Universiteit Leiden
The Netherlands

Opleiding Informatica

Sensitivity of Automatic Speech Recognition

on Non-Native Speakers Accents

Thalia Nanhekhan

Supervisor:
Joost Broekens

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

01/06/2025

Abstract

The imitation of human listening and speech comprehension has long been a goal in technology. Speech recognition (SR) refers to systems that recognize patterns in sound waves and convert them into text. Despite rapid advances, previous research has shown that Automatic Speech Recognition (ASR) technology remains uneven in accuracy across different user groups. This study investigates how sensitive ASR models are to non-native speakers with diverse accents. Three models, NVIDIA canary-1b, facebook wav2Vec2, and OpenAI's Whisper, were evaluated using Character Error Rate (CER), Word Error Rate (WER), and Semantic Error Rate (SER). A dataset of 48 participants was collected via a Qualtrics survey targeting English speakers with various accents. Statistical analyses, including two-way ANOVA and Tukey HSD tests, were used to examine the effects of ASR model, text type, and participants' first and second best languages on the error metrics. Results showed that text type had a strong, consistent effect on error rates for both best languages, with interaction effects only significant for participants' first best language. This suggests a greater influence of one's strongest language on their speech. No statistically significant differences were found across language groups with the Tukey HSD test, suggesting limited ASR sensitivity to non-native speech. Furthermore, model type, text type, and their interaction significantly impacted error rates, with Whisper consistently achieving the lowest scores across all metrics. Additionally, a modest but significant negative correlation was found between self-perceived accent heaviness and error rates using Pearson's r, indicating that heavier accents are associated with higher transcription errors. These findings underscore the need to consider speaker characteristics and linguistic context in ASR development. Future research could expand data quantity by including paid participants, which may also increase the diversity of accents represented. In addition to this, exploration of other state-of-the-art ASR models can help gain a broader understanding of the diversity of their performance across various speaker characteristics in order to improve inclusivity.

Contents

1 Introduction **1**
- 1.1 Research question . . . 1
- 1.2 Thesis overview . . . 2

2 Background **2**
- 2.1 Accents and important differences in language . . . 2
- 2.2 ASR-models . . . 3
- 2.3 Metrics . . . 5
  - 2.3.1 Character Error Rate . . . 5
  - 2.3.2 Word Error Rate . . . 5
  - 2.3.3 Semantic Error Rate . . . 5

3 Related Work **6**

4 Methodology **6**

5 Results **8**

6 Conclusions and Further Research **12**

References **13**

A Appendix: Survey Questionnaire **16**

B Appendix: Plots **21**

C Appendix: ASR Model Implementations (Jupyter code) **34**
- C.1 NVIDIA/canary-1b implementation . . . 34
- C.2 facebook/wav2vec2-large-960h implementation . . . 37
- C.3 OpenAI's Whisper implementation . . . 40

D Appendix: Metric Calculation and Plotting (Jupyter code) **43**

E Appendix: Data analysis and Plotting (RStudio code) **51**

1 Introduction

The imitation of human skills for listening and understanding speech has been a long-standing goal in technology. The concept of Speech Recognition (SR) itself is older than commonly assumed, with the earliest system developed by Bell Labs dating back to 1952 [O8]. SR can be defined as a system that recognizes patterns in input sound waves, which will then be translated into text. As technology has rapidly improved in the past seventy years, SR has been integrated in our daily lives. Systems like Apple's Siri and Google Translate offer this SR service for free, enabling voice commands, real-time translation, and automated transcription.

Despite these advancements, automatic speech recognition (ASR) technology is not equally accessible or accurate for all users, as speaker characteristics affect the working of ASR programs. SR is easier when the speaker uses the same linguistic variety that the ASR system was trained on [JM21]. A study has found that there are racial disparities in ASR due to differences in the speakers' accents [KNL+20]. However, the extent to which speaker accents impact ASR performance as well as whether certain accents are more affected than others remains uncertain.

This study aims to examine how sensitive the performance of ASR models is to the speech of non-native English speakers with diverse accents. By evaluating multiple ASR systems, we assess their performance differences across speaker groups and investigate the extent of variability in SR accuracy.

1.1 Research question

Recent studies have shown that while ASR has improved significantly, commercial ASR performs best on native English speakers as it is optimized for their experience [Tat17]. Different languages use intonation in distinct ways, which can affect speech recognition. For example, Beckman (1986) classifies English and German as "stress-accent languages", while Japanese is a "non-stress-accent language" [Bec86]. Despite the potential impact of such differences on the accuracy of ASR for non-native speakers, existing research often focuses on accents within a group of native speakers, the accents of non-native speakers whom share their first language, or the general accuracy of ASR itself [ZZH+22].

This study aims to address the question: How sensitive are ASR models in recognizing and processing speech from non-native speakers with diverse accents?

This research is important because it may highlight potential biases in ASR systems. This can help guide improvements for greater accessibility in SR.

Based on prior studies and observations, this study hypothesizes that the Character Error Rate (CER), the Word Error Rate (WER), and the Semantic Error Rate (SER) will be significantly higher for speech with a non-native accent. Thus the error rates will be positively correlated with the self-perceived accent strength of the speaker.

1.2 Thesis overview

This thesis is structured as follows. Chapter 1 provides an introduction to the topic and research problem. In Chapter 2, key definitions and concepts are introduced. Chapter 3 reviews prior research. Chapter 4 describes the experimental setup and methodology. Chapter 5 discusses the results and the data analysis Finally, Chapter 6 presents the conclusions of this research and discusses potential directions for future research.

2 Background

A solid understanding of foundational concepts is essential to contextualize the evaluation of ASR systems.

2.1 Accents and important differences in language

Accent refers the manner of a persons' speech, which contains a significant amount of social information about them. It can reveal to the recipient whether the speaker is a native or non-native speaker of the language. [LZ18]

Moyer (2013) gives us the following definition:[Moy13]

"Accent is a set of dynamic segmental and suprasegmental habits that convey linguistic meaning along with social and situational affiliation." (p. 11)

Second language (L2) learners naturally face the challenge of adapting to unfamiliar sounds and speech patterns, which may conflict with the phonological rules of their first language (L1). Additionally, learners must recognize how subtle variations in intonation, rhythm, and speech rate can convey specific meanings. The level of fluency achieved is often linked to the age at which a learner begins their language acquisition [LM14]. Due to this, having an accent as a L2 learner seems unavoidable.

For instance, while some languages do not treat word stress as a distinct phonological category, English is known for its unpredictable stress patterns. It requires specific rules to describe how stress is assigned in words. Stress refers to the prominence given to certain syllables or words, which may or may not be realized in speech. When this prominence is realized in speech, it contributes to what is perceived as an accent. Stress can be analyzed in two domains: word stress, which refers to syllable prominence within individual words, and sentence stress, which differentiates meanings at the sentence level [GTB07].

Although there exist nativized varieties of English, such as Ghanaian English, Indian English, and Singaporean English, they are often regarded as "second best" to the western varieties of English. This reflects a widespread belief that L2 speakers must conform to the accents of British or American English in order to be considered "valid" [LM14] or "fluent".

2.2 ASR-models

ASR is a machine-operated process of transcribing audio input. This is typically done by analyzing the input using a model or algorithm and generally produces a text output [SJ08]. Both a classification and language model are usually required for this process. Figure 1 below outlines the sequence of steps involved in the automatic transcription of an audio file.

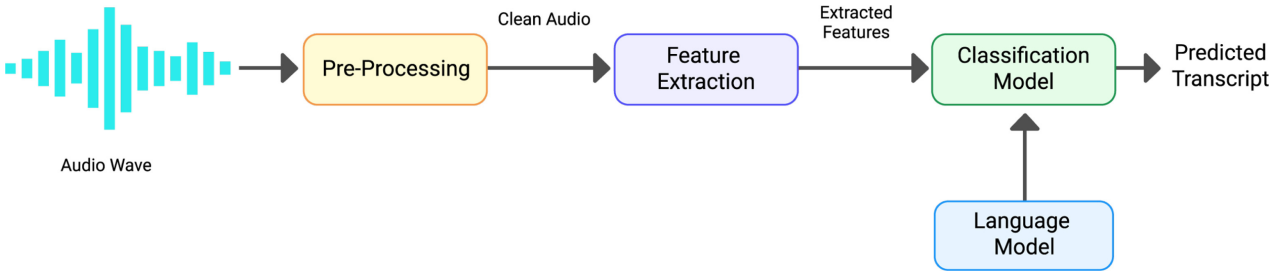

Figure 1: Outline of general ASR model [Pap21]

As technology through the years has progressed, there is a wide variety of ASR models available. Some utilize Hidden Markov Modeling (HMM), while others use a hybrid approach of HMM with Artificial Neural Networks (ANN). Researching and testing these methods in the 1980s led to significant work on building systems for large-vocabulary continuous speech recognition [LS12]. Table 1 provides an overview of the models with their characteristics listed. These include key factors such as model type, meaning cloud-based or local, and if the model uses real-time processing or batch processing.

ASR Models					
Model Name	**Type of Model**	**API Availability**	**Ease of Use**	**Real-time vs Batch Processing**	**Pricing**
Amazon Transcribe	Cloud	Yes	Moderate	Real-time	Limited free tier of 60 min, then $0.0004 per sec
AssemblyAI	Cloud	Yes	Simple	Real-time	Limited free tier of 5 min per month, then $0.00025 per sec
Deepgram Speech-to-Text API	Cloud	Yes	Simple	Real-time	Free 200 min, then $0.0003 per sec
facebook/ wav2vec2-large-960h (Hugging Face)	Local	No	Moderate	Batch	Free
Google Cloud Speech-to-Text AI	Cloud	Yes	Moderate	Real-time	Limited free tier of 1 hour per month, then $0.006 per 15 sec
Microsoft Azure AI Speech	Cloud	Yes	Moderate	Real-time	Limited free tier of 5 hours per month, then $1 per hour
Mozilla DeepSpeech	Local	No	Moderate	Batch	Free
NVIDIA/Canary-1b (Hugging Face)	Local	No	Moderate	Batch	Free
OpenAI Whisper	Both	Yes	Simple	Both	$0.006 per minute
Rev AI	Cloud	Yes	Simple	Real-time	$0.005 per minute
Vosk API	Local	Yes (Limited)	Simple	Real-time	Free

Table 1: Overview of the state of the art ASR models

2.3 Metrics

Specific metrics are required to evaluate the sensitivity of the ASR systems' performance.

2.3.1 Character Error Rate

The CER is a metric based on how many individual characters have been incorrectly transcribed in comparison to the reference text. This metric is calculated using the Levenshtein distance, which is given by the minimum amount of alterations needed to change one string into the other. The list of possible operations consists of an insertion, deletion, or substitution of a single character [MVM09]. The computation of a CER is achieved using the base formula:

$$CharacterErrorRate = (S + D + I)/N \text{ [MMD+]}$$

with the following variables:
S = Number of character substitutions
D = Number of character deletations
I = Number of character insertions
N = Number of characters in the reference text

2.3.2 Word Error Rate

The WER is a metric that, similarly to the CER, is calculated using the Levenshtein distance. However, for WER this distance is applied on each of the words of the predicted and the reference text instead of the individual characters. This is achieved using a slightly modified base formula:

$$WordErrorRate = (S + D + I)/N \text{ [MMD+]}$$

with the following variables:
S = Number of word substitutions
D = Number of word deletations
I = Number of word insertions
N = Number of words in the reference text

2.3.3 Semantic Error Rate

The SER measures how well a system captures the intended meaning of a sentence by comparing the semantic elements between the prediction and the reference text. Unlike CER and WER, the SER does not have a standardized formula to calculate the metric within Python libraries. As such, the metric can be determined by comparing the semantic similarity of the sentence embeddings, otherwise known as vectors. Then the cosine similarity, which is measured as the angle between two embeddings with an higher similarity implying the patterns of both embeddings are more similar [XZL15], will be calculated in order to decide the semantic similarity. Afterwards, the SER can be computed by subtracting the semantic similarity from 1.

3 Related Work

A study by Tatman evaluated ASR performance in the automatically generated captions of the video-platform Youtube across two genders and five English dialects. Speech samples from the "accent tag challenge" were used, where speakers explicitly identify their linguistic background. The study provided insights into ASR biases by controlling for both dialect and gender, as it found both a gender and dialect bias. There were differences found in accuracy, with women and speakers from Scotland both scoring lower. The study concluded the high WER scores stem from unequal training data and biases in ASR models, particularly against non-standard voice pitches and the unique speech patterns of different dialects [Tat17].

Maxwell-Smith and Foley also investigated ASR using segments from Youtube. However, they used Indonesian language lessons that prioritize transfer learning with English as the language of instruction from three different channels as their data. Furthermore, separate ASR models were used for this research. Their findings highlighted how differences in pronunciation reflected the transfer of vowel production in the Indonesian language and in some cases speech errors resulting from irregularities in English spelling. The study concluded that publicly available ASR models for Indonesians are not well-suited for processing language teaching data, thus emphasizing the need for models that are trained on possible linguistic transfer and pronunciation variations [MSF23].

A different study by Souza and Gottardi focused on how well freely available ASR-based dictation tools for language learners can understand foreign-accented speech. Additionally, the study discusses if these tools can help language learners with their pronunciation. English speech examples of Brazilian Portuguese and Spanish speakers from an online database were used, with the requirement that the length of residence of these speakers must be under 1.5 years. Microsoft Word and VoiceNotebook were chosen to be utilized as the dictation tools. Their results indicated that both programs understood the speakers speech well, but still made quite some mistakes based on the pronunciation of the speakers and the way the tools were correcting these words. The words that were most affected were connected speech and both consonant and vowel shifts [SG22].

4 Methodology

This study follows a cross-sectional design using between-participant comparisons, meaning the data is collected from participants in a short period of time with no follow-up, and comparisons were made between individuals rather than predefined groups. The primary goal of the study is to compare the transcription accuracy between English speakers based on their accent. Their audio data will be processed using numerous ASR models to determine differences in transcription accuracy. As such, this study evaluates the performance of various ASR models, which have been selected based on their widespread use and accessibility.

The dataset for this study will be constructed by collecting audio recordings from English speakers with different accents via a survey on Qualtrics, a platform for creating and analyzing online surveys. This will serve as the input for evaluating the performance of ASR models across these different groups [Qua25].

A participant criterion will be used in order to ensure the evaluation is based on the accent

rather than other factors. Specifically, participants must not have speech impairments. Above this, participants will fill in a form containing questions about their language and general background in order to label the data. Participants were required to submit a short audio recording as the last part of the survey. Feedback and participant behavior recorded by Qualtrics suggested that this requirement led to non-completion for some respondents.

Numerous questions regarding the language background of the participants were asked, in order to determine as closely as possible the type of accent they might have, as well as question about how they perceive their own accent. Other questions were used to gather general information regarding the participants' age and gender. The passages used as references for the participants to read and record can be divided into different types of audio; the first passage is a narrative text type, originating from the book "Alice in Wonderland", and the second consists of a list of commands. The first passage was chosen as it has no copyright restrictions and covers intonation and rhythm, while the second passage was created to resemble natural speech.

A total of 50 responses were collected, with 2 being found unusable due to incorrect file uploading. In both cases image files had been uploaded instead of audio files. This is likely due to confusion or misunderstanding of the requested tasks. These entries have been deleted from the collected dataset. The survey has been created in a way to specifically ensure incomplete entries, meaning answers missing from one or more questions, are not saved. In spite of that, Qualtrics shows that another 15 people started the survey without finishing it. However, it should be noted that these may not be different individuals, but rather the same people completing the survey at a different time, resulting in multiple survey attempts.

The selected ASR models include: NVIDIA Canary-1b (canary), facebook wav2vec large 960h (facebook wav2vec), and OpenAI Whisper (Whisper). The first two models are open-source and available on the HuggingFace platform [Hug25]. The latter model, OpenAI's Whisper, is developed by OpenAI and can be accessed through an application programming interface (API) for local deployment [Ope22]. These models have been implemented in separate Python files, which can be seen in Appendix C, in order to efficiently transcribe the audio files. A formatting issue with the canary model led to the multiplication of the audio files. This was resolved by processing each file individually rather than using a loop.

To assess ASR performance, CER, SER, and WER have been used. To do so, the datasets.load_metric("wer") from Hugging Face computes the WER automatically using its base formula word-tokenized. The CER is calculated using the `jiwer` library with the same base formula but character-tokenized [Sme20]. For the computation of SER both the reference and the transcription texts have to firstly be transformed into embeddings. This is done utilizing the `SentenceTransformer` library. Then the cosine similarity between both embeddings will be calculated using the `sentence_transformers.util` package [RG20]. Afterwards, the SER can be computed by subtracting the semantic similarity from 1. This can be seen in Appendix D.

