University of Sheffield

Computer Says 'Know': ASR Confidence and Transcription

Adam Spencer

Supervisor: Professor Jon Barker

A report submitted in fulfilment of the requirements
for the degree of BSc in Artificial Intelligence and Computer Science

in the

Department of Computer Science

May 24, 2023

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: Adam Spencer

Signature:

Date: May 24, 2023

Abstract

Can a computer know if someone said 'know' or 'no'?

If automatic speech recognition (ASR) systems could estimate where they were likely to have gone wrong, such a system could be used to aid a human transcriber by presenting them with the results it is less confident of and allowing them to work through the results until the errors are sparse.

This work explores the possibility of using a new ASR system by OpenAI called *Whisper* to reduce the amount of speech data which must be manually transcribed, finding a metric of confidence which can halve the number of ASR transcript errors while checking 44% fewer results than if they were unaided.

Acknowledgements

I would like to thank my family for supporting me throughout my degree (and, I suppose, my life) and my wonderful partner Isobel for showing me seemingly endless love and kindness.

I would also like to thank my supervisor, Prof. Jon Barker, for helping me develop my understanding and interest of this subject and for plenty of interesting conversations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Automatic Speech Recognition (ASR) is difficult. Some 'state-of-the-art' systems claim to produce error rates as low as 4%[1] while those same models achieve errors as high as 40% on other datasets[2]. Clearly modern ASR systems aren't quite ready to be used to transcribe speech with an expectation of highly-accurate results.

On the other hand, services which employ humans to produce speech transcripts such as *Rev*[3] charge $1.50 per minute and take around 24 hours to complete, though they claim error rates of approximately 1%. For those without access to substantial funding or time to wait, human transcripts are out of the question.

Rather than rely on either humans or ASR to produce a high quality transcript, why not leverage the speed of ASR and the accuracy of a human? If the ASR system were able to estimate its *confidence* in its output, a human transcriber would only need to correct the results which are more likely to contain errors and work their way through the computer-generated results until the frequency of errors had reduced to a level they could accept. The problem is, Whisper doesn't report its confidence in the transcript it produces.

This work shall explore the applicability of a new free and open-source ASR model called '*Whisper*'[4] to be used to aid a human in producing high-quality transcripts. By exploring ways to derive model confidence, some metrics to rank results in order of expected correctness shall be implemented, followed by a demonstration of different techniques using ASR confidence to aid a human transcriber. Finally, the ability of different confidence metrics to accurately predict transcription errors will be tested and discussed in detail.

The following chapters shall review and analyse the literature surrounding ASR and confidence, build some requirements for this project, design a system to find ASR confidence from a speech corpus and finally test the quality of this system using a rudimentary simulation.

Chapter 2

Literature Survey

The purpose of this chapter is to develop an understanding of the existing literature on the topics of automatic speech recognition, neural network confidence, speech corpora, and computer-aided transcription. Each topic is discussed in its own section, with subsections to explore parts of each topic in greater depth.

2.1 Automatic Speech Recognition

2.1.1 What is ASR?

Automatic Speech Recognition (ASR) is a term used to describe technology which allows computers to recognise and produce a text transcription of spoken language. The research and development of technology involving speech has been a part of computer science since the late 1930s[5, 6], with rudimentary ASR systems being constructed as early as the 1950s[7]. These early attempts at recognising human speech treated it as a 'pattern matching' problem, the theory being that words could be constructed by matching the pattern created in a speech signal to corresponding spoken phonemes[5]. Despite speech recognition fundamentally being a problem of matching speech patterns to text, these early attempts were not particularly robust; requiring re-tuning by a human operator in order to match the specificities of each new speaker's speech patterns[7].

The 1970s saw the application of statistical techniques to improve the robustness of ASR systems[5], the most widely adopted method being the application of 'hidden Markov models' (HMMs)[8]. Use of HMMs continued through the '90s[9] and is still in use today[10].

2.1.2 Hidden Markov Models

In order to better understand the way in which modern ASR systems have developed, why they function the way they do, and what limitations have yet to be solved, it's important to explore the approach which historically saw the widest adoption; hidden Markov models.

First, what is a 'Markov model' (also known as a Markov process)? In his 1960 work[11], Dynkin describes a Markov process using the example of a randomly-moving particle in space;

“ If the position of the particle is known at the instant t, supplementary information regarding the phenomena observed up till the instant t (and in particular, regarding the nature of the motion until t) has no effect on prognosis of the motion after the instant t (for a known ”present”, the ”future” and the ”past” are independent of each other). ”

From his description, we can draw the following assumptions for modelling a system as a Markov process;

- The system consists of states.
- The system is *in motion*, i.e. moving between states.
- This motion is random.
- The motion observed prior to t (e.g. $t-1$) does not influence the motion following $t+k$ where $k \geq 1$.
- Because the particle is constantly moving between states, the state at time t depends only on the state immediately prior, $t-1$.
- There is some probability, p, that the system moves from one state to another.

In a *Hidden* Markov Model (HMM), the states and transition probabilities between them are known, but for some output sequence the order and selection of states used to produce the output is not known. Knowing both the states and transition probabilities, it is therefore possible to calculate the most probable set of inputs used to produce the output.

To apply this model to speech, treat speech as a continuous sequence of discrete states, where each state is a feature vector representing an acoustic signal (either whole words, phonemes or even sub-phonetic features[9]). Assuming that each state is generated from a probabilistic distribution correlated with other states in the model[12] (i.e. probability that one state follows another) and having trained these distributions on known data, the output signal (i.e. the speech signal) can be used to determine the most probable sequence of tokens spoken. These tokens may then be decoded by a language model to construct a transcription[9].

Despite making up much of the research foundational to modern ASR, Markov models have a crucial flaw when applied to speech; parts of speech are dependent on more than just the part immediately before (i.e. $t-1$). For instance, in a presentation discussing *hats* it is unlikely that the word *cat* would be used, despite the phonetics of the word being largely the same.

2.1.3 The Transformer

Skipping ahead from the mid-1980s to the current day 'state-of-the-art', ASR has moved towards what is known as the 'encoder-decoder' model. At a high level, this model consists of two key parts; an encoder and a decoder. The encoder processes (*encodes*) input audio into features, these features are aligned with language and then processed by the decoder (*decoded*)

to produce an output transcript[13]. The key difference between modern approaches and the classical HMM-based approach is the use of widely researched 'machine learning' techniques, including various forms of neural network[14, 15, 16, 17].

