insertions)
& !is.na(data$Substitutions), ]

# Calculate error composition
event_types <- c("laughter", "sigh", "throat-clearing")
error_composition <- data.frame()

for(label in event_types) {
  label_data <- data[data$Label == label, ]

  if(nrow(label_data) > 0) {
```

```
    total_deletions <- sum(label_data$Deletions, na.rm = TRUE)
    total_insertions <- sum(label_data$Insertions, na.rm = TRUE)
    total_substitutions <- sum(label_data$Substitutions, na.rm = TRUE)
    total_errors <- total_deletions + total_insertions + total_substitutions

    if(total_errors > 0) {
      d_percent <- round((total_deletions / total_errors) * 100, 1)
      i_percent <- round((total_insertions / total_errors) * 100, 1)
      s_percent <- round((total_substitutions / total_errors) * 100, 1)
    } else {
      d_percent <- i_percent <- s_percent <- 0
    }

    composition_row <- data.frame(
      Label = label,
      Deletions = total_deletions,
      Insertions = total_insertions,
      Substitutions = total_substitutions,
      TotalErrors = total_errors,
      D_Percent = d_percent,
      I_Percent = i_percent,
      S_Percent = s_percent
    )

    error_composition <- rbind(error_composition, composition_row)
  }
}

# Print and save table
print(error_composition)
write.table(error_composition, file.path(output_dir, "table3_error_composition.tsv"),
            sep = "\t", row.names = FALSE, quote = FALSE)

# Create stacked bar chart
chart_data <- as.matrix(error_composition[, c("D_Percent", "I_Percent",
"S_Percent")])
rownames(chart_data) <- error_composition$Label

# Save plot
png(file.path(output_dir, "figure3_error_composition.png"), width = 800, height =
600)
barplot(t(chart_data),
        main = "Error Type Composition by Event Type",
        xlab = "Event Type", ylab = "Percentage (%)",
```

```r
        col = c("coral", "steelblue", "lightgreen"),
        legend.text = c("Deletions", "Insertions", "Substitutions"),
        args.legend = list(x = "topright", bty = "n"),
        ylim = c(0, 100), beside = FALSE)
dev.off()

cat("Files saved:\n- table3_error_composition.tsv\n-
figure3_error_composition.png\n")

# Step 4: Temporal position of errors

# Set path
data_dir <- "../descriptive_stat"
output_dir <- "../descriptive_stat/output"

# Read and prepare data
data <- read.delim(file.path(data_dir, "master_analysis_table.tsv"), sep = "\t")
data$Feature_Before <- as.numeric(data$Feature_Before)
data$Feature_During <- as.numeric(data$Feature_During)
data$Feature_After <- as.numeric(data$Feature_After)
data$No_Feature <- as.numeric(data$No_Feature)

# Calculate by event type
event_types <- c("laughter", "sigh", "throat-clearing")
results <- data.frame()

for(label in event_types) {
  label_data <- data[data$Label == label, ]

  before <- sum(label_data$Feature_Before, na.rm = TRUE)
  during <- sum(label_data$Feature_During, na.rm = TRUE)
  after <- sum(label_data$Feature_After, na.rm = TRUE)
  no_feature <- sum(label_data$No_Feature, na.rm = TRUE)
  total <- before + during + after + no_feature

  row <- data.frame(
    Label = label,
    Before = before, During = during, After = after, No_Feature = no_feature,
    Row_Total = total,
    Before_Pct = round((before/total)*100, 1),
    During_Pct = round((during/total)*100, 1),
    After_Pct = round((after/total)*100, 1),
    No_Feature_Pct = round((no_feature/total)*100, 1)
  )
```

```
  results <- rbind(results, row)
}

# Save table
print(results)
write.table(results, file.path(output_dir, "table4_temporal_position.tsv"),
                  sep = "\t", row.names = FALSE, quote = FALSE)

# Create plot
chart_data <- as.matrix(results[, c("Before_Pct", "During_Pct", "After_Pct",
"No_Feature_Pct")])
rownames(chart_data) <- results$Label

png(file.path(output_dir, "figure4_temporal_position.png"), width = 800, height =
600)
barplot(t(chart_data),
            main = "Temporal Position of Errors by Event Type",
            xlab = "Event Type", ylab = "Percentage (%)",
            col = c("lightgreen", "orange", "coral", "lightgray"),
            legend.text = c("Before", "During", "After", "No Feature"),
            args.legend = list(x = "topright", bty = "n"),
            ylim = c(0, 100), beside = FALSE)
dev.off()

cat("Files saved: table4_temporal_position.tsv, figure4_temporal_position.png\n")