As the hypothesis proposes that the performance of the ASR models would depend on the perceived accent of the participants, a two-way analysis of variance (ANOVA) will be conducted for each metric in RStudio [Pos23], which can be seen in Appendix E. This has been performed on model type, text type, and their interaction and on first/second best language of the participant and its

interaction with text type. If there is a significantly low p-value of the performance in the variables individually or in their interaction with one another, the Tukey's Honestly Significant Difference (HSD) will be performed as a post-hoc test to determine which groups differ significantly from each other. In addition to this, a correlation using Pearson's r is utilized to analyse the self-perceived accent heaviness of the participants on the error metrics achieved. Accent heaviness was rated using a 5-point Likert scale, which first had to be numerized to the following:

- 1 = "People usually do not understand me"
- 2 = "People often do not understand me"
- 3 = "People understand me with some effort"
- 4 = "Most people understand me relatively easily"
- 5 = "My accent does not hinder understanding me"

5 Results

The survey delivered a database consisting of 48 participants, after cleaning the data. It is noteworthy to mention that a significant number of participants withdrew after reaching the audio recording section, which suggests that this requirement may have affected the overall response rate and consequently the results of this research.

After calculating the different metrics on transcriptions, divided based upon model and text type, outliers can be observed across scatter plots 2, 3, 6, 7, 10 and 11 in Appendix B. While outliers are expected in the data, a few audio files particularly stand out depending on the text type rather than the model. For text type 1, this would be Response ID `R_9nWoBgpTutS4wTv`, a woman between the ages 55 to 64 with a self-noted Indonesian accent. For text type 2, this would be Response ID `R_9KIqHuN968p2vYY`, a woman between ages 18 to 24 with a self-noted American accent. Both have noted to be residents of Indonesia.

Two outliers, with Response ID's `R_2SYrXQ0D0LH5HOx` and `R_6O2ZobEcF2rpnuD`, have been found that performed significantly bad, both being men between 18 to 24 years old with no clear self-noted description of their accent.

To determine whether text type and participants' first and second best languages have an effect on error metrics, the data was regrouped and then an ANOVA was conducted. After regrouping the languages that were submitted, a total of 9 different languages for first best languages and 6 for second best languages were chosen, visualized in the bar charts 14 and 15 in Appendix B. The results of the ANOVA in tables 2 and 3 show that text type has a consistent and strong effect on both CER and WER across participants' first and second best languages. Interactions between text type and participants' best languages were found to be significant for CER with their first best language, but not with their second best language. This suggests that participants' best language plays a greater role in shaping their speech patterns, therefore influencing how they handle different text types.

Two-way ANOVA Results – First Best Language						
Metric	**Effect of**	**Df**	**Sum Sq**	**Mean Sq**	**F-value**	**p-value**
CER	text_type	1	0.337427243	0.337427243	48.41654139	2.61E-11
CER	Q4_clean	8	0.165335008	0.020666876	2.96543529	0.00338
CER	Interaction	8	0.140310950	0.017538869	2.51660582	0.01182
CER	Residuals	270	1.881698961	0.006969255		
WER	text_type	1	0.131658320	0.131658320	8.04991707	0.00490
WER	Q4_clean	8	0.225727629	0.028215954	1.72519357	0.09255
WER	Interaction	8	0.159209128	0.019901141	1.21680525	0.28905
WER	Residuals	270	4.415914596	0.016355239		
SER	text_type	1	0.046242869	0.046242869	4.69130011	0.03119
SER	Q4_clean	8	0.085569201	0.010696150	1.08511543	0.37369
SER	Interaction	8	0.022100583	0.002762573	0.28026070	0.97208
SER	Residuals	270	2.661431644	0.009857154		

Table 2: Results of the two-way ANOVA examining the effects of text type, first best language, and their interaction on each error metric

Two-way ANOVA Results – Second Best Language						
Metric	**Effect of**	**Df**	**Sum Sq**	**Mean Sq**	**F-value**	**p-value**
CER	text_type	1	0.337427243	0.337427243	46.12857259	6.79E-11
CER	Q5_clean	5	0.088102962	0.017620592	2.40885344	0.03686
CER	Interaction	5	0.080321495	0.016064299	2.19609766	0.05494
CER	Residuals	276	2.018920462	0.007314929		
WER	text_type	1	0.131658320	0.131658320	8.00795567	0.00500
WER	Q5_clean	5	0.153343643	0.030668729	1.86538777	0.10051
WER	Interaction	5	0.109808240	0.021961648	1.33579028	0.24922
WER	Residuals	276	4.537699469	0.016440940		
SER	text_type	1	0.046242869	0.046242869	5.10978627	0.02457
SER	Q5_clean	5	0.137922437	0.027584487	3.04805560	0.01073
SER	Interaction	5	0.133416640	0.026683328	2.94847848	0.01304
SER	Residuals	276	2.497762352	0.009049864		

Table 3: Results of the two-way ANOVA examining the effects of text type, second best language, and their interaction on each error metric

Across both the first and second best languages, the Tukey post-hoc tests for CER, WER, and SER revealed no statistically significant differences between any language pairs, as all the adjusted p-values are found to be above 0.05. While some comparisons showed relatively larger mean differences, such as Mandarin vs. Indonesian, they remained non-significant. This suggests that error rates were generally consistent across languages for all metrics. Assuming that participants who listed a language other than English as their first best language are non-native English speakers, these findings contradict the initial hypothesis.

In addition to this, in Appendix B, the scatter plots 16-21 show the CER, WER and SER metrics divided by the best languages of the participants, with a color scheme based on the type of text and model used. Furthermore, scatter plots and histograms 8-19 show that the general performance of text type 1 receives higher error rates than that of text type 2. This could be explained by the difference in language use, as text 1 originated from 1865 [Car65], while text 2 is a text created specifically for this research in 2025 and thus in a modern context.

Presented in table 4 are the results yielded by a two-way ANOVA examining the effects of model, text type, and their interaction on error rates. As can be seen by the consistently low p-values with corresponding high F-values, the table reveals significant effects across all three factors. These findings indicate that both the individual models and text types, as well as their interaction, have a great effect on the error rates. The effects are visualized in Bar Charts 22-24 in Appendix B, which include confidence intervals.

A low WER combined with a high CER for the same model may indicate that multiple character-level errors occur within the same words. This is particularly evident when the SER is also high, suggesting that entire words are being misrecognized and replaced by different, incorrect words. In contrast, a high WER accompanied by a low CER may imply wide-spreading of the character errors. These patterns suggest that each model exhibits different methods in transcribing, or rather a distinct specialization.

As a result of the significant effects found with the two-way ANOVA, a post-hoc test was conducted to determine which specific group differences were statistically significant. For this, Tukey HSD has been utilized in Rstudio. The results in table 5 show that all pairwise model comparisons show statistically significant p-values, except for the comparison between the models Whisper and canary on SER. This implies that for SER, both models perform similarly. Overall, the Whisper model receives the lowest error rates for this test, followed by facebook wav2vec, and then canary.

Two-way ANOVA Results						
Metric	**Effect of**	**Df**	**Sum Sq**	**Mean Sq**	**F-value**	**p-value**
CER	model	2	0.257626274	0.128813137	22.63020057	7.68E-10
CER	text_type	1	0.337427243	0.337427243	59.28002668	2.32E-13
CER	Interaction	2	0.324549284	0.162274642	28.50879796	5.29E-12
CER	Residuals	282	1.605169361	0.005692090		
WER	model	2	1.974441848	0.987220924	116.96675486	1.02E-37
WER	text_type	1	0.131658320	0.131658320	15.59898704	9.90E-05
WER	Interaction	2	0.446277639	0.223138819	26.43767263	2.99E-11
WER	Residuals	282	2.380131866	0.008440184		
SER	model	2	1.209771900	0.604885950	116.68116911	1.19E-37
SER	text_type	1	0.046242869	0.046242869	8.92014766	0.00307
SER	Interaction	2	0.097415501	0.048707750	9.39561788	0.000112
SER	Residuals	282	1.461914028	0.005184092		

Table 4: Two-way ANOVA results for each error metric based on model, text type, and their interaction

Tukey HSD Results					
Metric	**Comparison**	**Diff**	**Lower Bound**	**Upper Bound**	**p-value**
CER	fb - canary	-0.03190	-0.06223	-0.00157	0.03666
CER	Whisper - canary	-0.07307	-0.10340	-0.04274	1.02×10^{-7}
CER	Whisper - fb	-0.04117	-0.07150	-0.01084	0.00438
WER	fb - canary	0.06483	0.03018	0.09947	4.38×10^{-5}
WER	Whisper - canary	-0.13402	-0.16866	-0.09937	5.13×10^{-13}
WER	Whisper - fb	-0.19884	-0.23349	-0.16420	4.84×10^{-13}
SER	fb - canary	0.13246	0.10694	0.15799	4.84×10^{-13}
SER	Whisper - canary	-0.00955	-0.03508	0.01597	0.65224
SER	Whisper - fb	-0.14201	-0.16754	-0.11649	4.84×10^{-13}

Table 5: Tukey HSD post-hoc test results for pairwise comparisons between models for each error metric

In table 6 the results of the correlation between self-perceived accent heaviness and the error metrics can be found. Notably, the two lowest Likert scale categories were not selected by any participants, and thus were excluded from the analysis.

The results revealed a statistically significant negative correlation for all three metrics, indicating that stronger perceived accent heaviness is associated with higher error rates. This correlation is visualized in the linear regression plots 25-27 in Appendix B. Despite the relatively small effect size, with Pearson's r ranging from -0.12 to -0.14, the results still indicate a consistent, but modest, association between these factors. Although the correlation is statistically negative due to the coding of the Likert scale, with higher values representing lighter accents, this finding aligns with the hypothesis: heavier perceived accents are associated with higher error rates.

Pearson Correlation Results					
Metric	**Pearson's r**	**95% Confidence Interval**	**t-value**	**df**	**p-value**
CER	-0.1197	[-0.2321, -0.0042]	-2.039	286	0.0424
WER	-0.1216	[-0.2339, -0.0061]	-2.071	286	0.0392
SER	-0.1399	[-0.2514, -0.0247]	-2.390	286	0.0175

Table 6: Pearson's r correlation between self-perceived accent heaviness and each error metric

6 Conclusions and Further Research

The study's results demonstrate that text type had a strong and consistent effect on error rates (CER, WER, SER) across both participants' first and second best languages. Significant interaction effects were only observed for participants' first best language, suggesting that a speaker's strongest language has a greater influence on how they handle different text types in speech. However, it did not find that ASR models performed particularly sensitive to non-native speakers' speech, as the Tukey HSD test yielded no statistically significant results, thus rejecting the first part of the proposed hypothesis. Furthermore, the choice of ASR model significantly impacted transcription accuracy. The interaction effects between this factor and text type further influenced the error rates, highlighting the complexity of speech recognition performance across different conditions. Additionally, the self-perceived heaviness of a speaker's accent is negatively correlated with the transcription quality. Although the effects were modest, the results indicate that the heavier a participant perceived their accent to be, the higher their error rates and thus the less sensitive the models would perform on their speech. This supports the second part of the hypothesis. Notably, Whisper almost consistently outperformed the other models, followed by facebook wav2vec2, and lastly canary.

These findings highlight the importance of considering speaker characteristics and text context when evaluating and deploying ASR systems. Possible directions for future work may include paid participants in order to increase data quantity. This issue needs to be addressed in future research designs, especially given the high dropout rate at the audio recording stage. Furthermore, expanding the participant group to include a more diverse range of accents would allow for a deeper

investigation into the sensitivity of these models across different speech patterns and linguistic backgrounds.

A different direction would be to evaluate other models, such as Google Cloud Speech-to-Text and AssemblyAI. By exploring other state-of-the-art models, we can gain a broader understanding of how diverse ASR systems perform across various speaker characteristics and text contexts, potentially identifying strengths and weaknesses unique to each platform.

References

[Bec86] Mary E. Beckman. *Stress and Non-Stress Accent*. De Gruyter Mouton, Berlin, Boston, 1986.

[Car65] Lewis Carroll. *Alice's Adventures in Wonderland*. Macmillan Publishers, London, 1865. Original publication.

[GTB07] Ulrike Gut, Jürgen Trouvain, and William J. Barry. Bridging research on phonetic descriptions with knowledge from teaching practice – the case of prosody in non-native speech. In Jürgen Trouvain and Ulrike Gut, editors, *Non-Native Prosody: Phonetic Description and Teaching Practice*, Trends in Linguistics. Studies and Monographs; 186, pages 3–21. Mouton de Gruyter, Berlin · New York, 2007.

[Hug25] Hugging Face. Hugging face. https://huggingface.co/, 2025.

[JM21] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Pearson, 3rd edition, 2021.

[KNL+20] Allison Koenecke, Andrew Nam, Emily Lake, Joe Nudell, Minnie Quartey, Zion Mengesha, Connor Toups, John R. Rickford, Dan Jurafsky, and Sharad Goel. Racial disparities in automated speech recognition. *Proceedings of the National Academy of Sciences*, 117(14):7684–7689, 2020.

[LM14] John M. Levis and Alene Moyer, editors. *Social Dynamics in Second Language Accent*. De Gruyter Mouton, Berlin, Boston, 2014.

[LS12] John M. Levis and Ruslan Suvorov. Automatic speech recognition. In Carol A. Chapelle, editor, *Encyclopedia of Applied Linguistics*, pages 316–323. Wiley-Blackwell, 2012.

[LZ18] John Levis and Ziwei Zhou. *Accent*. John Wiley Sons, Inc, 01 2018.

[MMD+] Iain A McCowan, Darren Moore, John Dines, Daniel Gatica-Perez, Mike Flynn, Pierre Wellner, and Hervé Bourlard. On the use of information retrieval measures for speech recognition evaluation.

[Moy13] Alene Moyer. *Foreign Accent: The Phenomenon of Non-native Speech*. Cambridge University Press, Cambridge, UK, 2013.

[MSF23] Zara Maxwell-Smith and Ben Foley. Automated speech recognition of Indonesian-English language lessons on YouTube using transfer learning. In Oleg Serikov, Ekaterina Voloshina, Anna Postnikova, Elena Klyachko, Ekaterina Vylomova, Tatiana Shavrina, Eric Le Ferrand, Valentin Malykh, Francis Tyers, Timofey Arkhangelskiy, and Vladislav Mikhailov, editors, *Proceedings of the Second Workshop on NLP Applications to Field Linguistics*, pages 1–16, Dubrovnik, Croatia, may 2023. Association for Computational Linguistics.

[MVM09] Frederic P. Miller, Agnes F. Vandome, and John McBrewster. *Levenshtein Distance: Information theory, Computer science, String (computer science), String metric, Dam-*

erau?Levenshtein distance, Spell checker, Hamming distance. Alpha Press, 2009.

[Ope22] OpenAI. Whisper: Open-source automatic speech recognition. https://openai.com/index/whisper/, 2022. Released September 21, 2022.

[O8] Douglas O'Shaughnessy. Automatic speech recognition: History, methods and challenges. *Pattern Recognition*, 41(10):2965–2979, 2008.

[Pap21] Ilias Papastratis. Speech recognition: a review of the different deep learning approaches. *Accessed on*, 2:2021, 2021.

[Pos23] Posit Software, PBC. *RStudio: Integrated Development Environment for R*. Posit Software, PBC, Boston, MA, 2023.

[Qua25] Qualtrics. Qualtrics survey platform. https://leidenuniv.eu.qualtrics.com/jfe/form/SV_0IIXp0eTzkt0h2C, 2025.

[RG20] Nils Reimers and Iryna Gurevych. Sentence-transformers: Multilingual sentence embeddings using bert and beyond. https://github.com/UKPLab/sentence-transformers, 2020.

[SG22] Hanna Kivistö de Souza and William Gottardi. How well can asr technology understand foreign-accented speech? *Trabalhos em lingüística aplicada*, 61(3):764–781, 2022.

[SJ08] Andrew. Sears and Julie A. Jacko. *The human-computer interaction handbook : fundamentals, evolving technologies, and emerging applications*. Human factors and ergonomics. Lawrence Erlbaum Associates, New York, NY [etc, 2nd ed. edition, 2008.

[Sme20] Jitsi Smeets. jiwer: Speech recognition evaluation tool. https://github.com/jitsi/jiwer, 2020.

[Tat17] Rachael Tatman. Gender and dialect bias in youtube's automatic captions. In *Proceedings of the First ACL Workshop on Ethics in NLP*, pages 53–59, Valencia, Spain, April 2017. Association for Computational Linguistics.

[XZL15] Peipei Xia, Li Zhang, and Fanzhang Li. Learning similarity with cosine similarity ensemble. *Information sciences*, 307:39–52, 2015.

[ZZH+22] Yuanyuan Zhang, Yixuan Zhang, Bence Mark Halpern, Tanvina Patel, and Odette Scharenborg. Mitigating bias against non-native accents. In *Proceedings of Interspeech 2022*, Incheon, Korea, September 2022.

A Appendix: Survey Questionnaire

This survey will gather information about your language background to help us analyze speech and accents. Your responses will remain confidential and will only be used for research purposes. We appreciate your time and contribution!

If you have any questions about the survey or research, or if you wish to withdraw your information, please contact t.k.nanhekhan@umail.leidenuniv.nl.