Recent research has proposed a new network architecture called the *Transformer*[18], aiming to reduce the computational complexity of encoder-decoder models by forgoing convolutional or recurrent neural networks (CNNs and RNNs) and instead relying on 'attention'. The motivation for the *Transformer* can be understood as follows;

- CNNs (e.g. [19]) and RNNs (e.g. [20]), while popular, have greater per-layer computational complexity than self-attention[18].
- Recurrent neural networks must perform $O(n)$ sequential operations for a sequence length n, whereas self-attention has a constant (i.e. $O(1)$) maximum number of sequential operations, enabling parallel computation[18].
- By allowing each layer in the encoder and decoder to attend to the whole output of the previous layer,

In a multilayer network, attention layers are used to build relations between separate parts of an input sequence by allowing each node (or 'attention head'[21]) to attend to all outputs from the previous layer. A technique referred to as 'multi-head attention'[18] enables attention to be calculated in parallel for all inputs in a sequence.

The calculation for attention is:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

where;

- Q is known as the *Query* vector,
- K is known as the *Key* vector, and
- V is known as the *Value* vector.

The output of the attention function is described as "a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key"[18]. To understand what these vectors denote in the context of speech recognition, we must understand what the different types of attention *are* and how their use *differs* in the encoder and decoder;

- self-attention layers (in both the encoder and decoder) take input from the previous layer within the same block (i.e. encoder or decoder). In this layer, Q, K, and V are all calculated by multiplying an input embedding (a vector representation of the raw input sequence) by three different trained weight vectors.

- cross-attention layers (present only in the decoder) enable mixing of the encoder's output with the output of the decoder's previous self-attention block, allowing the decoder to attend over the whole encoded input. In this layer, only K and V are taken from the encoder; Q is derived from the previous decoder block.

For the sake of clarity, a simplified version of the operation of the transformer (as explained in [18]) is as follows;

1. An input sequence is transformed into an embedding, which is then input into the *encoder stack* (the layers which make up the encoder) after being modified using *positional embeddings*, which act to preserve information about the position of each part of the input sequence.

2. The *attention head* of each self-attention layer in the encoder first multiplies the input by three trained weight matrices to acquire the values of Q, K, and V, then calculates Attention using these values as input. Each attention head is trained with its own weight matrix, so there are an equal number of Attention results and attention heads. The resulting attention matrices are concatenated and then multiplied by another weight matrix to produce the layer's output to be fed forward to the next layer.

3. The output of the final encoder layer is transformed into a pair of K and V vectors and used as input (via *cross-attention* for the decoder, which works almost identically to the encoder (i.e. a stack of self-attention blocks). The key difference between the encoder and decoder is that the decoder uses previous decoded output as its Q vector and it may not self-attend to all of the output at once, only the output earlier to its current position (unlike the encoder which attends to the whole sequence simultaneously)

4. Finally, the output of the decoder goes through some normalisation steps to find and output the most probable *output embedding*.

An *output embedding* may be thought of as a way to link together an expected output (e.g. a word) and a vector in one-hot encoding space. In the Transformer's decoder this allows a score represented in a one-hot encoding to be matched to an output.

Though this is all rather complex, the key points to understand are that a Transformer uses both an encoder and decoder, the decoder can attend to only previously decoded output, and these two 'blocks' work together to produce an output.

2.1.4 Evaluating ASR Systems

As with any computational model, the performance of an ASR system is evaluated in terms of both speed and accuracy. Calculating the speed of a model is simple enough; just time it! Accuracy, however, is not so simple because there are multiple metrics upon which a reference and ASR-generated transcript (known as a 'hypothesis') may differ.

There are three different levels at which to calculate the similarity between a reference and hypothesis; the word-level, phoneme-level, and character-level[22] Comparing words and characters is relatively self-explanatory, the transcripts are either split up into individual words or characters for comparison. A phoneme-level comparison involves transforming textual words into the parts of speech which constitute them, known as phonemes.

The accuracy calculation to derive an 'error-rate' (also known as the edit-distance[23]) is as follows;

$$WER = \frac{S + D + I}{S + D + C}$$

where:

- S is the number of *substituted* words (words which appear in the place of another correct word)
- D is the number of *deleted* words (words which aren't present in the hypothesis but appear in the reference)
- I is the number of *inserted* words (words which do not appear in the reference but appear in the hypothesis)
- C is the number of *correct* words (words which appear in both the reference and hypothesis)

Word error-rate (WER) is the most commonly used metric in the literature[24], though arguments can be made that it is not entirely representative of the degree to which a system has an understanding of speech. Take, for example, a reference which contains a compound word like 'soundproof' or 'eggshell' – if an ASR system were to output 'sound proof' or 'egg shell' it would have incurred one substitution and one deletion equating to a WER of 200% despite having produced an output which is representative of the input.

Judging by calls to work to produce a metric to replace WER still being made today[25, 2] and the relative lack of published ASR papers discussing results using metrics other than WER, there is not yet a viable alternative.

2.1.5 Problems in ASR

Despite their ubiquity, modern ASR systems aren't without fault.

Cutting edge systems like *(*wav2vec) are touted as being capable of achieving 'greater-than-human' scores on specific datasets[1, 26, 27] such as *LibriSpeech*[28], achieving as low as 1.4% error[29]. LibriSpeech consists entirely of English audiobook recordings which have been selected in-part based on their quality[28]; not particularly representative of everyday speech[2], lacking features such as speaker overlap which are common in conversation[30]. An evaluation of modern proclaimed 'state-of-the-art' ASR systems found WER scores averaging approximately

17% when faced with real-world conversational data[2] despite reporting results below 4% on LibriSpeech.

Racial disparities in the accuracy of ASR systems has also been reported, with average WER scores for black speakers almost double that of white speakers[31]. The authors of this article attribute this performance difference to the acoustic models used in the ASR systems rather than the language models, thus concluding that there is a lack of training data from black speakers.

These two problems serve indicate that, in order to improve accuracy, ASR systems should be trained on more diverse data in terms of the context of the speech and the demographic of speakers.

2.2 Whisper

In late September 2022, the OpenAI research laboratory (known for such projects as GPT-3/4 and ChatGPT) released a new open-source ASR system called 'Whisper' [4] which uses the encoder-decoder Transformer architecture discussed in section 2.1.3. It is described as a 'zero-shot' model, meaning it is expected to produce good results *without* any dataset-specific fine tuning. Whisper is unique in being very large (trained on 680,000 hours of speech data), open-source, and fully supervised; all the training data used to create the model has been accurately labeled and quality-checked by humans, unlike much larger unsupervised (or semi-supervised) models such as 'BigSSL' (1,000,000+ hours of data) [26].