# Step 5: Acoustic features vs. error occurrence

# Set path
data_dir <- "../descriptive_stat"
output_dir <- "../descriptive_stat/output"

# Read data
data <- read.delim(file.path(data_dir, "master_analysis_table.tsv"), sep = "\t")

#Prepare data
data$AnyError <- ifelse(data$Total_WER_Errors > 0, 1, 0)

# Filter undefined labels
data <- data[data$Mean_HNR != "--undefined--" &
                  data$Spectral_Tilt != "--undefined--" &
                  data$ZCR != "--undefined--", ]

# Convert to numeric
```

```
data$Mean_HNR <- as.numeric(data$Mean_HNR)
data$Spectral_Tilt <- as.numeric(data$Spectral_Tilt)
data$ZCR <- as.numeric(data$ZCR)
data <- data[!is.na(data$Mean_HNR) & !is.na(data$Spectral_Tilt)
& !is.na(data$ZCR), ]

# Create plots
features <- list(
  list(data = data$Mean_HNR, name = "HNR", ylabel = "HNR (dB)", file =
"figure5_hnr_anyerror.png"),
  list(data = data$Spectral_Tilt, name = "Spectral Tilt", ylabel = "Spectral Tilt (dB)",
file = "figure6_tilt_anyerror.png"),
  list(data = data$ZCR, name = "ZCR", ylabel = "ZCR (Hz)", file =
"figure7_zcr_anyerror.png")
)

for(feature in features) {
  no_error <- feature$data[data$AnyError == 0]
  with_error <- feature$data[data$AnyError == 1]
  plot_data <- list("0" = no_error, "1" = with_error)

  png(file.path(output_dir, feature$file), width = 800, height = 600)
  boxplot(plot_data,
          main = paste("Distribution of", feature$name, "by Error Occurrence"),
          xlab = "Any Error (0 = no error; 1 = at least one error)",
          ylab = feature$ylabel,
          col = c("steelblue", "coral"))
  dev.off()
}

cat("Files saved:\n- figure5_hnr_anyerror.png\n- figure6_tilt_anyerror.png\n-
figure7_zcr_anyerror.png\n")

# Step 6: spearman correlation

# Set path
data_dir <- "../descriptive_stat"
output_dir <- "../descriptive_stat/output"

# Read data
data <- read.delim(file.path(data_dir, "master_analysis_table.tsv"), sep = "\t")

# Prepare data
data <- data[data$Mean_HNR != "--undefined--" &
```

```
                    data$Spectral_Tilt != "--undefined--" &
                    data$ZCR != "--undefined--", ]

data$Mean_HNR <- as.numeric(data$Mean_HNR)
data$Spectral_Tilt <- as.numeric(data$Spectral_Tilt)
data$ZCR <- as.numeric(data$ZCR)
data <- data[!is.na(data$Mean_HNR) & !is.na(data$Spectral_Tilt)
& !is.na(data$ZCR), ]

# Calculate correlation
cor1 <- cor.test(data$Mean_HNR, data$Spectral_Tilt, method = "spearman")
cor2 <- cor.test(data$Mean_HNR, data$ZCR, method = "spearman")
cor3 <- cor.test(data$Spectral_Tilt, data$ZCR, method = "spearman")

# Create table
results <- data.frame(
  Pair = c("HNR — Spectral Tilt", "HNR — ZCR", "Spectral Tilt — ZCR"),
  rho = round(c(cor1$estimate, cor2$estimate, cor3$estimate), 3),
  p = round(c(cor1$p.value, cor2$p.value, cor3$p.value), 3)
)

# Print and save table
print(results)
write.table(results, file.path(output_dir, "table5_spearman_correlations.tsv"),
                  sep = "\t", row.names = FALSE, quote = FALSE)

cat("File saved: table5_spearman_correlations.tsv\n")
```

Appendix 10: regression_model.rmd

```
# Logistic Regression Analysis

library(sandwich)
library(lmtest)

# Set path
data_dir <- "../logistic_model"
output_dir <- "../logistic_model/output"

# Read data
data_file <- file.path(data_dir, "modeling_dataset.tsv")
data <- read.delim(data_file, sep = "\t")
data$Speaker <- as.factor(data$Speaker)

# Fit models
m1 <- glm(AnyError ~ zHNR + zTilt + zZCR + Label, data = data, family =
binomial)
m2 <- glm(DelOccur ~ zHNR + zTilt + zZCR + Label, data = data, family =
binomial)
m3 <- glm(InsOccur ~ zHNR + zTilt + zZCR + Label, data = data, family = binomial)
m4 <- glm(SubOccur ~ zHNR + zTilt + zZCR + Label, data = data, family =
binomial)