This study is conducted as part of a Bachelor's thesis at Leiden University, Faculty of Science.

[Q1] By continuing with this survey, you confirm that you are voluntarily participating, that you understand how your data will be used, and that you agree to the anonymity and confidentiality of your responses.

- I agree and wish to participate
- I do not agree and wish to exit the survey

Q[2] What is your age?

- Under 18
- 18-24
- 25-34
- ...
- 75-84
- 85 or older

Q[3] What is your gender?

- Male
- Female
- Non-binary / third gender
- Prefer not to say

Q[4] What is your first best/most fluent language?

Q[5] What is your second best/most fluent language?

Q[6] What is your third best/most fluent language?

Q[7] In which country do you currently reside?

- Afghanistan
- Albania
- Algeria
- ...
- Zimbabwe

Q[8] Do you have any speech impairments or conditions affecting pronunciation?

- Yes
- No

Q[9] At what age did you start learning English?

- Under 18
- 18-24
- 25-34
- ...

- 75-84
- 85 or older

Q[10] Where did you (mainly) learn English?
(Participants were permitted to select multiple options.)

- At home through family
- At school (as part of formal education)
- Through media (movies, music, books, etc.)
- Through (voluntary) language courses or tutoring
- Through work or professional environment
- *Other

Q[11] How often do you speak English in daily life?

Once a year	Once every 3 months	Once every month	Once every week	Daily
O	O	O	O	O

Q[12] How would you describe your English accent without referencing its quality? For example: South British or nasal

Q[13] how heavy is your accent when speaking English (1-5 labelled)?

People usually do not understand me	People often do not understand me	People understand me with some effort	Most people understand me relatively easily	My accent does not hinder understanding me
O	O	O	O	O

Q[14] How comfortable do you feel speaking English?

Extremely uncomfortable	Somewhat uncomfortable	Neither comfortable nor uncomfortable	Somewhat comfortable	Extremely comfortable
O	O	O	O	O

Q[15] Do you use voice assistants while speaking English? (Example: Siri and Google Voice Search)

Once a year or never	Once every 3 months	Once every month	Once every week	Daily
O	O	O	O	O

Q[16] How well did/do voice assistants understand you?

Not well at all	Slightly well	Moderately well	Very well	Extremely well	Not Applicable
O	O	O	O	O	O

Q[17] For the following part, we ask you to read and record a passage from the book "Alice in Wonderland" by Lewis Carroll.

What you need to do: 1. Read the passage aloud and at a comfortable pace, in a way you would naturally speak. Make sure that your recording is clear and your voice is audible. Try to minimize mistakes, small errors are acceptable but try not to repeat sentences. 2. Save the recording in a common audio format (e.g. MP3 or WAV). This can be done using the "Voicerecorder"-app via Windows or via a voice memo app on Macbook and mobile. 3. Upload your recording through the provided upload location below in this survey.

If you need help, please refer to the tutorials below on how to record and upload:

- on Windows: https://youtu.be/sMcTWII4oiY
- on mobile: https://youtube.com/shorts/A1JfMHXu75s?feature=share

The passage is as follows: Alice waited a little, half expecting to see it again, but it did not appear, and after a minute or two she walked on in the direction in which the March Hare was said to live. "I've seen hatters before," she said to herself; "the March Hare will be much the most interesting, and perhaps as this is May it won't be raving mad—at least not so mad as it was in March." As she said this, she looked up, and there was the Cat again, sitting on a branch of a tree. "Did you say pig, or fig?" said the Cat. "I said pig," replied Alice; "and I wish you wouldn't keep appearing and vanishing so suddenly: you make one quite giddy." "All right," said the Cat; and this time it vanished quite slowly, beginning with the end of the tail, and ending with the grin, which remained some time after the rest of it had gone.

Choose file

No file chosen

Q[18] For the next section, we ask you to read and record the following instructions/commands, which have been combined into a single text.

What you need to do: 1. Read the passage aloud and at a comfortable pace, in a way you would naturally speak. Make sure that your recording is clear and your voice is audible. Try to minimize mistakes, small errors are acceptable but try not to repeat sentences. 2. Save the recording in a common audio format (e.g. MP3 or WAV). This can be done using the "Voicerecorder"-app via Windows or via a voice memo app on Macbook and mobile. 3. Upload your recording through the provided upload location below in this survey.

If you need help, please refer to the tutorials below on how to record and upload:

- on Windows: https://youtu.be/sMcTWII4oiY
- on mobile: https://youtube.com/shorts/A1JfMHXu75s?feature=share

Read out loud the following instructions/commands as if you are instructing a voice assistant:

- Set an alarm for seven o'clock.
- Explain this part using the following mathematical equation.
- What is the solution to this question?
- Find nearby Italian restaurants that provide gluten-free and vegetarian options.
- What is the name of the song that contains the following text: Look at me now, will I ever learn? I don't know how, but I suddenly lose control.
- Send a text message to Tim saying, I will be there in twenty minutes.
- Set a reminder for a hairsalon appointment on April fiftheenth at one O'clock.
- Find the best cafe near the Eiffel Tower and give the directions to it.

Choose file

No file chosen

We thank you for your time spent taking this survey.

Your response has been recorded.

B Appendix: Plots

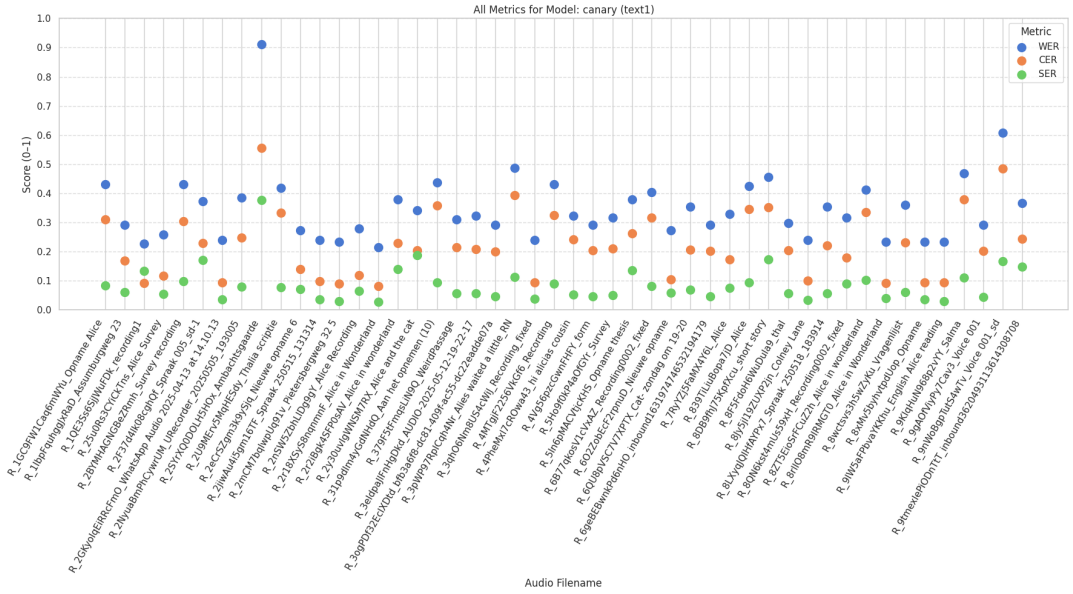

Figure 2: Scatterplot of all error rates for text type 1 (Q17) on NVIDIA/canary-1b model

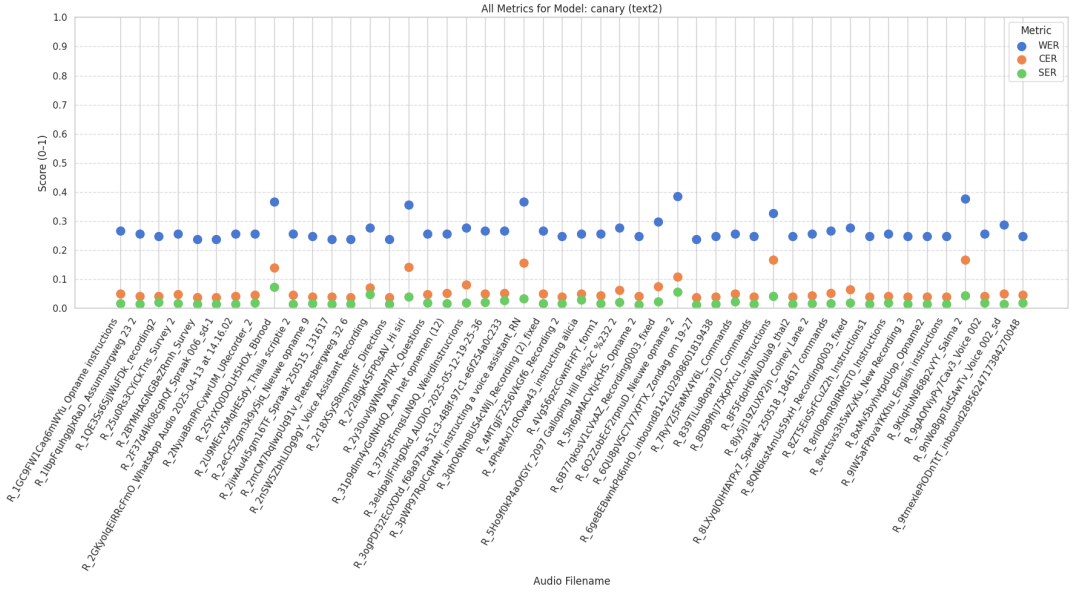

Figure 3: Scatterplot of all error rates for text type 2 (Q18) on nvidia/canary-1b model

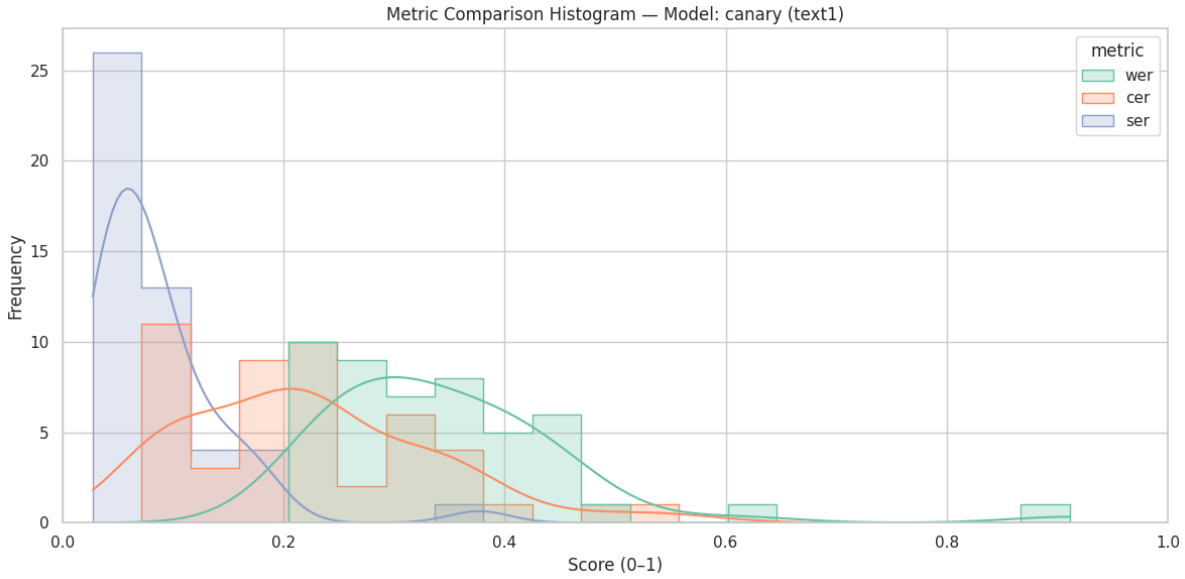

Figure 4: Histogram of all error rates for text type 1 (Q17) on NVIDIA/canary-1b model

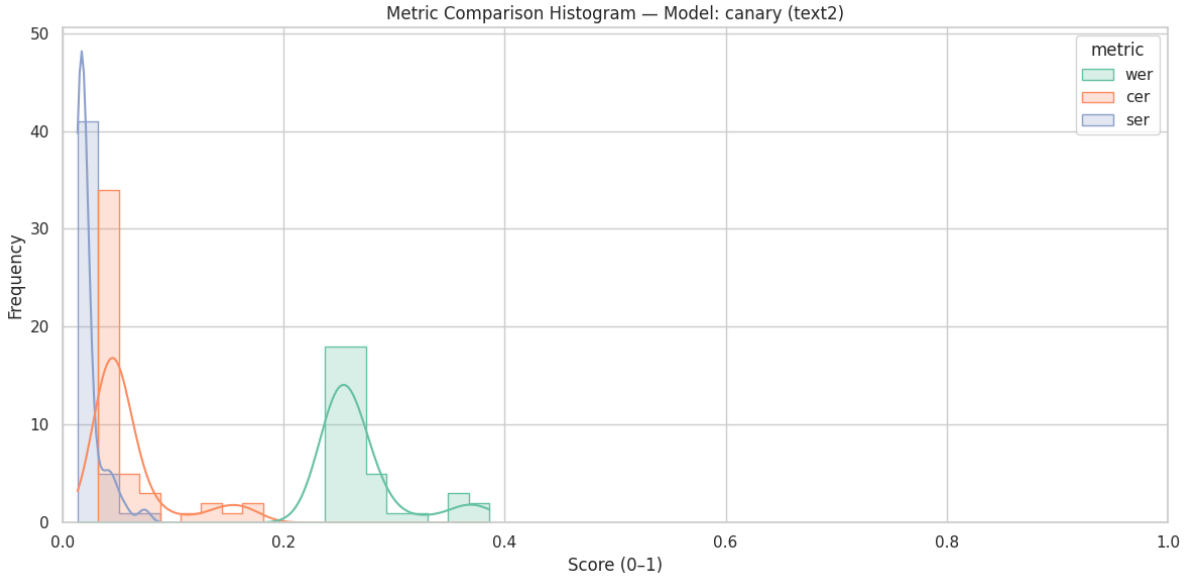

Figure 5: Histogram of all error rates for text type 2 (Q18) on NVIDIA/canary-1b model

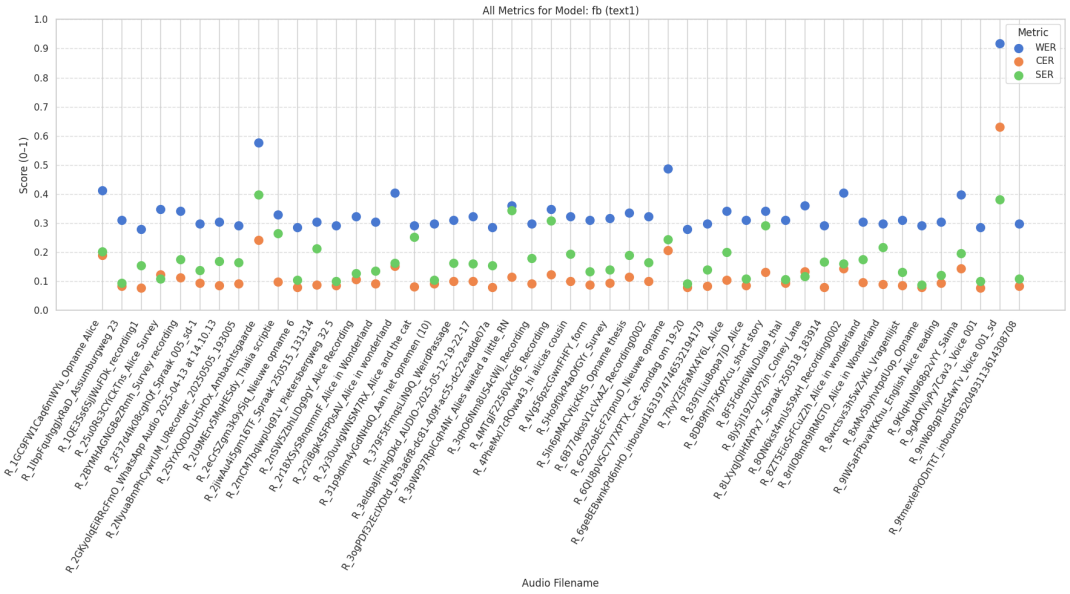

Figure 6: Scatterplot of all error rates for text type 1 (Q17) on facebook/wav2vec2-large-960h model