Unsupervised training is appealing for training speech recognisers because there is a wealth of unlabeled recordings, and labeled recordings are uncommon for less widely-spoken languages[32]. Unsupervised systems have a clear disadvantage, however, when compared to supervised; they lack clear decoder mappings[4], meaning that even for a successfully encoded input there may not be a clear mapping from that input into a speech token. To solve this, fine-tuning to map encodings to decoded text tokens is done on the part of the model's developers, though this is a precarious route to overfitting; if the model is too fine-tuned to its training data, performance will suffer when faced with data which isn't well represented in the training set. For example, if an unsupervised model were trained using the voices of young people, it may perform with considerably poorer accuracy when used to transcribe elderly speakers due to differences inherent to their speech[33].

The way Whisper functions is discussed in detail in section 4.3.

2.3 Confidence

Neural networks will always produce an output for an accepted input, no matter how likely or unlikely the output is of being correct. The network may even assign a very large probability to its given output; consider, for example, a classifier trained to predict the city in the UK which someone was born in given the co-ordinates of their current address. The classifier simply assigns the closest city to their address and often makes a correct prediction. However,

if the input co-ordinates were somewhere in Iceland it may assign a high probability to their place of birth being in Inverness because it is much closer than any of the other UK cities, even though the true probability that this hypothetical Icelander was born in Inverness is quite low.

The point of this example is to illustrate that the probability assigned by a neural network is not always a good indication of correctness. An estimation of a model's expected correctness is referred to as *confidence*, i.e. how *confident* is the model that its prediction is correct?

Chapter 3

Requirements and Analysis

The objective of this work is not to produce a fully-working, infallible system which aims to receive actual use by transcribers, rather, the aim of this work is to explore the current state of the field of ASR and to understand the extent that current ASR technology could provide aid to a human transcriber. With the purpose of facilitating an extensive evaluation, this chapter shall list the requirements for this work to meet its objective and provide a detailed analysis of each requirement.

3.1 Generate transcripts using ASR

Evaluating the quality of ASR transcription requires a key set of data; ASR-generated transcripts. Rather than comparing different ASR systems, Whisper[4] has been chosen as the only system to use for generating transcripts because;

- it is new (made available in September 2022);
- it is entirely free and open-source, meaning it is easily modifiable and available to be used without licence; and
- it reportedly achieves very good results across different speech corpora.

As mentioned in the literature review, Whisper is implemented using *PyTorch*, meaning this work would benefit greatly from access to high-performance GPUs. This would enable fast turnaround times when transcribing large speech corpora and thus enable rapid evaluation and tweaking of settings to minimise erroneous results.

The key to generating useful transcripts is some high-quality speech recordings from a speech corpus. While preliminary testing of Whisper may use data from any available corpora, it would be very useful to obtain some data which is;

- not present in Whisper's training data, to prevent the model from regurgitating labels for data it has already seen;

- is well-suited to Whisper's particularities, aiming to maximise the usefulness of results; and
- is representative of real data which would benefit from computer-aided transcription, as to enable more practical evaluation.

Two preliminary aims, therefore, are to understand what kind of data is suited to Whisper and then what kind of data would benefit from computer-aided transcription. Once these aims are understood, a suitable dataset may be gathered and used for evaluation, however it is also useful to understand the caveats related to using well-suited data! The naïve assumption that all data seen by *any* computer system is 'perfect' would misrepresent the usefulness of the system in question. To combat this, this work must properly acknowledge the limited extent to which a computer-aided transcription system using Whisper is viable, and evaluate how the viability could be increased to be more applicable to real-world tasks.

3.2 Implement various confidence measures

Neural network confidence is widely discussed in the literature. Considering the aim of this work, it shall focus on re-creating and applying existing measures of confidence to Whisper, whether through modification to the model itself or through inference of the model's output.

If some such modifications fall out of scope of the project, this work would still benefit from a discussion of how those approaches may be applied to a future system and what advantages they may bring.

3.3 Understand the effectiveness of selected confidence measures

Evaluation of the extent to which Whisper may aid human transcription may be done by comparing the accuracy of Whisper's predictions against a reference and the reported model confidence.

This type of evaluation requires taking the following steps to complete;

1. Selection of suitable measure(s) of system accuracy
2. Extensive normalisation of text to facilitate accurate measurements
3. Visualisation of results

The standard across surveyed literature is *word error rate* (WER), despite potential limitations. For the sake of evaluation, other measurements such as *phone error rate* (PER) and *character error rate* (CER) should be calculated alongside WER.

3.4 Explore designs for computer-aided transcription

It would be of great utility to understand how system confidence may aid a human transcriber as this would further refine the 'lens' through which the system may be evaluated, and as such is vital to the completion of this work. There's limited use in a purely theoretical exploration of a computer system such as this, which is designed to be interfaced with by a human. Instead, demonstrating the benefits of a computer-aided transcription system would be easily facilitated using a graphical program.

A number of considerations are required for the design of this system specifically due to its intended nature to serve as an example rather than a final implementation, including;

- several design iterations should be produced to demonstrate different extents of computer aid;
- each design decision should be discussed thoroughly to demonstrate the intended effect and any notable caveats; and
- a suitable number of screenshots are required in the appendix to demonstrate use of the system.

Chapter 4

Design

This chapter shall explain the design choices made while completing the project including the methods for calculating confidence, the format used to store transcription data, and the overall design of a demo system for computer-aided transcription.

4.1 Speech Data

According to its authors, Whisper's robustness is due likely in part to its use of a language model in its decoder[4]. Though likely beneficial for keeping track of sentence context, this poses a potential threat to the models accuracy in a number of circumstances, including;

- Misspoken words or sentences with improper syntax (e.g. 'then' instead of 'than'), for these errors may be corrected by the model, despite being inaccurate to the original recording.
- Disjoint terms (e.g. 'book purple dish soap'), as such terms are highly unlikely to occur in sequence and thus the language model will not consider them a probable output.

To combat these drawbacks, this work will use a conversational speech corpus rather than one made of spoken disjoint terms. While not representative of all speech, an argument can be made that the majority of speech which must be transcribed (e.g. conference recordings, courtroom hearings, lectures, etc.) has a maintained context throughout and is not significantly formed of disjoint terms, though the potential for disjoint terms to be present in a recording should be acknowledged as a potential area of weakness for Whisper.

While introducing the corpus selected for this work that the original objective was to explore the impact of age-related changes to speech on ASR performance, and for this goal the *LifeLUCID* corpus[34] was determined to be the best match. Despite the scope of the project having since changed, the data gathered fits the new objective very well, as it is formed of 52 recordings of conversations between 104 discrete speakers aged between 8 and 85 years old. They are solving a 'spot-the-difference' task, and the data selected for this work

was recorded in normal conditions (that is, they can hear and communicate with each other normally).