# Get robust results for each model
get_results <- function(model, data) {
  robust_se <- vcovCL(model, cluster = data$Speaker)
  robust_test <- coeftest(model, vcov = robust_se)

  coefs <- robust_test[, "Estimate"]
  ses <- robust_test[, "Std. Error"]
  pvals <- robust_test[, "Pr(>|z|)"]

  ors <- exp(coefs)
  ci_low <- exp(coefs - 1.96 * ses)
  ci_high <- exp(coefs + 1.96 * ses)

  results <- data.frame(
    Variable = names(coefs),
    OR = round(ors, 2),
    CI_Low = round(ci_low, 2),
    CI_High = round(ci_high, 2),
    p = round(pvals, 3)
  )
```

```
  results$p[results$p < 0.001] <- "<0.001"
  return(results)
}

# Get results
r1 <- get_results(m1, data)
r2 <- get_results(m2, data)
r3 <- get_results(m3, data)
r4 <- get_results(m4, data)

# Remove intercept
r1 <- r1[r1$Variable != "(Intercept)", ]
r2 <- r2[r2$Variable != "(Intercept)", ]
r3 <- r3[r3$Variable != "(Intercept)", ]
r4 <- r4[r4$Variable != "(Intercept)", ]

# Make table
variables <- r1$Variable
table_result <- data.frame(
  Variable = variables,
  M1_AnyError = paste0(r1$OR, " [", r1$CI_Low, ", ", r1$CI_High, "] (", r1$p, ")"),
  M2_DelOccur = paste0(r2$OR, " [", r2$CI_Low, ", ", r2$CI_High, "] (", r2$p, ")"),
  M3_InsOccur = paste0(r3$OR, " [", r3$CI_Low, ", ", r3$CI_High, "] (", r3$p, ")"),
  M4_SubOccur = paste0(r4$OR, " [", r4$CI_Low, ", ", r4$CI_High, "] (", r4$p, ")")
)

# Clean variable names
table_result$Variable <- gsub("zHNR", "z-HNR", table_result$Variable)
table_result$Variable <- gsub("zTilt", "z-Spectral Tilt", table_result$Variable)
table_result$Variable <- gsub("zZCR", "z-ZCR", table_result$Variable)
table_result$Variable <- gsub("Labelsigh", "Event: sigh", table_result$Variable)
table_result$Variable <- gsub("Labelthroat-clearing", "Event: throat-clearing",
table_result$Variable)

# Print and save table
print(table_result)
write.table(table_result, file.path(output_dir, "table6_logistic_main.tsv"), sep = "\t",
row.names = FALSE, quote = FALSE)

# Make plots
# Plot 1: Insertion by event type
new_data <- data.frame(
  zHNR = 0, zTilt = 0, zZCR = 0,
```

```
  Label = c("laughter", "sigh", "throat-clearing")
)
pred <- predict(m3, newdata = new_data, type = "response")
pred_pct <- pred * 100

png(file.path(output_dir, "figure8_insertion_by_label.png"), width = 800, height =
600)
barplot(pred_pct, names.arg = new_data$Label,
        main = "Predicted Insertion Probability by Event Type",
        xlab = "Event Type", ylab = "Probability (%)",
        col = "steelblue")
dev.off()

# Plot 2: AnyError vs HNR
hnr_values <- seq(-2, 2, by = 0.1)
new_data2 <- data.frame(
  zHNR = hnr_values, zTilt = 0, zZCR = 0, Label = "laughter"
)
pred2 <- predict(m1, newdata = new_data2, type = "response")
pred2_pct <- pred2 * 100

png(file.path(output_dir, "figure9_anyerror_vs_hnr.png"), width = 800, height = 600)
plot(hnr_values, pred2_pct, type = "l", lwd = 2, col = "steelblue",
     main = "Any Error Probability vs HNR",
     xlab = "z-HNR", ylab = "Probability (%)")
dev.off()

# Plot 3: AnyError vs ZCR
zcr_values <- seq(-2, 2, by = 0.1)
new_data3 <- data.frame(
  zHNR = 0, zTilt = 0, zZCR = zcr_values, Label = "laughter"
)
pred3 <- predict(m1, newdata = new_data3, type = "response")
pred3_pct <- pred3 * 100

png(file.path(output_dir, "figure10_anyerror_vs_zcr.png"), width = 800, height = 600)
plot(zcr_values, pred3_pct, type = "l", lwd = 2, col = "steelblue",
     main = "Any Error Probability vs ZCR",
     xlab = "z-ZCR", ylab = "Probability (%)")
dev.off()
```