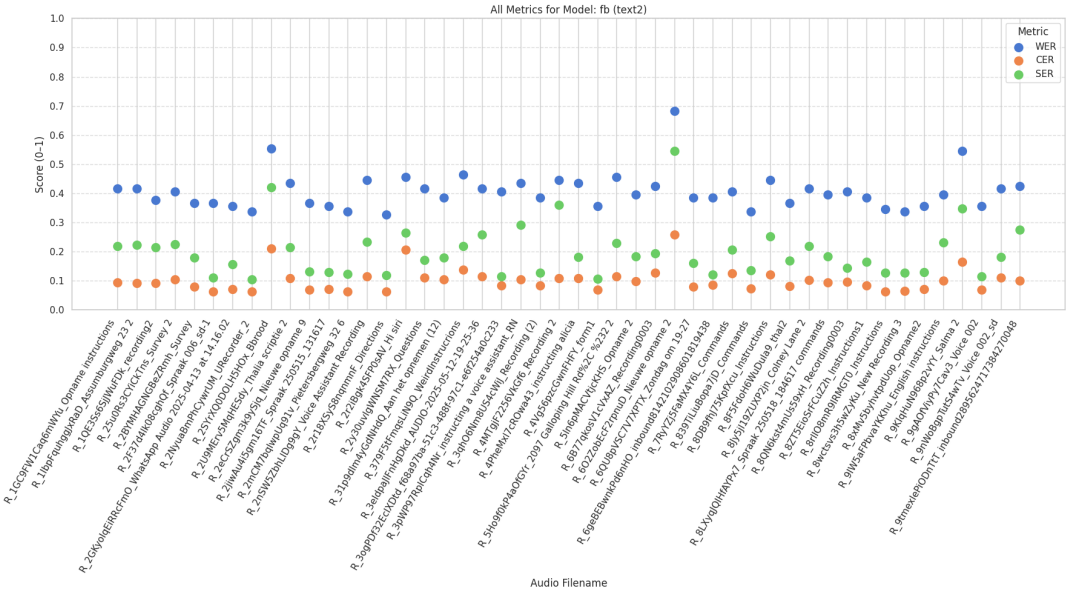

Figure 7: Scatterplot of all error rates for text type 2 (Q18) on facebook/wav2vec2-large-960h model

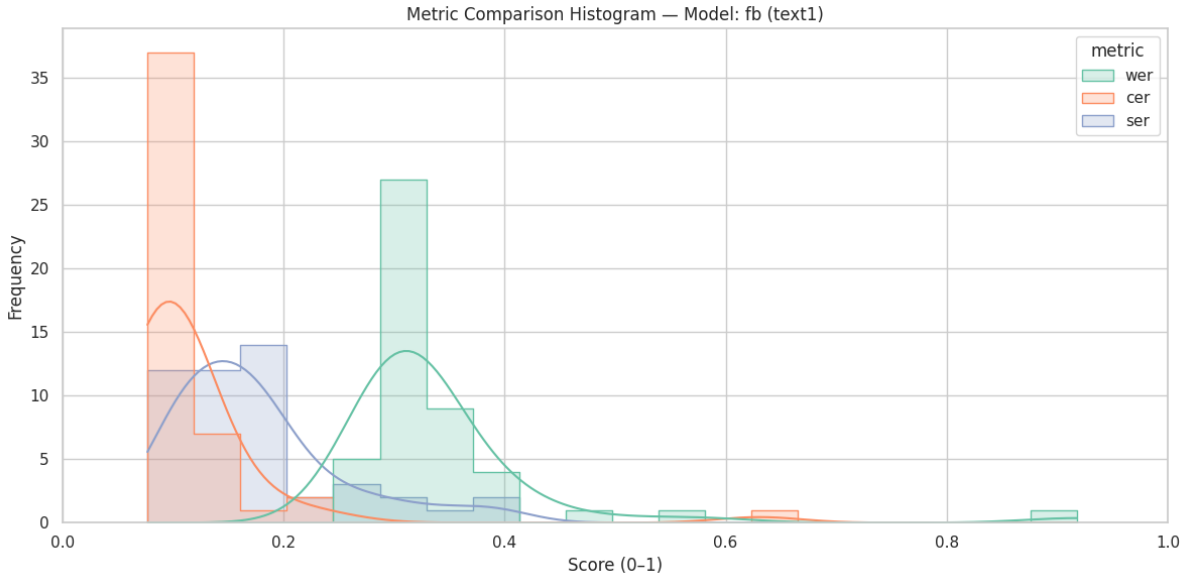

Figure 8: Histogram of all error rates for text type 1 (Q17) on facebook/wav2vec2-large-960h model

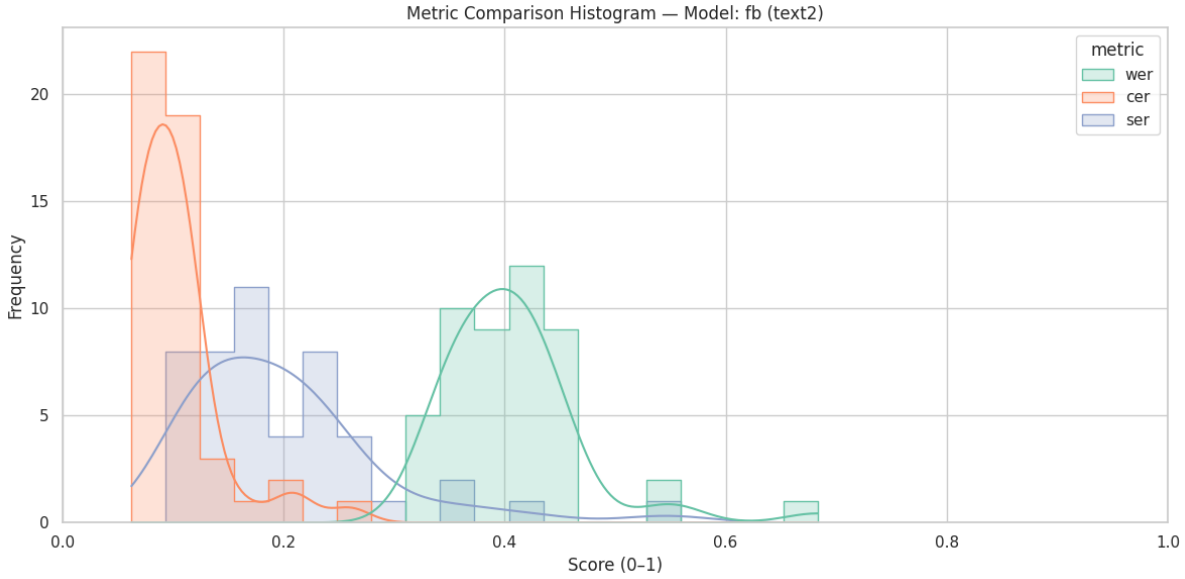

Figure 9: Histogram of all error rates for text type 2 (Q18) on facebook/wav2vec2-large-960h model

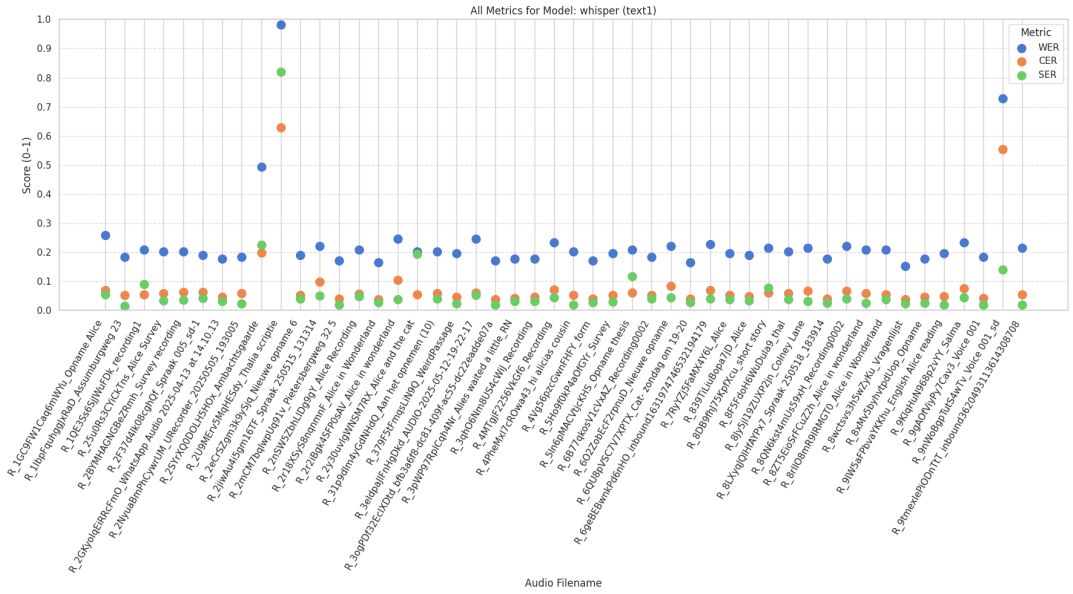

Figure 10: Scatterplot of all error rates for text type 1 (Q17) on OpenAI's Whisper model

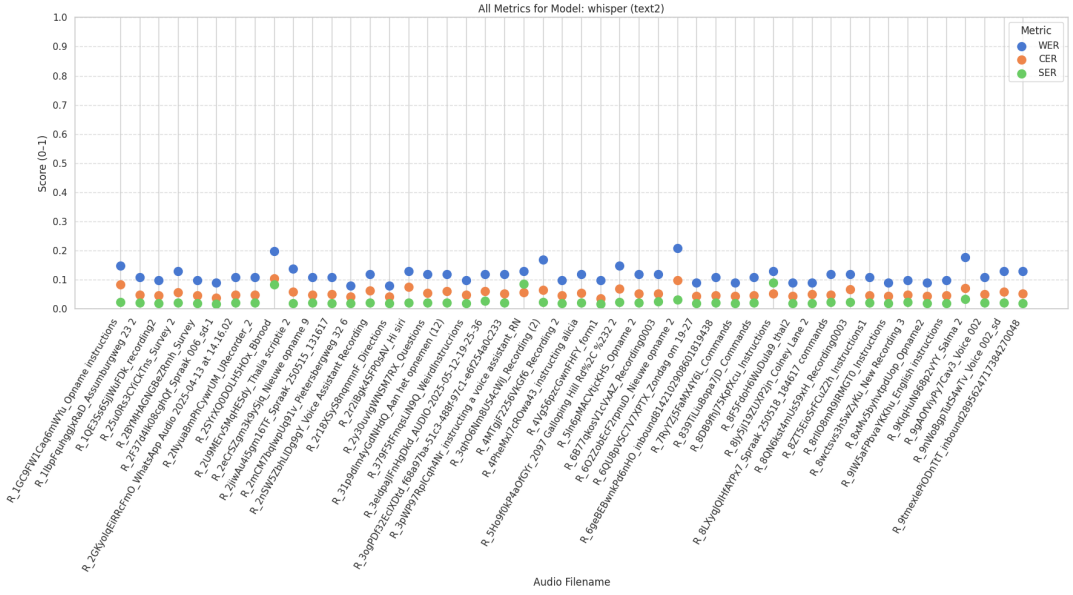

Figure 11: Scatterplot of all error rates for text type 2 (Q18) on OpenAI's Whisper model

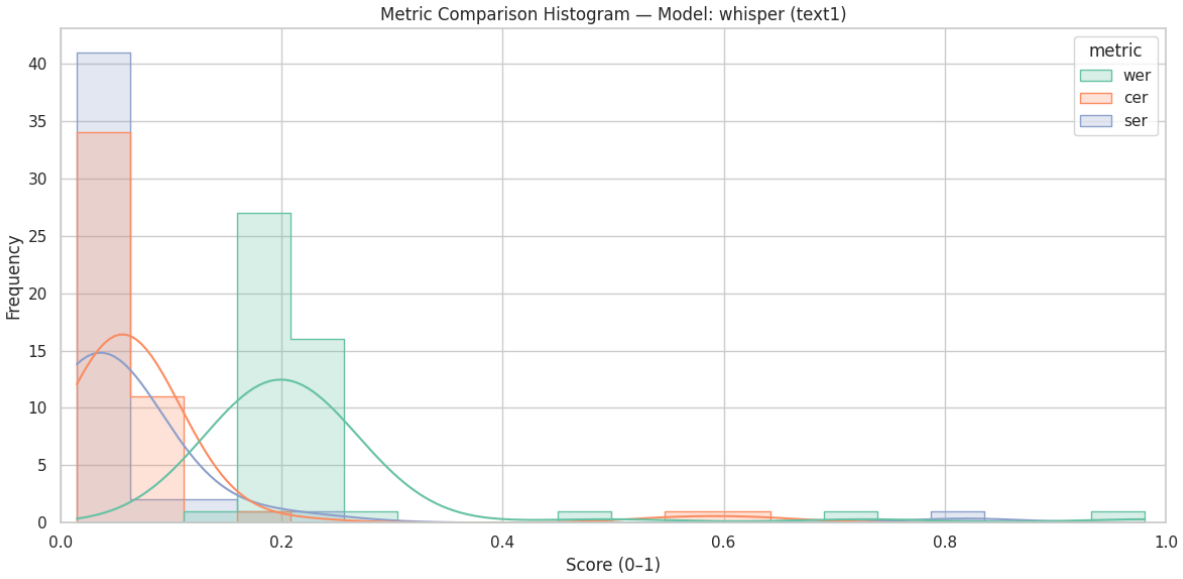

Figure 12: Histogram of all error rates for text type 1 (Q17) on OpenAI's Whisper model

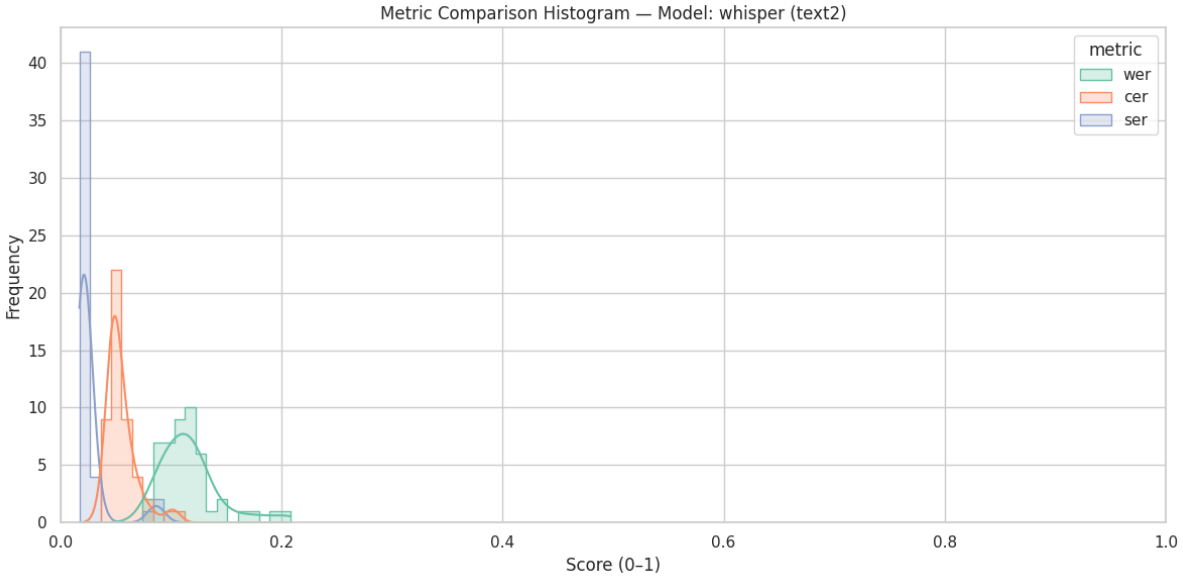

Figure 13: Histogram of all error rates for text type 2 (Q18) on OpenAI's Whisper model