The corpus' authors mention that the reference transcripts were generated by an ASR system and only one channel's audio was human-corrected. The ability to compare the quality of ASR output to a reference transcript is required to evaluate this work, thus, only the human-corrected transcript and corresponding audio channel were used to ensure the references are reliable. This leaves 52 10-minute recordings of individual speakers with gaps where the other participant is speaking.

4.1.1 TextGrid Format

The reference transcripts are supplied in *Praat TextGrid* format which is produced by the Praat software suite[35]. The TextGrid format consists of each individual part of speech (words, hesitations, mid-sentence silences, silences when the other participant is speaking etc.) being present in consecutive entries with the time in the recording which they start and finish.

```
intervals [14]:
  xmin = 21.05
  xmax = 21.47
  text = "BUSH"
```

Figure 4.1: Example of an entry in TextGrid format

Figure 4.1 is an example of a single 'interval' in the TextGrid format. A larger example is available in Appendix 7 You may observe that `xmin` and `xmax` denote the points in the recording at which the section starts and ends, and `text` denotes the content of the section. Considering that each entry appears consecutively and that there are over 1,000 in each file, the format is not easily human-readable.

4.1.2 Data Preparation

There are a number of issues with using the data in its original format, including;

1. Whisper struggles to maintain alignment when transcribing long form data[4], so the approximate 10-minute length of each recording requires shortening to maintain system performance;
2. TextGrids are not human-readable; and
3. The output of the ASR system should be stored alongside the reference transcripts to ease evaluation, which is not possible using TextGrids.

The solution to the first problem would best be solved by splitting the conversation recordings into individual utterances. Luckily, the human-evaluated references include metadata

which shows the start- and stop-times of each word and non-word part of speech, meaning it can easily be split into individual utterances without using voice activation detection or other automatic techniques. Using the documentation for LifeLUCID it was possible to determine that there are two types of non-speech token;

1. 'Break' tokens – these are tokens which denote the speaker is not mid-utterance; either listening to the other participant or engaged in irrelevant discussion (these latter parts are silenced in the recordings).

2. 'Junk' tokens – these denote either:

    - The speaker has paused (but the other participant is not speaking).
    - A bell or dog bark is being played as part of their task (these are silent in the recording).
    - Hesitations (e.g., 'umm', 'uhhh', etc.)
    - Other non-speech, non-breaking tokens (not specified in their documentation but present in the transcripts).

For the purpose of this work, an utterance is defined as an uninterrupted piece of speech without any long pauses. By providing threshold values for the minimum length of a 'break' token and maximum length of a 'junk' token, the boundaries at which utterances start and end can be easily computed from the reference TextGrids. The utterance boundaries can then be used to extract individual utterances from the full-length recordings, resulting in a series of numbered audio files.

The second and third problems can be solved together by changing from the TextGrid format to JSON (JavaScript Object Notation). This format is human-readable[36] and able to hold all the data and metadata required for this work, including a way to reference the original piece of audio it represents.

4.2 Running Whisper

Whisper comes packaged with models of various sizes, requiring between approximately 1 and 10GB of VRAM and an increasing amount of time to produce transcripts. Considering the objective to create an automatic transcription system which uses entirely free and open-source software, it is worth assuming that the individuals or institutions who would benefit the most from this system are those without the resources to rely on professional manual transcription. It follows, then, that these 'target users' would not have access to high-powered computers and instead rely on consumer-grade hardware to generate transcriptions. Thus, for the purposes of this work, the medium English-language model was selected as it requires only 5GB of VRAM and takes approximately half as much time to produce transcripts as the larger models[4].

4.3 Confidence Scoring

Whisper does not have a clear confidence scoring system in its unaltered state. This is a caveat for calculating confidence resulting from the design philosophy of Whisper's authors; it is a one-shot model so it is not expected to be fine-tuned to a dataset before use. Without fine-tuning or prior exposure to the type of data present in the input, Whisper can't be trained to predict the correctness of its output.

Therefore, for it to be viably used to aid a human transcriber, some method to estimate system confidence must be implemented based on the way it converges on and scores an output. At a high level, there are two sources from which to estimate confidence: Whisper's standard output, and its internal processes.

Table 4.1 gives a brief comparison of the benefits and drawbacks associated with each of these sources of confidence;

Score Source	Benefits	Drawbacks
Standard Model Output	Does not require any modification to Whisper.	Only shows an average probability score per utterance.
Model Internal Scoring	Allows access to per-word scoring and the steps the model takes to converge on an output.	Requires modification to Whisper.

Table 4.1: Comparison between sources of model confidence

The following subsections provide a detailed explanation of how each approach works and should help illustrate the benefits and drawbacks of each.

4.3.1 Confidence From Model Output

Though it does not yield a clear confidence score, Whisper does output various data relating to its processing of the input. These include;

- `avg_logprob` – The average of the log token-probability for a segment of speech (discussed further below)
- `compression_ratio` – The ratio of the length of the UTF-8-encoded text to its gzip-compressed representation. Due to the way gzip operates, this ratio indicates the 'repetitiveness' of the decoded text; a higher ratio means that the result is more repetitive, suggesting that there may have been a decoding error.
- `temperature` – Before producing an output, if the `avg_logprob` is below a certain threshold or the `compression_ratio` is above some threshold, the model will treat the decoding as failed and compute a new output with an increased temperature parameter. Temperature is used to introduce some randomness while computing predictions, therefore

the higher the final output `temperature` value is, the more randomness had to be introduced in order to determine the given output.

- `no_speech_prob` – Whisper is trained to complete many tasks, one of which being the detection of non-speaking moments in a recording. This value indicates the model's predicted probability that there is no speech in the input audio file.

Of these metrics, this work will focus on the `avg_logprob`, which is the average of the log probabilities for each token in a segment of the input, which are computed as the log softmax of the *logits*, which can be thought of as unnormalised 'scores' that have been assigned to each token while decoding. The softmax function is used to normalise these scores such that their sum is equal to 1 (allowing them to be used as probabilities).

The logits themselves are calculated during each forward pass through the decoder, derived from both learned positional and token embeddings. In the context of the Transformer's decoder, learned positional embeddings can be thought of as a way to learn information about token positions in the transcript[37]. By using token- and learned position-level weights with cross-attention from the encoder[18], Whisper acts like an "audio-conditional language model"[4]; the scores (logits) assigned to tokens are based on both the encoded audio input and a model of language.