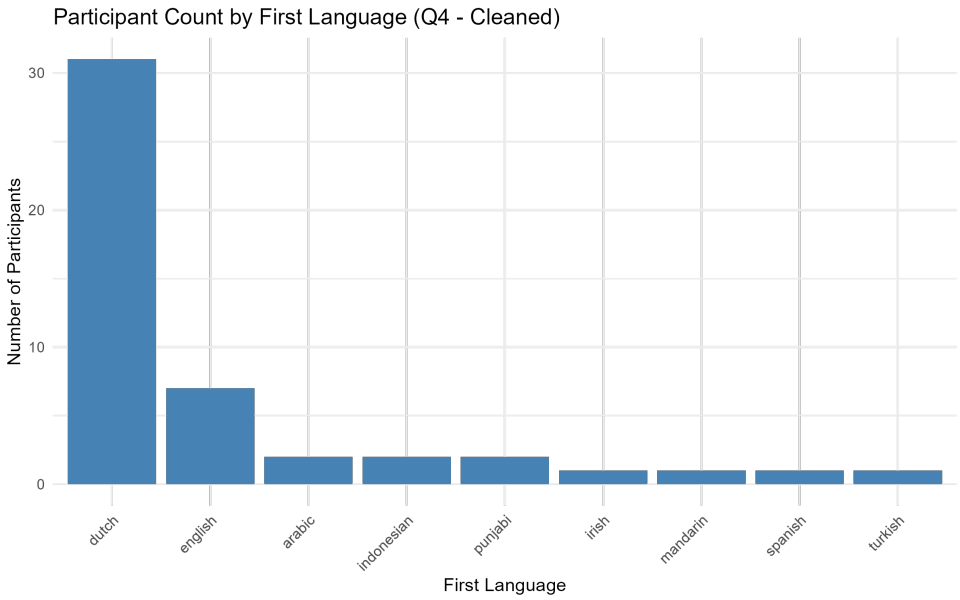

Figure 14: Bar chart of listed first best languages (Q4) after regrouping

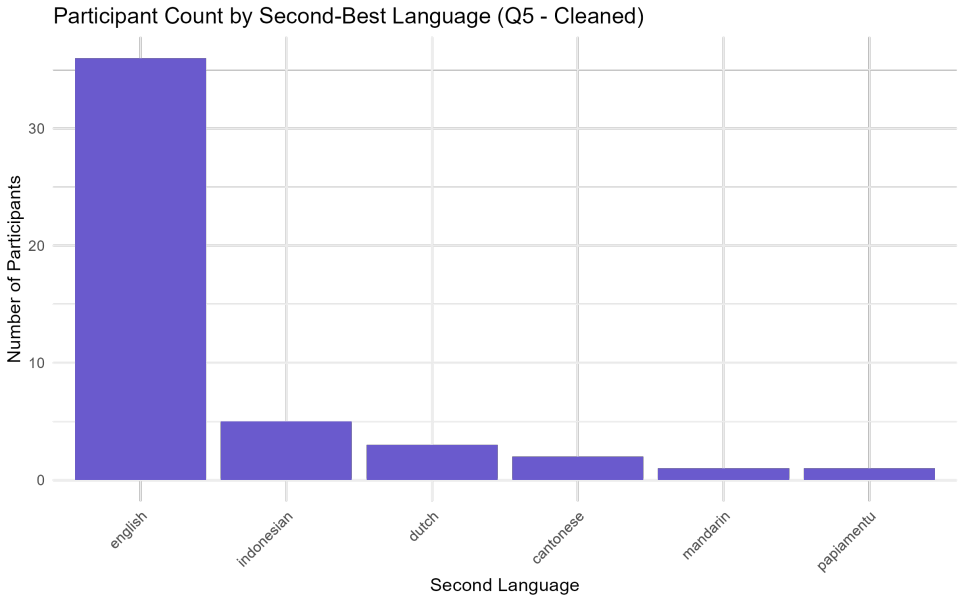

Figure 15: Bar chart of listed second best languages (Q5) after regrouping

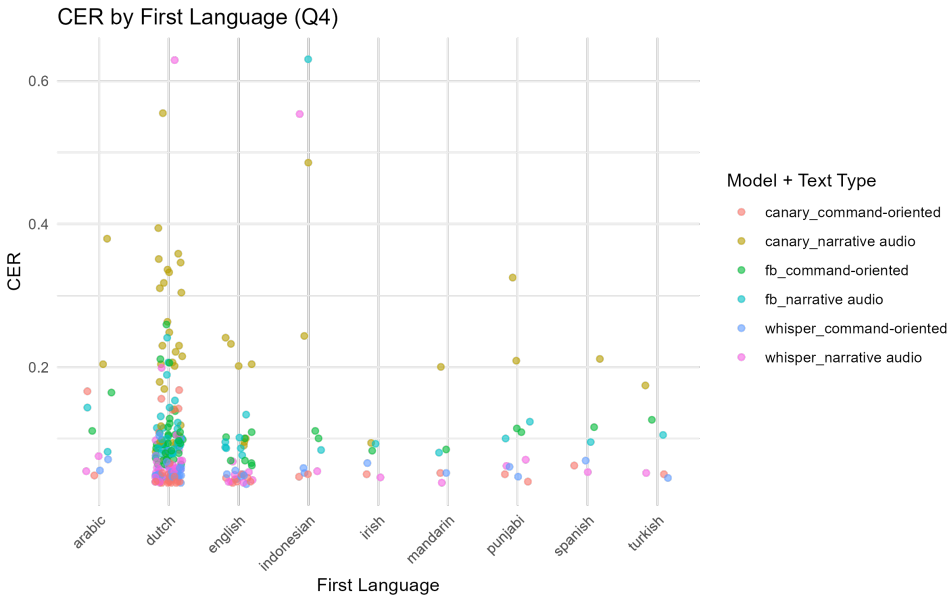

Figure 16: Scatterplot of CER divided based on first best language

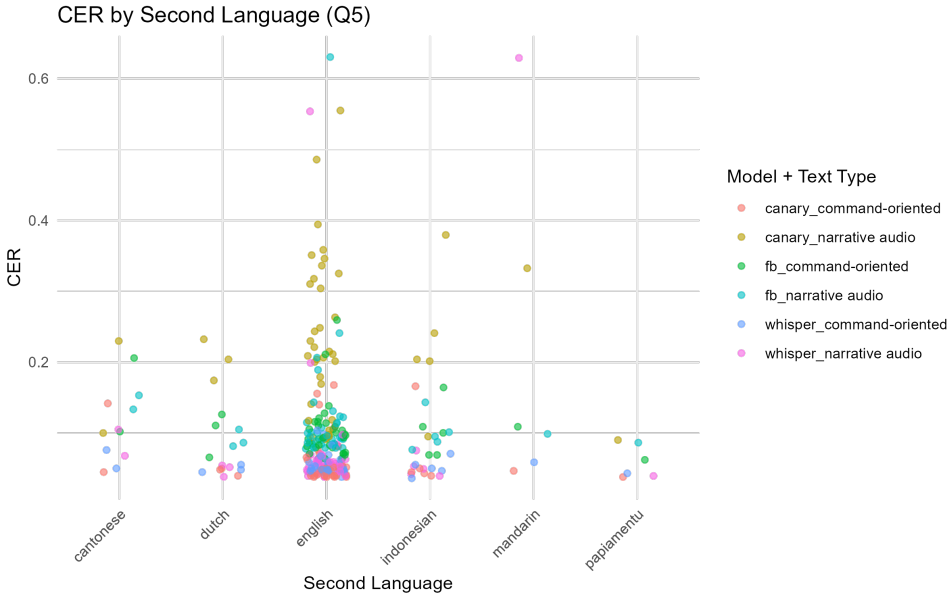

Figure 17: Scatterplot of CER divided based on second best language

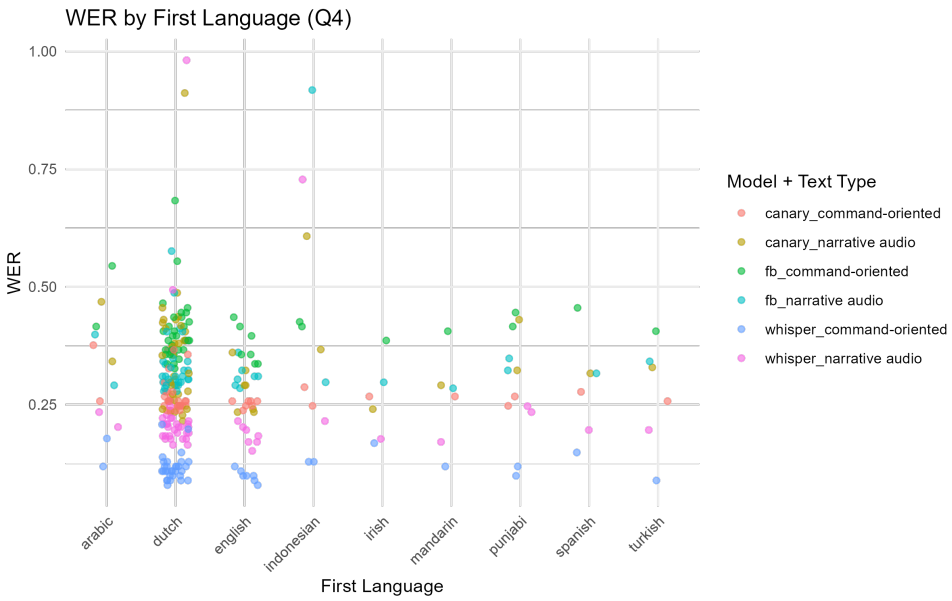

Figure 18: Scatterplot of WER divided based on first best language

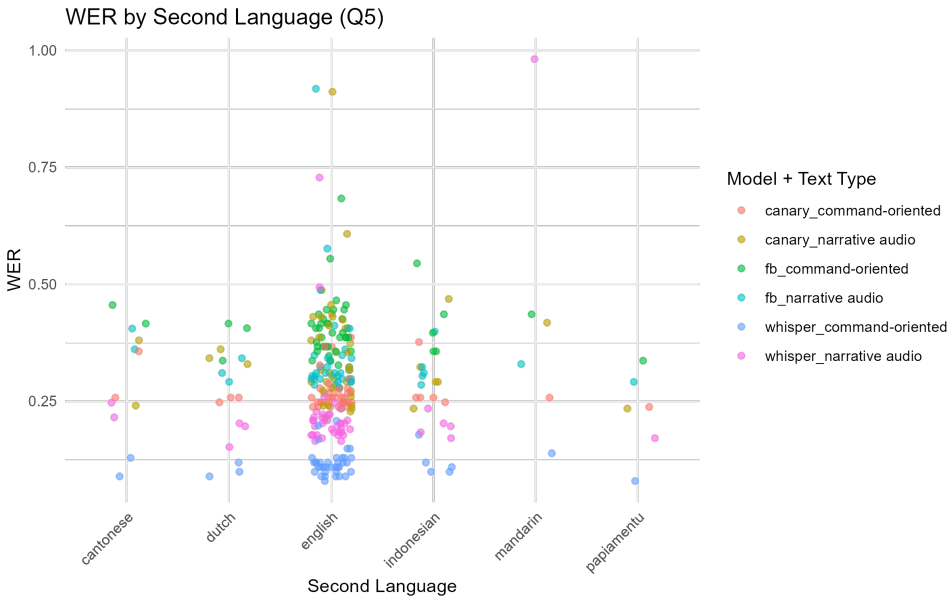

Figure 19: Scatterplot of WER divided based on second best language

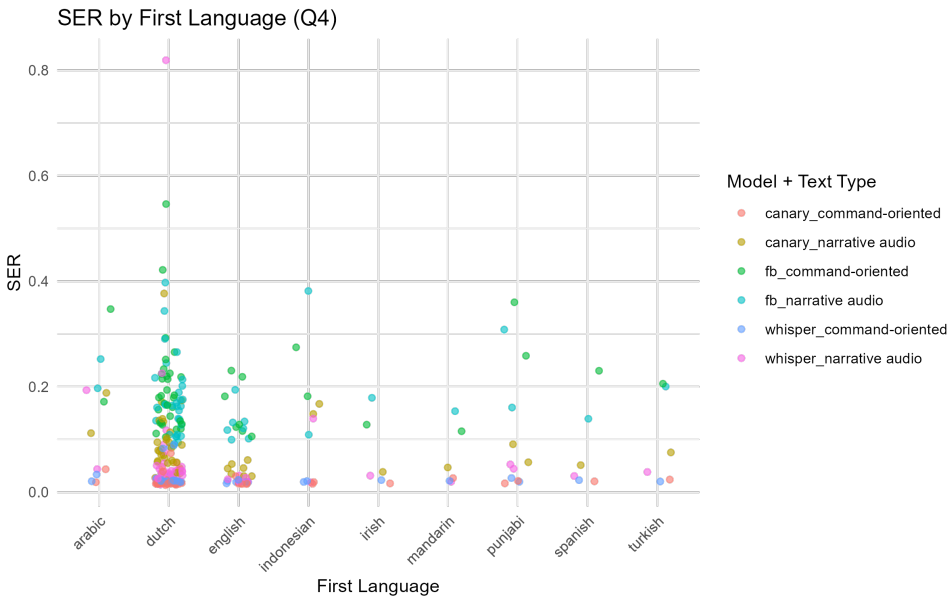

Figure 20: Scatterplot of SER divided based on first best language

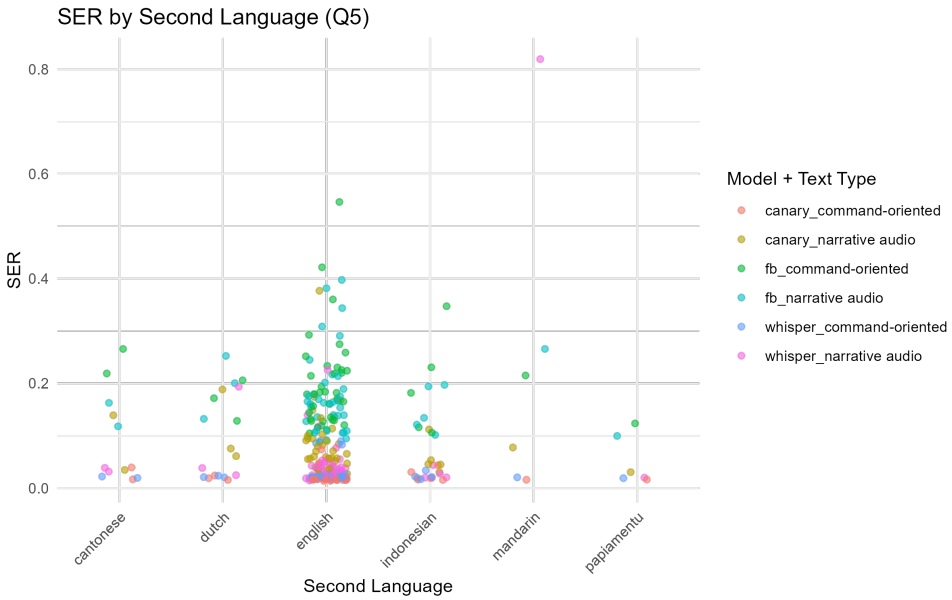

Figure 21: Scatterplot of SER divided based on second best language

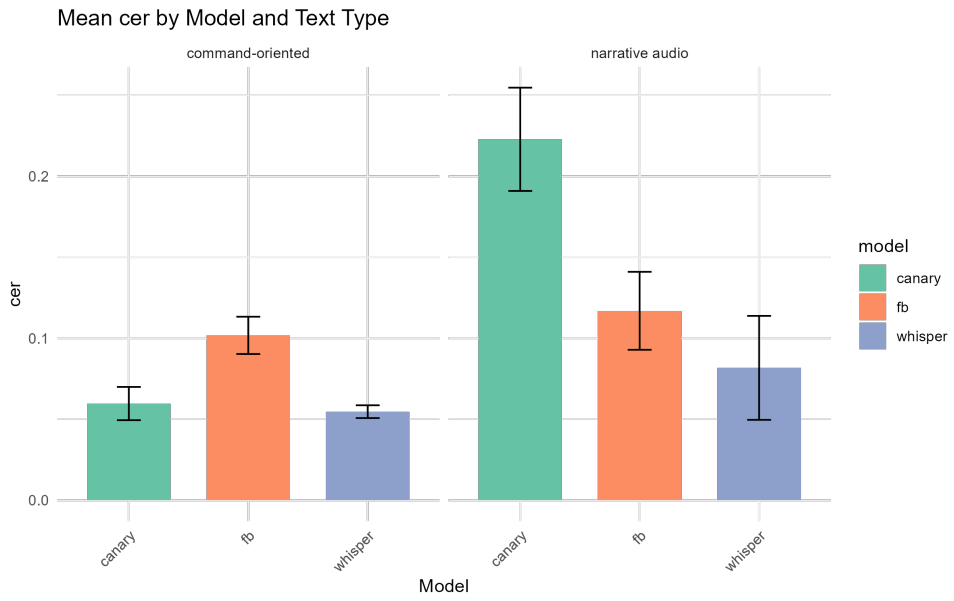

Figure 22: Bar chart of mean CER per model and text type with confidence interval

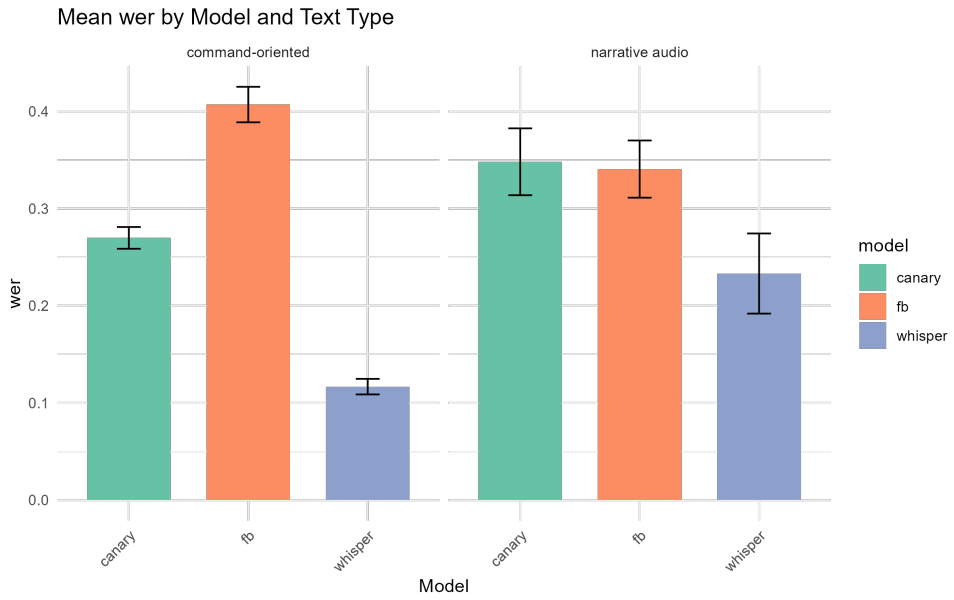

Figure 23: Bar chart of mean WER per model and text type with confidence interval

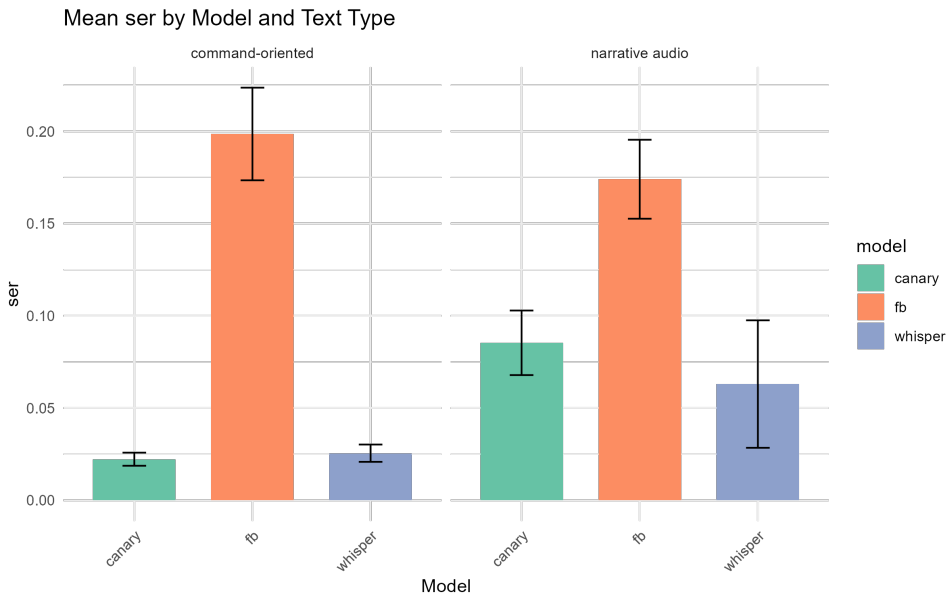

Figure 24: Bar chart of mean SER per model and text type with confidence interval

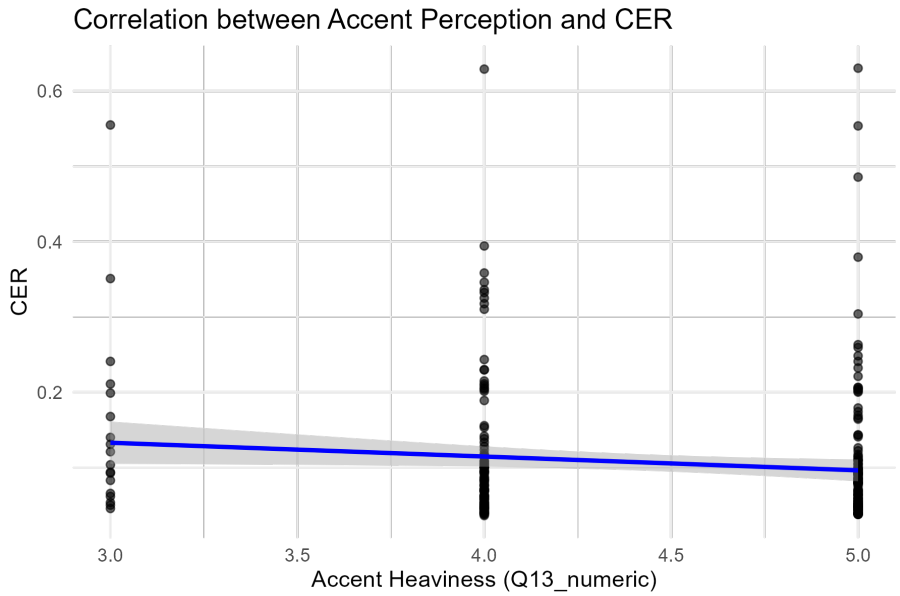

Figure 25: Linear regression plot of correlation between Self-perceived accent heaviness and CER

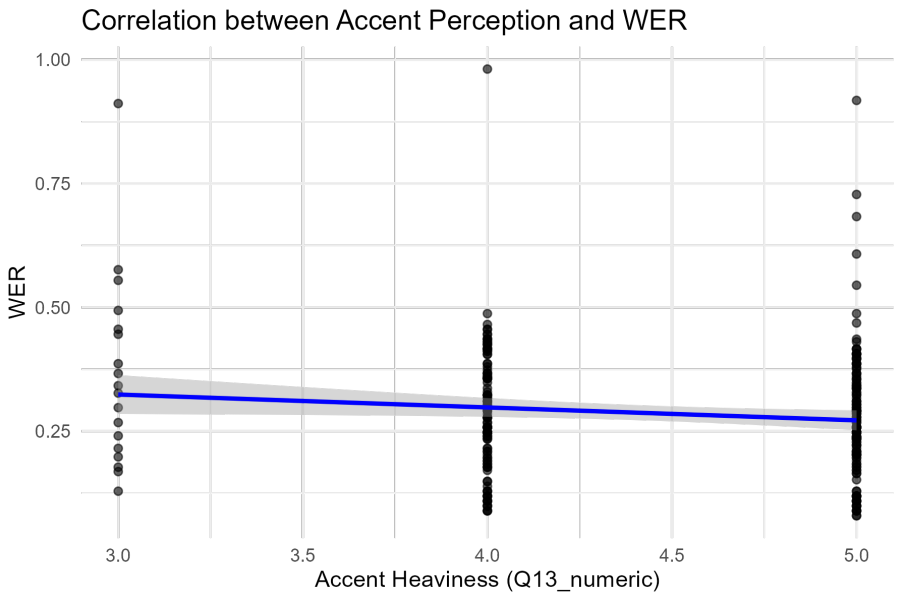

Figure 26: Linear regression plot of correlation between Self-perceived accent heaviness and WER

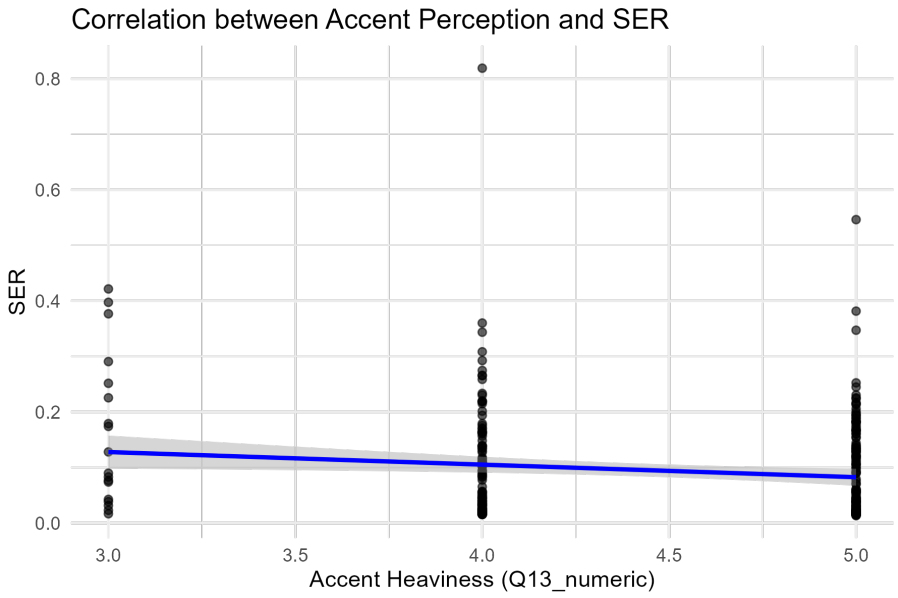

Figure 27: Linear regression plot of correlation between Self-perceived accent heaviness and SER

C Appendix: ASR Model Implementations (Jupyter code)

C.1 NVIDIA/canary-1b implementation

```
# -*- coding: utf-8 -*-
"""canary2.0.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1
       B7REaf7Gf2iD905PkE1fYN1jtDbeLFyx
"""

!pip install protobuf==3.20.3
!pip install huggingface_hub==0.19.4
!pip install transformers==4.35.2
!pip install nemo_toolkit['asr']

!pip install numpy==1.24.3 numba --force-reinstall

import numpy
import os
from os import listdir
from os.path import isfile, join

from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')

# The directory path (adjust the path to the folder in Google Drive)
audiopath_1 = '/content/drive/My Drive/audio1'
audiopath_2 = '/content/drive/My Drive/audio2'
textpath_1 = '/content/drive/My Drive/text1'
textpath_2 = '/content/drive/My Drive/text2'

# Ensure output directories exist
os.makedirs(textpath_1, exist_ok=True)
os.makedirs(textpath_2, exist_ok=True)

from nemo.collections.asr.models import EncDecMultiTaskModel

# load model
canary_model = EncDecMultiTaskModel.from_pretrained('nvidia/canary-1b'
```

```
    )

# update dcode params
decode_cfg = canary_model.cfg.decoding
decode_cfg.beam.beam_size = 1
canary_model.change_decoding_strategy(decode_cfg)

# Function to fix audio format
def fix_audio_format(file_path):
    waveform, sr = torchaudio.load(file_path)
    if waveform.shape[0] > 1:
        waveform = waveform.mean(dim=0, keepdim=True)
    elif waveform.ndim == 1:
        waveform = waveform.unsqueeze(0)

    if sr != 16000:
        waveform = torchaudio.transforms.Resample(orig_freq=sr,
            new_freq=16000)(waveform)

    fixed_path = file_path.replace(".wav", "_fixed.wav")
    torchaudio.save(fixed_path, waveform, 16000)
    return fixed_path

!pip install torch
!pip install torchaudio

import torchaudio
import torch

# Specify the filename you want to process
filename = "[filename]"

# Construct full path to the file
file_path = os.path.join(audiopath_1, filename)

# Ensure the file exists and is a valid audio format
if os.path.isfile(file_path) and filename.lower().endswith(('.wav', '.
    flac', '.mp3', '.m4a')):
    # Fix the audio format
    fixed_path = fix_audio_format(file_path)

    # Ensure fixed file exists
    if os.path.isfile(fixed_path):
        # Transcribe using canary_model
        prediction = canary_model.transcribe(audio=[fixed_path])[0]
```

```
        cleaned = prediction.text.strip().lower().capitalize()

        # Generate output path
        base_name = os.path.splitext(os.path.basename(fixed_path))[0]
        output_file = os.path.join(textpath_1, f"
            canary_transcription_1_{base_name}.txt")

        # Save transcription
        with open(output_file, "w", encoding="utf-8") as f:
            f.write(cleaned + "\n")
    else:
        print("Fixed file not found:", fixed_path)
else:
    print("Specified file is invalid or does not exist:", file_path)

# Specify the filename you want to process
filename = "[filename]"

# Construct full path to the file
file_path = os.path.join(audiopath_2, filename)

# Ensure the file exists and is a valid audio format
if os.path.isfile(file_path) and filename.lower().endswith(('.wav', '.
    flac', '.mp3', '.m4a')):
    # Fix the audio format
    fixed_path = fix_audio_format(file_path)

    # Ensure fixed file exists
    if os.path.isfile(fixed_path):
        # Transcribe using canary_model
        prediction = canary_model.transcribe(audio=[fixed_path])[0]
        cleaned = prediction.text.strip().lower().capitalize()

        # Generate output path
        base_name = os.path.splitext(os.path.basename(fixed_path))[0]
        output_file = os.path.join(textpath_2, f"
            canary_transcription_2_{base_name}.txt")

        # Save transcription
        with open(output_file, "w", encoding="utf-8") as f:
            f.write(cleaned + "\n")
    else:
        print("Fixed file not found:", fixed_path)
else:
    print("Specified file is invalid or does not exist:", file_path)
```

C.2 facebook/wav2vec2-large-960h implementation

```
# -*- coding: utf-8 -*-
"""facebook_hf.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/16
       T7h9M4kAX4DC_HqmpjCZ0ON0a9mgnCz
"""

import os
from os import listdir
from os.path import isfile, join
from transformers import Wav2Vec2Processor, Wav2Vec2ForCTC
import torchaudio
import torch

from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')

# The directory path (adjust the path to the folder in Google Drive)
audiopath_1 = '/content/drive/My Drive/audio1'
audiopath_2 = '/content/drive/My Drive/audio2'
output_dir_1 = '/content/drive/My Drive/text1'
output_dir_2 = '/content/drive/My Drive/text2'

# Create output directories if they don't exist
os.makedirs(output_dir_1, exist_ok=True)
os.makedirs(output_dir_2, exist_ok=True)

# Collect audio files
audio_files_1 = [
    os.path.join(audiopath_1, f)
    for f in os.listdir(audiopath_1)
    if os.path.isfile(os.path.join(audiopath_1, f)) and f.lower().
       endswith(('.wav', '.flac', '.mp3', '.m4a'))
]

audio_files_2 = [
    os.path.join(audiopath_2, f)  # Construct full file path
    for f in os.listdir(audiopath_2)
    if os.path.isfile(os.path.join(audiopath_2, f)) and f.lower().
```

```
        endswith(('.wav', '.flac', '.mp3', '.m4a'))
]

# Load Facebook Wav2Vec2 model
processor = Wav2Vec2Processor.from_pretrained("facebook/wav2vec2-large
    -960h")
fb_model = Wav2Vec2ForCTC.from_pretrained("facebook/wav2vec2-large-960
    h")

def transcribe_Facebook(mypath):
    waveform, sample_rate = torchaudio.load(mypath)

    if waveform.ndim > 1:
        waveform = waveform[0, :]
    waveform = waveform.squeeze()

    resampler = torchaudio.transforms.Resample(orig_freq=sample_rate,
        new_freq=16000)
    waveform = resampler(waveform)

    input_values = processor(waveform, sampling_rate=16000,
        return_tensors="pt").input_values

    with torch.no_grad():
        logits = fb_model(input_values).logits

    predicted_ids = torch.argmax(logits, dim=-1)
    transcript = processor.batch_decode(predicted_ids)[0]
    return transcript

# Transcribe and save audio1 -> text1
for audiopath in audio_files_1:
    transcription = transcribe_Facebook(audiopath)
    base_name = os.path.splitext(os.path.basename(audiopath))[0]
    filename = f"fb_transcription_1_{base_name}.txt"
    output_path = os.path.join(output_dir_1, filename)

    cleaned = transcription.strip().lower().capitalize()
    with open(output_path, "w", encoding="utf-8") as f:
        f.write(cleaned + "\n")

# Transcribe and save audio2 -> text2
for audiopath in audio_files_2:
    transcription = transcribe_Facebook(audiopath)
    base_name = os.path.splitext(os.path.basename(audiopath))[0]
```

```
    filename = f"fb_transcription_2_{base_name}.txt"
    output_path = os.path.join(output_dir_2, filename)

    cleaned = transcription.strip().lower().capitalize()
    with open(output_path, "w", encoding="utf-8") as f:
        f.write(cleaned + "\n")

"""tussendoor"""

# File names (add the correct extension)
file_1 = "R_2GKyoIqEiRRcFmO_WhatsApp Audio 2025-04-13 at 14.10.13.mp3"
file_2 = "R_2GKyoIqEiRRcFmO_WhatsApp Audio 2025-04-13 at 14.16.02.mp3"

# Full paths
audio_file_1 = os.path.join(audiopath_1, file_1)
audio_file_2 = os.path.join(audiopath_2, file_2)

# Load model
processor = Wav2Vec2Processor.from_pretrained("facebook/wav2vec2-large
    -960h")
fb_model = Wav2Vec2ForCTC.from_pretrained("facebook/wav2vec2-large-960
    h")

# Transcription function
def transcribe_Facebook(filepath):
    waveform, sample_rate = torchaudio.load(filepath)
    if waveform.ndim > 1:
        waveform = waveform[0, :]
    waveform = waveform.squeeze()

    resampler = torchaudio.transforms.Resample(orig_freq=sample_rate,
        new_freq=16000)
    waveform = resampler(waveform)

    input_values = processor(waveform, sampling_rate=16000,
        return_tensors="pt").input_values

    with torch.no_grad():
        logits = fb_model(input_values).logits

    predicted_ids = torch.argmax(logits, dim=-1)
    transcript = processor.batch_decode(predicted_ids)[0]
    return transcript.strip().lower().capitalize()

# Transcribe and save file 1
```

```
transcript_1 = transcribe_Facebook(audio_file_1)
out_path_1 = os.path.join(output_dir_1, f"fb_transcription_{os.path.
    splitext(file_1)[0]}.txt")
with open(out_path_1, "w", encoding="utf-8") as f:
    f.write(transcript_1 + "\n")

# Transcribe and save file 2
transcript_2 = transcribe_Facebook(audio_file_2)
out_path_2 = os.path.join(output_dir_2, f"fb_transcription_{os.path.
    splitext(file_2)[0]}.txt")
with open(out_path_2, "w", encoding="utf-8") as f:
    f.write(transcript_2 + "\n")
```