4.3.2 Confidence From Model Internals

By default, the only information Whisper makes available about the probability scoring it used to decide on some output is `avg_logprob` despite having internally computed scores for every token in the output.

The 'tokens' that have been referred to in this section are computed using the GPT-2 tokeniser from the *tiktoken* library[38]. Words may be represented by one or more tokens called 'subword tokens' which are decoded by the tokeniser; the scores for each of these are different so a per-word score may be computed as the average score for each subword token in a word.

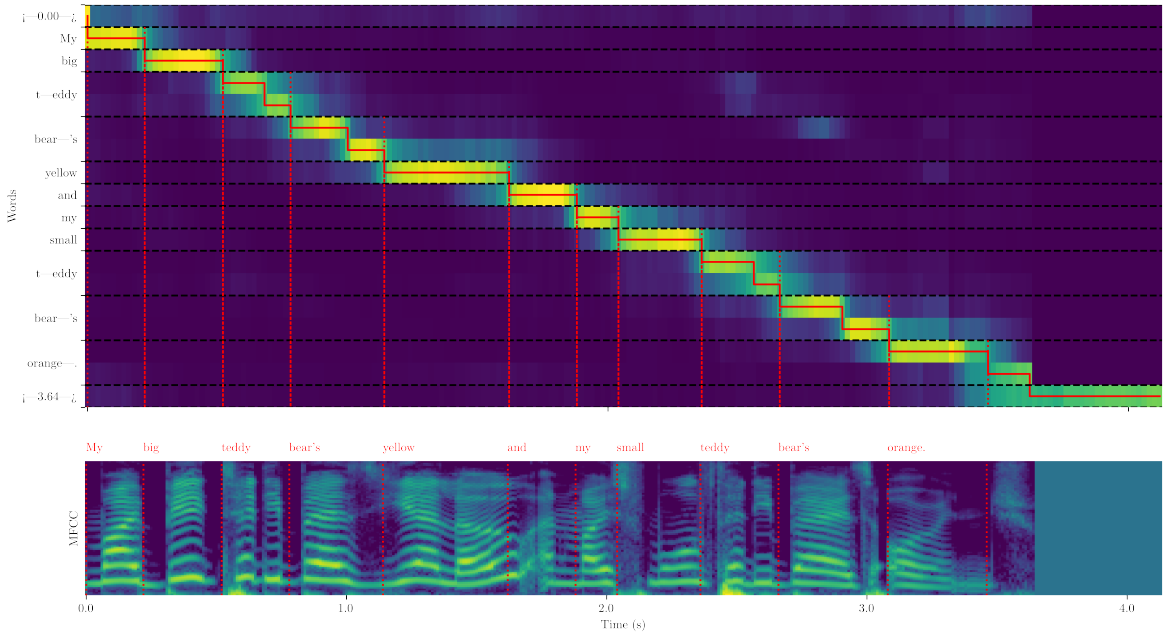


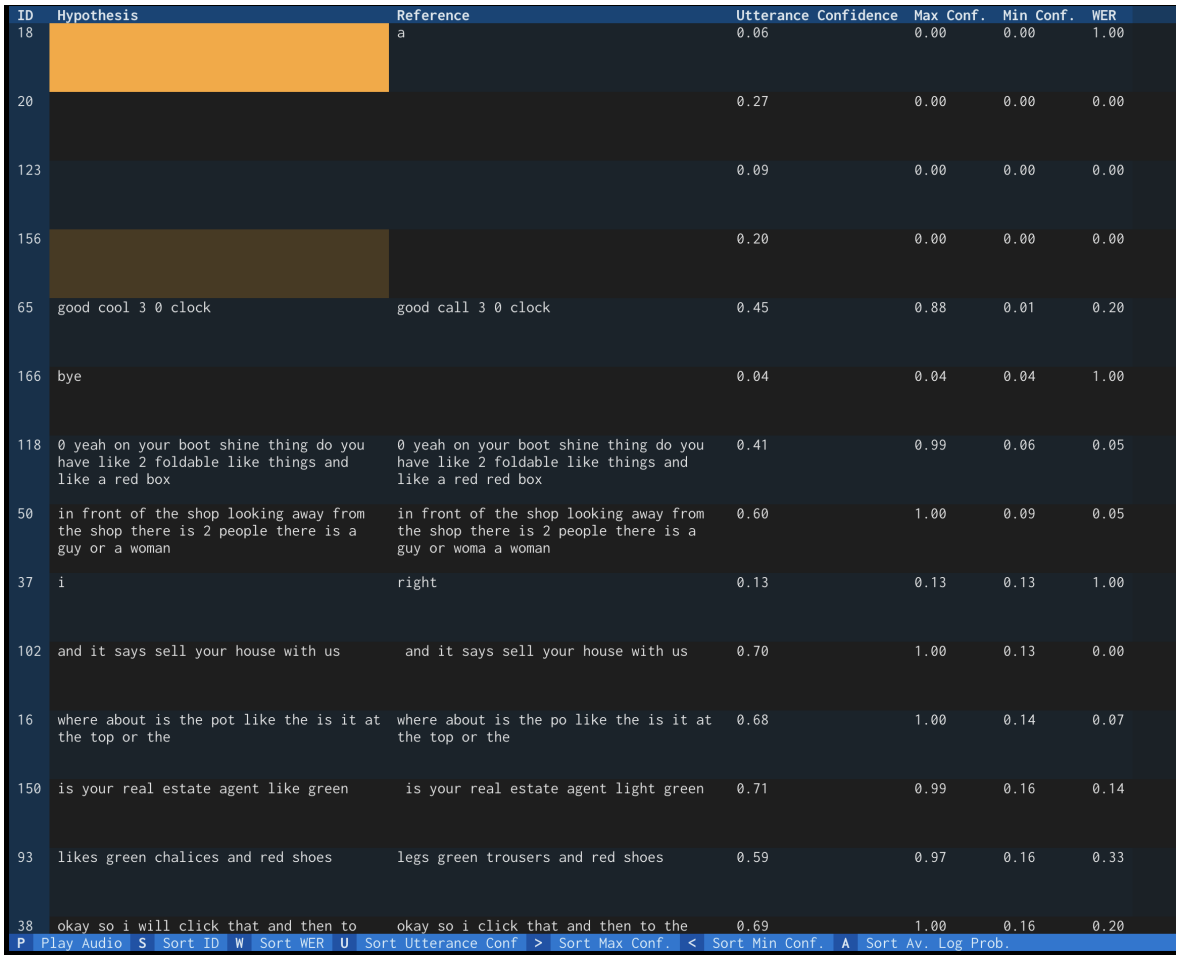
Figure 4.2: Word alignments example

Figure 4.2 was generated using the *whisper-timestamped* library[39, 40, 4]. The top plot shows "the transformation of cross-attention weights used for the [word] alignment"[39]. Notice that the words 'teddy' and 'bears' are split into subwords.