C.3 OpenAI's Whisper implementation

```
# -*- coding: utf-8 -*-
"""Whisper.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1
        u8kxXXqbJvK33SNrXPMStOSZoHhVvSxH
"""

!pip install git+https://github.com/openai/whisper.git

!pip install --upgrade --no-deps --force-reinstall git+https://github.
    com/openai/whisper.git

!sudo apt update && sudo apt install ffmpeg

import whisper

model_whisper = whisper.load_model("medium")

import os
from os import listdir
from os.path import isfile, join

from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')
```

```
# The directory path (adjust the path to the folder in Google Drive)
audiopath_1 = '/content/drive/My Drive/audio1'
audiopath_2 = '/content/drive/My Drive/audio2'
textpath_1 = '/content/drive/My Drive/text1/text1'
textpath_2 = '/content/drive/My Drive/text2/text2'

# List all files in the directories and filter for audio files
audio_files_1 = [
    os.path.join(audiopath_1, f)
    for f in os.listdir(audiopath_1)
    if os.path.isfile(os.path.join(audiopath_1, f)) and f.lower().
       endswith(('.wav', '.flac', '.mp3', '.m4a'))
]

audio_files_2 = [
    os.path.join(audiopath_2, f)
    for f in os.listdir(audiopath_2)
    if os.path.isfile(os.path.join(audiopath_2, f)) and f.lower().
       endswith(('.wav', '.flac', '.mp3', '.m4a'))
]

# Make sure the output directories exist
os.makedirs(textpath_1, exist_ok=True)
os.makedirs(textpath_2, exist_ok=True)

# Process audio files in audiopath_1
if os.path.isdir(audiopath_1):
    for root, dirs, files in os.walk(audiopath_1):
        for filename in files:
            if filename.lower().endswith(('.wav', '.flac', '.mp3', '.
               m4a')):
                filepath = os.path.join(root, filename)
                result = model_whisper.transcribe(filepath, fp16=False
                   )
                transcript = result["text"].strip()

                base_name = os.path.splitext(filename)[0]
                out_file = os.path.join(textpath_1, f"
                   whisper_transcription_1_{base_name}.txt")
                with open(out_file, "w", encoding="utf-8") as f:
                    f.write(transcript + "\n")

# Process audio files in audiopath_2
if os.path.isdir(audiopath_2):
    for root, dirs, files in os.walk(audiopath_2):
```

```
for filename in files:
    if filename.lower().endswith(('.wav', '.flac', '.mp3', '.
       m4a')):
        filepath = os.path.join(root, filename)
        result = model_whisper.transcribe(filepath, fp16=False
            )
        transcript = result["text"].strip()

        base_name = os.path.splitext(filename)[0]
        out_file = os.path.join(textpath_2, f"
            whisper_transcription_2_{base_name}.txt")
        with open(out_file, "w", encoding="utf-8") as f:
            f.write(transcript + "\n")
```

D Appendix: Metric Calculation and Plotting (Jupyter code)

```
# -*- coding: utf-8 -*-
"""metrics.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1
        WTqexqxb_tFsoBcF0UtSbo7AvJfd1Txx
"""

!pip install jiwer sentence-transformers

import os
import re
import pandas as pd
from jiwer import wer, cer
from sentence_transformers import SentenceTransformer, util

# Reference texts
reference_1 = "Alice waited a little, half expecting to see it again, but it did not appear, and after a minute or two she walked on in the direction in which the March Hare was said to live. Ive seen hatters before, she said to herself; the March Hare will be much the most interesting, and perhaps as this is May it wont be raving mad at least not so mad as it was in March. As she said this, she looked up, and there was the Cat again, sitting on a branch of a tree. Did you say pig, or fig? said the Cat. I said pig, replied Alice; and I wish you wouldnt keep appearing and vanishing so suddenly: you make one quite giddy. All right, said the Cat; and this time it vanished quite slowly, beginning with the end of the tail, and ending with the grin, which remained some time after the rest of it had gone."
reference_2 = "Set an alarm for seven o'clock. Explain this part using the following mathematical equation. What is the solution to this question? Find nearby Italian restaurants that provide gluten-free and vegetarian options. What is the name of the song that contains the following text: Look at me now, will I ever learn? I don't know how, but I suddenly lose control. Send a text message to Tim saying, I will be there in twenty minutes. Set a reminder for a hairsalon appointment on April fiftheenth at one O'clock. Find the best cafe near the Eiffel Tower and give the directions to it."
```

```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Set paths
folders_with_references = {
    "/content/drive/MyDrive/text1": reference_1,
    "/content/drive/MyDrive/text2": reference_2
}
csv_path = "/content/drive/MyDrive/csv/"

# Load Sentence Embedding model
semantic_model = SentenceTransformer('paraphrase-MiniLM-L6-v2')

import os
import re
import pandas as pd
from jiwer import wer, cer
from sentence_transformers import SentenceTransformer, util

def load_transcriptions(folder_ref_map):
    # Map folder names to text types
    foldername_to_text_type = {
        "text1": "narrative-audio",
        "text2": "command-oriented"
    }

    all_data = []
    for folder, reference_text in folder_ref_map.items():  # Iterate
       through folder and reference text
        # Extract folder name only (last part of path)
        folder_name = os.path.basename(os.path.normpath(folder))

        # Determine text_type based on folder name
        text_type = foldername_to_text_type.get(folder_name, "unknown"
           )

        for fname in sorted(os.listdir(folder)):
            if fname.endswith(".txt"):
                full_path = os.path.join(folder, fname)
                with open(full_path, "r", encoding="utf-8") as f:
                    pred = f.read().strip()

                # Filename pattern: <model>_transcription_<refid>_<
                   audio>.txt
```

```
                match = re.match(r"([A-Za-z]+)_transcription_([0-9]+)_
                    (.*)\.txt", fname)
                if not match:
                    print(f"      - Skipping: Unexpected filename
                        format:    {fname}")
                    continue
                model, ref_id, audio_id = match.groups()

                all_data.append({
                    "filename": fname,
                    "model": model.lower(),
                    "reference_id": ref_id,
                    "audio_id": audio_id,
                    "text_type": text_type,
                    "prediction": pred,  # Add the prediction text
                        here
                    "reference": reference_text
                })
    return pd.DataFrame(all_data)

# Compute metrics
def compute_metrics(df):
    wers, cers, sers = [], [], []
    for _, row in df.iterrows():
        wers.append(wer(row["reference"], row["prediction"]))
        cers.append(cer(row["reference"], row["prediction"]))

        # Semantic similarity
        ref_emb = semantic_model.encode(row["reference"],
            convert_to_tensor=True)
        pred_emb = semantic_model.encode(row["prediction"],
            convert_to_tensor=True)
        similarity = util.cos_sim(ref_emb, pred_emb).item()
        sers.append(1 - similarity)

    df["wer"] = wers
    df["cer"] = cers
    df["ser"] = sers
    return df

# Run the pipeline
df_transcriptions = load_transcriptions(folders_with_references)
df_with_metrics = compute_metrics(df_transcriptions)

# Save results
```

```
output_file = os.path.join(csv_path, "all_models_metrics.csv")
df_with_metrics.to_csv(output_file, index=False)

print("    - Metrics computed and saved to:", output_file)
print(df_with_metrics.head())

"""# Graphs"""

!pip install matplotlib

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

sns.set(style="whitegrid", palette="muted")

# Create output directory if needed
os.makedirs(csv_path, exist_ok=True)

metrics = ["wer", "cer", "ser"]
models = df_with_metrics["model"].unique()

df_with_metrics["dataset"] = df_with_metrics["reference_id"].apply(
    lambda x: "text1" if x == "1" else "text2"
)

save_path = "/content/drive/MyDrive/plots"

# Create seperate scatterplots per metric per model per text type
for dataset_label in ["text1", "text2"]:
    dataset_df = df_with_metrics[df_with_metrics["dataset"] ==
       dataset_label]

    for metric in metrics:
        for model in models:
            model_subset = dataset_df[dataset_df["model"] == model]

            plt.figure(figsize=(18, 10))
            sns.scatterplot(
                data=model_subset,
                x="audio_id",
                y=metric,
                s=100,
            )
```

```
            plt.title(f"{metric.upper()} for model: {model} ({
                dataset_label})")
            plt.ylabel(f"{metric.upper()} (0  1  )")
            plt.xlabel("Audio Filename")

            # Improve x-axis spacing
            unique_labels = model_subset["audio_id"].unique()
            x_positions = np.arange(len(unique_labels))
            plt.xticks(x_positions, unique_labels, rotation=60, ha='
                right')

            plt.yticks(np.arange(0, 1.1, 0.1))
            plt.ylim(0, 1)
            plt.grid(True, axis='y', linestyle='--', alpha=0.7)

            plt.tight_layout()

            # Save figure to specified path
            filename = f"{metric}_{model}_{dataset_label}.png"
            plt.savefig(os.path.join(save_path, filename))
            plt.show()
            plt.close()

# Create scatterplots with all metrics per model per text type
for dataset_label in ["text1", "text2"]:
    dataset_df = df_with_metrics[df_with_metrics["dataset"] ==
        dataset_label]

    for model in models:
        model_subset = dataset_df[dataset_df["model"] == model]

        plt.figure(figsize=(18, 10))

        unique_labels = model_subset["audio_id"].unique()
        x_positions = np.arange(len(unique_labels))

        for metric in metrics:
            metric_values = []

            for audio_id in unique_labels:
                value = model_subset.loc[model_subset["audio_id"] ==
                    audio_id, metric]
                metric_values.append(value.values[0] if not value.
                    empty else np.nan)
```

```
            plt.scatter(
                x_positions,
                metric_values,
                label=metric.upper(),
                s=100  # Marker size
            )

        plt.xticks(x_positions, unique_labels, rotation=60, ha='right')
        plt.yticks(np.arange(0, 1.1, 0.1))
        plt.ylim(0, 1)
        plt.grid(True, axis='y', linestyle='--', alpha=0.7)

        plt.title(f"All Metrics for Model: {model} ({dataset_label})")
        plt.xlabel("Audio Filename")
        plt.ylabel("Score (0  1  )")
        plt.legend(title="Metric")
        plt.tight_layout()

        # Save figure
        filename = f"all_metrics_{model}_{dataset_label}.png"
        plt.savefig(os.path.join(save_path, filename))
        plt.show()
        plt.close()

# Create seperate histograms per metric per model per text type
for dataset_label in ["text1", "text2"]:
    dataset_df = df_with_metrics[df_with_metrics["dataset"] ==
        dataset_label]

    for model in models:
        model_subset = dataset_df[dataset_df["model"] == model]

        for metric in metrics:
            plt.figure(figsize=(10, 6))
            sns.histplot(
                data=model_subset,
                x=metric,
                kde=True,
                bins=20,
                color='skyblue'
            )

            plt.title(f"{metric.upper()} Histogram     Model: {model}
                ({dataset_label})")
```

```
            plt.xlabel(f"{metric.upper()} (0 1 )")
            plt.ylabel("Frequency")
            plt.xlim(0, 1)
            plt.tight_layout()

            filename = f"{metric}_histogram_{model}_{dataset_label}.
                png"
            plt.savefig(os.path.join(save_path, filename))
            plt.show()
            plt.close()

# Create histograms with all metrics per model per text type
for dataset_label in ["text1", "text2"]:
    dataset_df = df_with_metrics[df_with_metrics["dataset"] ==
        dataset_label]

    for model in models:
        model_subset = dataset_df[dataset_df["model"] == model]

        # Reshape data: convert wide to long format
        long_df = model_subset.melt(
            id_vars=["audio_id"],
            value_vars=metrics,
            var_name="metric",
            value_name="value"
        )

        plt.figure(figsize=(12, 6))
        sns.histplot(
            data=long_df,
            x="value",
            hue="metric",
            bins=20,
            kde=True,
            palette="Set2",
            element="step"
        )

        plt.title(f"Metric Comparison Histogram  Model: {model} ({
            dataset_label})")
        plt.xlabel("Score (0 1 )")
        plt.ylabel("Frequency")
        plt.xlim(0, 1)
        plt.tight_layout()
```

```
        # Save
        filename = f"all_metrics_histogram_{model}_{dataset_label}.png
            "
        plt.savefig(os.path.join(save_path, filename))
        plt.show()
        plt.close()

# Check the number of rows for the 'canary' model for each dataset
for dataset_label in ["text1", "text2"]:
    dataset_df = df_with_metrics[df_with_metrics["dataset"] ==
        dataset_label]
    canary_subset = dataset_df[dataset_df["model"] == "canary"]
    print(f"Number of rows for 'canary' model in {dataset_label}: {len
        (canary_subset)}")
    print(canary_subset)
```