The aforementioned *whisper-timestamped* library is able to assign confidence scores to each word, computed as the exponential of the mean of the log probabilities of each subword token in a given word. Though based on probability scores which may not be wholly representative of confidence (as discussed in section 2.3), operating on individual words rather than whole utterances should provide more detail of the per-word scores than `avg_logprob` does.

4.4 A System For Transcription

As discussed in the requirements chapter (section 3.4), this work would benefit greatly from a piece of software which serves to demonstrate computer-aided transcription. At a basic level, the software should enable a user to sort utterances in a conversation by some estimation of confidence to allow them to manually correct the ASR-generated transcripts in non-descending order of correctness.

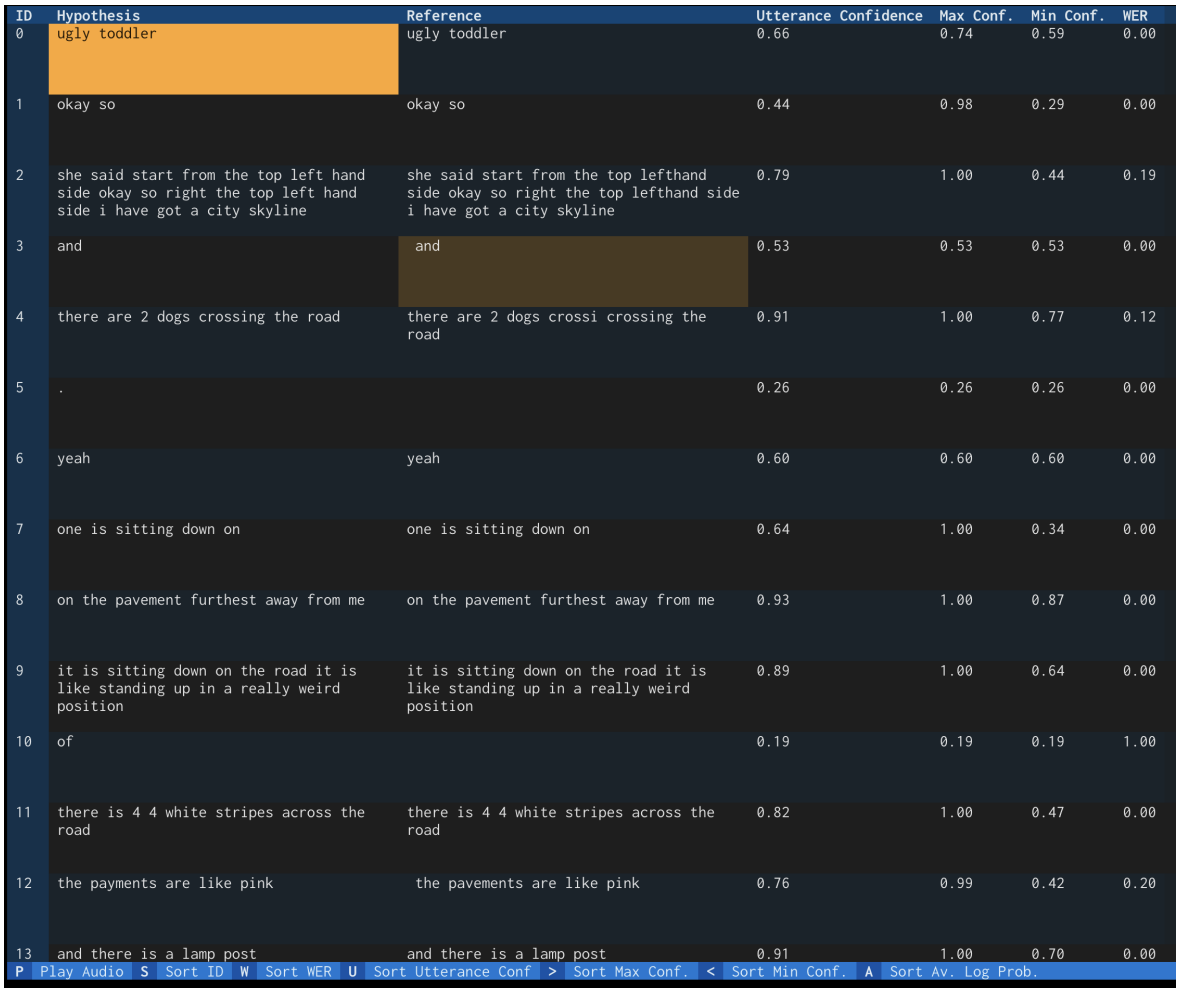


ID	Hypothesis	Reference	Utterance Confidence	Max Conf.	Min Conf.	WER
18		a	0.06	0.00	0.00	1.00
20			0.27	0.00	0.00	0.00
123			0.09	0.00	0.00	0.00
156			0.20	0.00	0.00	0.00
65	good cool 3 0 clock	good call 3 0 clock	0.45	0.88	0.01	0.20
166	bye		0.04	0.04	0.04	1.00
118	0 yeah on your boot shine thing do you have like 2 foldable like things and like a red box	0 yeah on your boot shine thing do you have like 2 foldable like things and like a red red box	0.41	0.99	0.06	0.05
50	in front of the shop looking away from the shop there is 2 people there is a guy or a woman	in front of the shop looking away from the shop there is 2 people there is a guy or woma a woman	0.60	1.00	0.09	0.05
37	i	right	0.13	0.13	0.13	1.00
102	and it says sell your house with us	and it says sell your house with us	0.70	1.00	0.13	0.00
16	where about is the pot like the is it at the top or the	where about is the po like the is it at the top or the	0.68	1.00	0.14	0.07
150	is your real estate agent like green	is your real estate agent light green	0.71	0.99	0.16	0.14
93	likes green chalices and red shoes	legs green trousers and red shoes	0.59	0.97	0.16	0.33
38	okay so i will click that and then to	okay so i click that and then to the	0.69	1.00	0.16	0.20

P Play Audio S Sort ID W Sort WER U Sort Utterance Conf > Sort Max Conf. < Sort Min Conf. A Sort Av. Log Prob.

Figure 4.3: Utterances in confidence order.

If there is any question about the context of a section of speech (e.g. there are unclear or parts of speech), the user may return to chronological order and assess the surrounding utterances for information.



ID	Hypothesis	Reference	Utterance Confidence	Max Conf.	Min Conf.	WER
0	ugly toddler	ugly toddler	0.66	0.74	0.59	0.00
1	okay so	okay so	0.44	0.98	0.29	0.00
2	she said start from the top left hand side okay so right the top left hand side i have got a city skyline	she said start from the top lefthand side okay so right the top lefthand side i have got a city skyline	0.79	1.00	0.44	0.19
3	and	and	0.53	0.53	0.53	0.00
4	there are 2 dogs crossing the road	there are 2 dogs crossi crossing the road	0.91	1.00	0.77	0.12
5	.		0.26	0.26	0.26	0.00
6	yeah	yeah	0.60	0.60	0.60	0.00
7	one is sitting down on	one is sitting down on	0.64	1.00	0.34	0.00
8	on the pavement furthest away from me	on the pavement furthest away from me	0.93	1.00	0.87	0.00
9	it is sitting down on the road it is like standing up in a really weird position	it is sitting down on the road it is like standing up in a really weird position	0.89	1.00	0.64	0.00
10	of		0.19	0.19	0.19	1.00
11	there is 4 4 white stripes across the road	there is 4 4 white stripes across the road	0.82	1.00	0.47	0.00
12	the payments are like pink	the pavements are like pink	0.76	0.99	0.42	0.20
13	and there is a lamp post	and there is a lamp post	0.91	1.00	0.70	0.00

P Play Audio S Sort ID W Sort WER U Sort Utterance Conf > Sort Max Conf. < Sort Min Conf. A Sort Av. Log Prob.

Figure 4.4: Utterances in chronological order.

A user may be influenced by the ASR transcript if they read it while listening to the accompanying audio. To solve this, words may be 'blanked out' depending on their assigned confidence score, those below a threshold parameter are replaced by blank sections so the user can't be conditioned by what the ASR predicted the output to be.

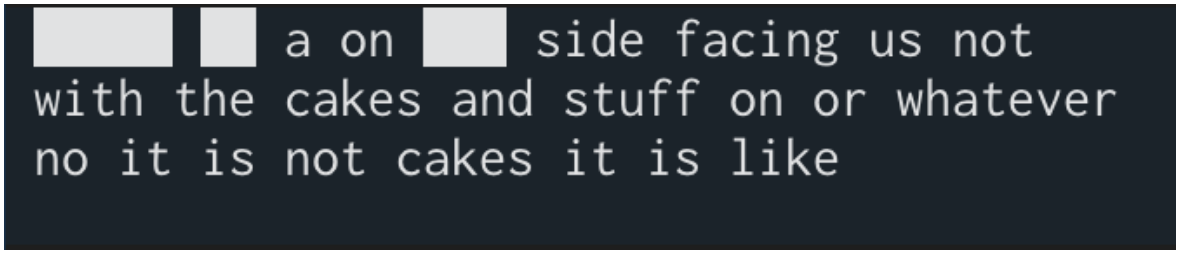


Figure 4.5: Example of a 'blanked out' utterance.

Displaying word-level confidence scores for every word in a sentence is not an intuitive way to interact with a computer system; instead, by colouring the words based on their assigned confidence score, the user may provide corrections to words which are considered less likely to be correct than others.

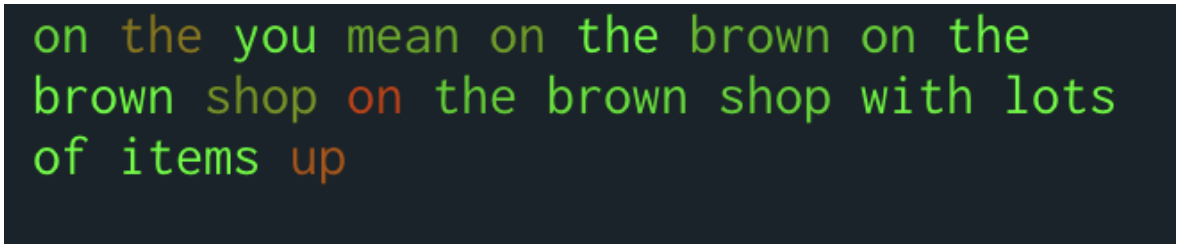


Figure 4.6: Example of an utterance with confidence-based highlighting.

All of these options may be selected using a simple menu.

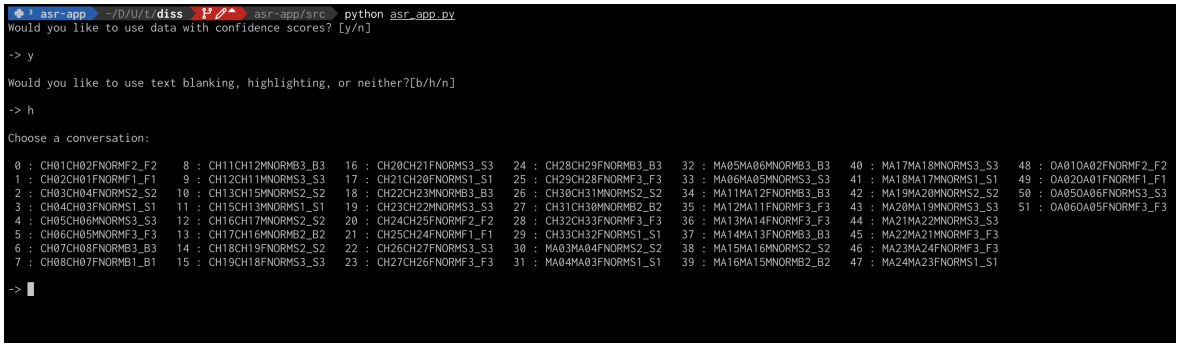


Figure 4.7: Menu presented to a user before starting the application.

This system is not capable of taking input from a user, rather it exists with the purpose of providing an example to build a more fully-featured computer-aided transcription from. It is run in a terminal window using Python with the following packages;

- *Textual*[41] to build the application front-end,
- *pysoundfile*[42] and *python-sounddevice*[43] to load and play audio clips, and
- *Rich*[44] to enable text highlighting in the terminal.

Chapter 5

Implementation

The aim of this chapter is to provide some more insight into the choices made while implementing the design as discussed in the previous chapter and explain the motivations that lead to those choices.

5.1 Data Processing

TextGrid data was broken into 'utterances' using a script called `get_utterances` (available in appendix A). This script takes as input a directory containing TextGrid data, an output directory and threshold values for the minimum break length and maximum pause length allowed in an utterance. The output is in JSON format, generating a file to which other data is added.