E Appendix: Data analysis and Plotting (RStudio code)

```
# Load required packages
library(dplyr)
library(openxlsx)
library(ggplot2)
library(ggpubr)

# Read the files
df1 <- read.csv("survey_values.csv", stringsAsFactors = FALSE)
df2 <- read.csv("all_models_metrics.csv", stringsAsFactors = FALSE)

# Extract response_ID from filename
df2 <- df2 %>%
  mutate(response_ID = sub(".*(R_[A-Za-z0-9]+).*", "\\1", filename))

# Ensure response_ID is character in both
df1$response_ID <- as.character(df1$ResponseId)
df2$response_ID <- as.character(df2$response_ID)

# Merge all relevant columns from df2
merged_df <- df1 %>%
  left_join(df2, by = "response_ID")


# Define columns to remove
columns_to_remove <- c(
  "StartDate", "EndDate", "Status", "IPAddress", "Progress", "Duration
     ..in.seconds.",
  "Finished", "RecordedDate", "RecipientLastName", "RecipientFirstName
     ", "RecipientEmail",
  "ExternalReference", "LocationLatitude", "LocationLongitude", "
     DistributionChannel",
  "UserLanguage", "Q1", "Q6", "Q7", "Q8", "Q9", "Q10", "Q10_6_TEXT", "
     Q11",
  "Q14", "Q15", "Q16", "Q17_Size", "Q17_Type", "Q18_Size", "Q18_Type"
)

# Clean the merged dataset
merged_df_cleaned <- merged_df %>%
  select(-one_of(intersect(columns_to_remove, names(merged_df))))


# === Two-way ANOVA: model type x text type packages on metrics ===
```

```
anova_result1 <- aov(cer ~ model * text_type, data = merged_df_cleaned
    )
summary(anova_result1)
anova_result2 <- aov(wer ~ model * text_type, data = merged_df_cleaned
    )
summary(anova_result2)
anova_result3 <- aov(ser ~ model * text_type, data = merged_df_cleaned
    )
summary(anova_result3)

to_df <- function(result, metric_name) {
  df <- as.data.frame(summary(result)[[1]])
  df$Effect <- rownames(df)
  rownames(df) <- NULL
  names(df)[which(names(df) == "Pr(>F)")]
  df <- df[, c("Effect", names(df)[1:5])]
  df$Metric <- metric_name
  return(df)
}

anova_df1 <- to_df(anova_result1, "cer")
anova_df2 <- to_df(anova_result2, "wer")
anova_df3 <- to_df(anova_result3, "ser")

# Combine all into one table
all_anova_df <- rbind(anova_df1, anova_df2, anova_df3)

# Save to CSV
write.csv(all_anova_df, "anova_results_all.csv", row.names = FALSE)

# Save to XLSX (single sheet)
write.xlsx(all_anova_df, "anova_results_all.xlsx", rowNames = FALSE)

# Function to calculate mean and CI
summary_stats <- function(data, metric) {
  data %>%
    filter(!is.na(model), !is.na(text_type)) %>%   # <--- Add this
       line
    group_by(model, text_type) %>%
    summarise(
      mean = mean(.data[[metric]], na.rm = TRUE),
      sd = sd(.data[[metric]], na.rm = TRUE),
      n = n(),
      se = sd / sqrt(n),
      ci = qt(0.975, df = n - 1) * se,
```

```
      .groups = "drop"
    ) %>%
    mutate(metric = metric)
}

# Get summaries for all metrics
cer_summary <- summary_stats(merged_df_cleaned, "cer")
wer_summary <- summary_stats(merged_df_cleaned, "wer")
ser_summary <- summary_stats(merged_df_cleaned, "ser")

# Combine all into one dataframe
all_summary <- bind_rows(cer_summary, wer_summary, ser_summary)

# Function to plot for one metric
plot_metric <- function(summary_df, metric_name) {
  ggplot(summary_df %>% filter(metric == metric_name),
         aes(x = model, y = mean, fill = model)) +
    geom_bar(stat = "identity", position = position_dodge(width = 0.9)
        , width = 0.7) +
    geom_errorbar(aes(ymin = mean - ci, ymax = mean + ci),
                  width = 0.2, position = position_dodge(width = 0.9))
                      +
    facet_wrap(~ text_type) +
    labs(title = paste("Mean", metric_name, "by Model and Text Type"),
         y = metric_name, x = "Model") +
    theme_minimal() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
    scale_fill_brewer(palette = "Set2")
}

# Plot each metric
plot_cer <- plot_metric(all_summary, "cer")
ggsave("cer_model_plot.png", plot_cer, width = 8, height = 5)

plot_wer <- plot_metric(all_summary, "wer")
ggsave("wer_model_plot.png", plot_wer, width = 8, height = 5)

plot_ser <- plot_metric(all_summary, "ser")
ggsave("ser_model_plot.png", plot_ser, width = 8, height = 5)


# === Tukey HSD: model types on metrics ===

# Run one-way ANOVA for Tukey HSD
anova_cer <- aov(cer ~ model, data = merged_df_cleaned)
```

```
anova_wer <- aov(wer ~ model, data = merged_df_cleaned)
anova_ser <- aov(ser ~ model, data = merged_df_cleaned)

# Run Tukey HSD post-hoc tests
tukey_cer <- TukeyHSD(anova_cer)
tukey_wer <- TukeyHSD(anova_wer)
tukey_ser <- TukeyHSD(anova_ser)

# Convert each to data frame and add identifiers
df_cer <- as.data.frame(tukey_cer$model)
df_cer$Comparison <- rownames(df_cer)
df_cer$Metric <- "cer"

df_wer <- as.data.frame(tukey_wer$model)
df_wer$Comparison <- rownames(df_wer)
df_wer$Metric <- "wer"

df_ser <- as.data.frame(tukey_ser$model)
df_ser$Comparison <- rownames(df_ser)
df_ser$Metric <- "ser"

# Combine all into one table
tukey_all <- rbind(df_cer, df_wer, df_ser)

# Reorder columns
tukey_all <- tukey_all[, c("Metric", "Comparison", "diff", "lwr", "upr
    ", "p-adj")]

# Save as CSV
write.csv(tukey_all, "tukey_model_comparisons.csv", row.names = FALSE)

# Save as XLSX
write.xlsx(tukey_all, "tukey_model_comparisons.xlsx", rowNames = FALSE
    )


summary_stats_tukey <- function(data, metric) {
  data %>%
    filter(!is.na(model)) %>%
    group_by(model) %>%
    summarise(
      mean = mean(.data[[metric]], na.rm = TRUE),
      sd = sd(.data[[metric]], na.rm = TRUE),
      n = n(),
      se = sd / sqrt(n),
```

```
      ci = qt(0.975, df = n - 1) * se,
      .groups = "drop"
    ) %>%
    mutate(metric = metric)
}

# Create summaries for tukey plots (model only)
cer_tukey_summary <- summary_stats_tukey(merged_df_cleaned, "cer")
wer_tukey_summary <- summary_stats_tukey(merged_df_cleaned, "wer")
ser_tukey_summary <- summary_stats_tukey(merged_df_cleaned, "ser")

tukey_summary_all <- bind_rows(cer_tukey_summary, wer_tukey_summary,
   ser_tukey_summary)

# Then plot with this new summary
plot_tukey_metric <- function(summary_df, metric_name) {
  ggplot(summary_df %>% filter(metric == metric_name),
         aes(x = model, y = mean, fill = model)) +
    geom_bar(stat = "identity", position = position_dodge(width = 0.7)
        , width = 0.6) +
    geom_errorbar(aes(ymin = mean - ci, ymax = mean + ci),
                  width = 0.2, position = position_dodge(width = 0.7))
                      +
    labs(
      title = paste("Mean", toupper(metric_name), "by Model with 95%
         CI"),
      x = "Model",
      y = metric_name
    ) +
    theme_minimal() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
    scale_fill_brewer(palette = "Set2") +
    guides(fill = "none")
}

# Plot and save
plot_cer_tukey <- plot_tukey_metric(tukey_summary_all, "cer")
ggsave("tukey_cer_barplot.png", plot_cer_tukey, width = 8, height = 5)

plot_wer_tukey <- plot_tukey_metric(tukey_summary_all, "wer")
ggsave("tukey_wer_barplot.png", plot_wer_tukey, width = 8, height = 5)

plot_ser_tukey <- plot_tukey_metric(tukey_summary_all, "ser")
ggsave("tukey_ser_barplot.png", plot_ser_tukey, width = 8, height = 5)
```

```
# === Two-way ANOVA: text type X Best Language (Q4 & Q5) ===


# Standardize Q4 values
merged_df_cleaned$Q4_clean <- tolower(trimws(merged_df_cleaned$Q4))
merged_df_cleaned$Q4_clean <- recode(merged_df_cleaned$Q4_clean,
                                     "nederlands" = "dutch",
                                     "english" = "english",
                                     "englisch" = "english",
                                     "arabic" = "arabic",
                                     "netherlands" = "dutch",
                                     "dutch" = "dutch",
                                     "indonesian" = "indonesian",
                                     "bahasa indonesia" = "indonesian"
)

# Standardize Q5 values
merged_df_cleaned$Q5_clean <- tolower(trimws(merged_df_cleaned$Q5))
merged_df_cleaned$Q5_clean <- recode(merged_df_cleaned$Q5_clean,
                                     "cantonese" = "cantonese",
                                     "chinees" = "mandarin",
                                     "english" = "english",
                                     "englisch" = "english",
                                     "bahasa indonesia" = "indonesian"
                                       ,
                                     "indonesia" = "indonesian",
                                     "indonesian" = "indonesian",
                                     "dutch" = "dutch",
                                     "nederlands" = "dutch",
                                     "papiamentu" = "papiamentu"
)

View(merged_df_cleaned)

# Count and adjust Q4_clean frequencies
q4_clean_counts <- as.data.frame(table(merged_df_cleaned$Q4_clean))
colnames(q4_clean_counts) <- c("Language", "Count")
q4_clean_counts$Adjusted_Count <- q4_clean_counts$Count / 6

# Bar chart for Q4
ggplot(q4_clean_counts, aes(x = reorder(Language, -Adjusted_Count), y
   = Adjusted_Count)) +
  geom_bar(stat = "identity", fill = "#4682B4") +
  labs(title = "Participant Count by First Language (Q4 - Cleaned)",
```

```
       x = "First Language", y = "Number of Participants") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
ggsave("bar_q4_cleaned_counts.png", width = 8, height = 5)


# Count and adjust Q5_clean frequencies
q5_clean_counts <- as.data.frame(table(merged_df_cleaned$Q5_clean))
colnames(q5_clean_counts) <- c("Language", "Count")
q5_clean_counts$Adjusted_Count <- q5_clean_counts$Count / 6

# Bar chart for Q5
ggplot(q5_clean_counts, aes(x = reorder(Language, -Adjusted_Count), y
   = Adjusted_Count)) +
  geom_bar(stat = "identity", fill = "#6A5ACD") +
  labs(title = "Participant Count by Second-Best Language (Q5 -
     Cleaned)",
       x = "Second Language", y = "Number of Participants") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
ggsave("bar_q5_cleaned_counts.png", width = 8, height = 5)



# TWO-WAY ANOVA for Q4 (Best first language)
anova_q4_cer <- aov(cer ~ text_type * Q4_clean, data = merged_df_
   cleaned)
anova_q4_wer <- aov(wer ~ text_type * Q4_clean, data = merged_df_
   cleaned)
anova_q4_ser <- aov(ser ~ text_type * Q4_clean, data = merged_df_
   cleaned)

# TWO-WAY ANOVA for Q5 (Best second-best language)
anova_q5_cer <- aov(cer ~ text_type * Q5_clean, data = merged_df_
   cleaned)
anova_q5_wer <- aov(wer ~ text_type * Q5_clean, data = merged_df_
   cleaned)
anova_q5_ser <- aov(ser ~ text_type * Q5_clean, data = merged_df_
   cleaned)

# Convert ANOVA summaries to data frames
anova_q4_df_cer <- to_df(anova_q4_cer, "cer")
anova_q4_df_wer <- to_df(anova_q4_wer, "wer")
anova_q4_df_ser <- to_df(anova_q4_ser, "ser")
```

```
anova_q5_df_cer <- to_df(anova_q5_cer, "cer")
anova_q5_df_wer <- to_df(anova_q5_wer, "wer")
anova_q5_df_ser <- to_df(anova_q5_ser, "ser")

# Combine and save ANOVA results
anova_q4_all <- rbind(anova_q4_df_cer, anova_q4_df_wer, anova_q4_df_
    ser)
anova_q5_all <- rbind(anova_q5_df_cer, anova_q5_df_wer, anova_q5_df_
    ser)

write.csv(anova_q4_all, "anova_q4_language_results.csv", row.names =
    FALSE)
write.csv(anova_q5_all, "anova_q5_language_results.csv", row.names =
    FALSE)

write.xlsx(anova_q4_all, "anova_q4_language_results.xlsx", rowNames =
    FALSE)
write.xlsx(anova_q5_all, "anova_q5_language_results.xlsx", rowNames =
    FALSE)


# POST-HOC TUKEY TESTS for Q4
tukey_q4_cer <- TukeyHSD(aov(cer ~ Q4_clean, data = merged_df_cleaned)
    )
tukey_q4_wer <- TukeyHSD(aov(wer ~ Q4_clean, data = merged_df_cleaned)
    )
tukey_q4_ser <- TukeyHSD(aov(ser ~ Q4_clean, data = merged_df_cleaned)
    )

df_q4_cer <- as.data.frame(tukey_q4_cer$Q4_clean)
df_q4_wer <- as.data.frame(tukey_q4_wer$Q4_clean)
df_q4_ser <- as.data.frame(tukey_q4_ser$Q4_clean)

df_q4_cer$Comparison <- rownames(df_q4_cer); df_q4_cer$Metric <- "cer"
df_q4_wer$Comparison <- rownames(df_q4_wer); df_q4_wer$Metric <- "wer"
df_q4_ser$Comparison <- rownames(df_q4_ser); df_q4_ser$Metric <- "ser"

tukey_q4_all <- rbind(df_q4_cer, df_q4_wer, df_q4_ser)
tukey_q4_all <- tukey_q4_all[, c("Metric", "Comparison", "diff", "lwr"
    , "upr", "p-adj")]

write.csv(tukey_q4_all, "tukey_q4_language_comparisons.csv", row.names
     = FALSE)
write.xlsx(tukey_q4_all, "tukey_q4_language_comparisons.xlsx",
    rowNames = FALSE)
```

```
# POST-HOC TUKEY TESTS for Q5
tukey_q5_cer <- TukeyHSD(aov(cer ~ Q5_clean, data = merged_df_cleaned)
    )
tukey_q5_wer <- TukeyHSD(aov(wer ~ Q5_clean, data = merged_df_cleaned)
    )
tukey_q5_ser <- TukeyHSD(aov(ser ~ Q5_clean, data = merged_df_cleaned)
    )

df_q5_cer <- as.data.frame(tukey_q5_cer$Q5)
df_q5_wer <- as.data.frame(tukey_q5_wer$Q5)
df_q5_ser <- as.data.frame(tukey_q5_ser$Q5)

df_q5_cer$Comparison <- rownames(df_q5_cer); df_q5_cer$Metric <- "cer"
df_q5_wer$Comparison <- rownames(df_q5_wer); df_q5_wer$Metric <- "wer"
df_q5_ser$Comparison <- rownames(df_q5_ser); df_q5_ser$Metric <- "ser"

tukey_q5_all <- rbind(df_q5_cer, df_q5_wer, df_q5_ser)
tukey_q5_all <- tukey_q5_all[, c("Metric", "Comparison", "diff", "lwr"
    , "upr", "p-adj")]

write.csv(tukey_q5_all, "tukey_q5_language_comparisons.csv", row.names
    = FALSE)
write.xlsx(tukey_q5_all, "tukey_q5_language_comparisons.xlsx",
    rowNames = FALSE)

# Create a new variable to make separation of model type x text type
merged_df_cleaned$model_text_type <- paste(merged_df_cleaned$model,
    merged_df_cleaned$text_type, sep = "_")

# Clean up the rows of the dataframe
merged_df_cleaned <- merged_df_cleaned[
  !is.na(merged_df_cleaned$model) &
    !is.na(merged_df_cleaned$text_type) &
    merged_df_cleaned$Q4 != "Imported(rcd3_text)" &
    merged_df_cleaned$Q4 != "What-is-your-First/Best/Most-Fluent-
        Language?", ]


# CER by First Language
ggplot(merged_df_cleaned, aes(x = Q4_clean, y = cer, color = model_
    text_type)) +
  geom_jitter(width = 0.2, alpha = 0.6) +
  labs(title = "CER-by-First-Language-(Q4)", x = "First-Language", y =
```

```
      "CER", color = "Model + Text Type") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
ggsave("scatter_cer_q4.png", width = 8, height = 5)

# WER by First Language
ggplot(merged_df_cleaned, aes(x = Q4_clean, y = wer, color = model_
    text_type)) +
  geom_jitter(width = 0.2, alpha = 0.6) +
  labs(title = "WER by First Language (Q4)", x = "First Language", y =
      "WER", color = "Model + Text Type") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
ggsave("scatter_wer_q4.png", width = 8, height = 5)

# SER by First Language
ggplot(merged_df_cleaned, aes(x = Q4_clean, y = ser, color = model_
    text_type)) +
  geom_jitter(width = 0.2, alpha = 0.6) +
  labs(title = "SER by First Language (Q4)", x = "First Language", y =
      "SER", color = "Model + Text Type") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
ggsave("scatter_ser_q4.png", width = 8, height = 5)

# CER by Second Language
ggplot(merged_df_cleaned, aes(x = Q5_clean, y = cer, color = model_
    text_type)) +
  geom_jitter(width = 0.2, alpha = 0.6) +
  labs(title = "CER by Second Language (Q5)", x = "Second Language", y
      = "CER", color = "Model + Text Type") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
ggsave("scatter_cer_q5.png", width = 8, height = 5)

# WER by Second Language
ggplot(merged_df_cleaned, aes(x = Q5_clean, y = wer, color = model_
    text_type)) +
  geom_jitter(width = 0.2, alpha = 0.6) +
  labs(title = "WER by Second Language (Q5)", x = "Second Language", y
      = "WER", color = "Model + Text Type") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
ggsave("scatter_wer_q5.png", width = 8, height = 5)
```

```
# SER by Second Language
ggplot(merged_df_cleaned, aes(x = Q5_clean, y = ser, color = model_
    text_type)) +
  geom_jitter(width = 0.2, alpha = 0.6) +
  labs(title = "SER by Second Language (Q5)", x = "Second Language", y
      = "SER", color = "Model + Text Type") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
ggsave("scatter_ser_q5.png", width = 8, height = 5)


# === Correlation: text type X Accent heaviness (Q13) ===

# Correlation calculation of accent heaviness
merged_df_cleaned$Q13_numeric <- as.numeric(recode(merged_df_cleaned$
    Q13,
                                                   "People usually do
                                                       not understand me
                                                       " = 1,
                                                   "People often do
                                                       not understand me
                                                       " = 2,
                                                   "People understand
                                                       me with some
                                                       effort" = 3,
                                                   "Most people
                                                       understand me
                                                       relatively easily
                                                       " = 4,
                                                   "My accent does not
                                                       hinder
                                                       understanding me"
                                                       = 5
))


summary(merged_df_cleaned$Q13_numeric)



cor.test(merged_df_cleaned$Q13_numeric, merged_df_cleaned$cer)
cor.test(merged_df_cleaned$Q13_numeric, merged_df_cleaned$wer)
cor.test(merged_df_cleaned$Q13_numeric, merged_df_cleaned$ser)

# Save data to CSV and plot
```

```r
cor_data <- merged_df_cleaned %>%
  select(response_ID, Q13_numeric, cer, wer, ser)

write_csv(cor_data, "q13_numeric_cor_data.csv")

plot_with_lm <- function(yvar, label) {
  ggplot(cor_data, aes(x = Q13_numeric, y = .data[[yvar]])) +
    geom_point(alpha = 0.6) +
    geom_smooth(method = "lm", se = TRUE, color = "blue") +
    labs(
      title = paste("Correlation between Accent Perception and",
        toupper(yvar)),
      x = "Accent Heaviness (Q13_numeric)",
      y = label
    ) +
    theme_minimal()
}

# Create and display plots
plot_cer <- plot_with_lm("cer", "CER")
plot_wer <- plot_with_lm("wer", "WER")
plot_ser <- plot_with_lm("ser", "SER")

print(plot_cer)
print(plot_wer)
print(plot_ser)

# Save plots
ggsave("q13_vs_cer.png", plot_cer, width = 6, height = 4)
ggsave("q13_vs_wer.png", plot_wer, width = 6, height = 4)
ggsave("q13_vs_ser.png", plot_ser, width = 6, height = 4)

write.csv(merged_df_cleaned, "merged_df_cleaned.csv", row.names =
  FALSE)
write.xlsx(merged_df_cleaned, "merged_df_cleaned.xlsx")
```