The second step is to use the script `segment_audio` to split up audio files into individual utterances within directories given the title of the original conversation they were extracted from. This code is available in appendix B. Upon completion, the output of this script is a directory with subdirectories containing hundreds of audio files.

Thirdly, the script `do_whisper_confidence` was run (available in appendix C) using the University of Sheffield's Bessemer HPC cluster[45]. GPU clusters on Bessemer have access to Nvidia V100 GPUs with 32GB of VRAM, enabling transcriptions to be generated with Whisper very quickly. Instead of using the standard Whisper model, the *whisper-timestamped*[39] library is used because it can calculate word-level confidence scores.

The fourth and fifth steps to process data were to run the scripts `normalise_text` and `calculate_wer` (available in appendices D & E respectively). The first of these scripts runs a text normaliser which is packaged with Whisper[4] to perform English text normalisation, then the second script calculates WER scores and finds the per-word confidence scores for each word in the output. WER is calculated using the *jiwer*[46] library.

```
"0": {
  "start": 15.04,
  "end": 15.74,
  "transcript": "okay",
  "whisper": {
    "text": "okay",
    "segments": [
      {
        "id": 0,
        "seek": 0,
        "start": 0.0,
        "end": 0.4,
        "text": " OK.",
        "tokens": [
          50363,
          7477,
          13,
          50440
        ],
        "temperature": 0.0,
        "avg_logprob": -0.7832436561584473,
        "compression_ratio": 0.2727272727272727,
        "no_speech_prob": 0.02687731385231018,
        "confidence": 0.317,
        "words": [
          {
            "text": "OK.",
            "start": 0.0,
            "end": 0.4,
            "confidence": 0.317
          }
        ]
      }
    ],
    "language": "en"
  },
  "confidence_scoring": {
    "words": "okay",
    "confidence_scores": [
      0.317
    ],
    "utterance_confidence": 0.317
  },
  "wer": 0.0,
  "avg_logprob": -0.7832436561584473,
  "file_measures": {
    "wer": 0.11153358681875793
  }
},
```


Figure 5.1: Example JSON entry from output

An example entry from the JSON output is presented in figure 5.1. The decision was made to preserve all of the output data from Whisper, meaning each entry is quite long. An entry can be identified based on its filename and the number which indexes the entry ("0" in this case), where the filename of the accompanying audio segment has the same identifier (`000.wav` in this case). As this example is the first entry in an output file, it contains a field called `file_measures` which tracks the overall WER for the entire file.

This does, however, lead to the issue of readability re-emerging. One of the goals of switching to JSON from TextGrid format was to improve the readability of the raw data. Despite being kept in neat utterances, the data is not easily read as it is largely metadata rather than just the text. Changing this would require removing some of the metadata (e.g. removing this 'words' field from the 'whisper' field), though the need for data preservation was ultimately considered paramount and thus the data remained in this format.

5.1.1 Audio Issue

During the first few experiments processing data through the script pipeline it was discovered that some audio files were being transcribed very poorly. It was assumed that the same channels were labeled 'A' and 'B' for all of the recordings, however around half of the recordings use the opposite channel to the other. This is important because the reference transcript TextGrid files are all labeled 'A' and 'B', thus the wrong audio channel had been segmented.

This issue was quickly diagnosed by calculating file-averages of the `no_speech_prob` output by Whisper, where it was discovered that half of the recordings were mostly silent (i.e. the speaker was listening to the other and not talking). The conversations that had used the wrong channel were all discovered and the audio was re-segmented using the other channel. The final corpus used for evaluation did not contain any of these errors.

5.2 Demo Software

The demo software's front-end is built using *Textual*[41], a Python library for terminal-based graphical applications. A terminal-based system was decided on to ensure compatibility across operating systems while using Python, as other techniques like *PyQt*[47] do not display the same on all OSs. Rich comes with various 'widgets' which enable quick and simple building of terminal applications, including the `DataTable` which was used for the demo software. The code for the front-end is available in appendix F.

To gather results, highlight and blank text, play audio and order results, a back-end was developed consisting of a simple class for each row and a class to represent all of the data in the table. The back-end code is available in appendix G.

5.2.1 Caveats

The demo software lacks the ability to be used as a real piece of computer-aided transcription software; it can't take user input and has occasional audio glitches.

These problems prevent it from being used in any setting outside of simple demonstration and would require modification to both the front-end design and the method for playing audio. Audio glitches are relatively uncommon and dissipate once the user plays a piece of audio multiple times, though if the system is intended to be used accurately the audio should be reliable and incapable of confusing a listener. User input could be taken by, for example, adding a new column which takes keyboard input or allowing users to overwrite 'blanked out' sections of text.

Chapter 6

Results and Discussion

6.1 Simulating Computer-Aided Transcription

An effective computer-aided transcription system must reduce the human cost required to lower the amount of errors a transcription to an acceptable level. By ordering transcribed utterances using some confidence metric and letting a human transcriber make corrections, an effective system should see the error rate fall more rapidly than if the results were corrected in a random order.

A simple simulation was devised in order to test the effectiveness of each potential measure of system confidence by using the formula for WER as follows;

1. For a set of M utterances $U = \{u_0, u_1, \dots u_M\}$ (either a single conversation or entire corpus), the substitutions (S_i), deletions (D_i), insertions (I_i), and number of words in the reference (N_i) are calculated for each utterance, u_i.

2. U is ordered using a confidence metric.

3. The WER, w_i, is calculated for a slice of U containing all items including and following some utterance, u_i;
$$w_i = \frac{\sum_{j=i}^{M}(S_j + D_j + I_j)}{\sum_{k=0}^{M} N_k}$$

4. Increment i by 1 then repeat the previous step for all $0 \leq i \leq M$. Notice that the denominator is the same for all slices but the numerator changes to simulate each utterance being corrected in order.

5. Output is a set $W = \{w_0, w_1, \dots w_M\}$, where some item w_i is equal to the WER of U after correcting all utterances prior to u_i.

These results shall be analysed by plotting graphs of WER against the percentage of utterance which have been human-corrected, henceforth referred to as 'Cost'.

6.2 Simulation Results

This section details the results acquired from running this simple simulation using results ordered with each of the following metrics;

- `avg_logprob` taken directly from Whisper's standard output;
- utterance-average confidence score; and
- utterance-minimum and -maximum word-confidence scores.

Where "utterance-minimum and -maximum word-confidence" refers to an ordering of utterances based on each utterances minimum and maximum word-level confidence score.

Results which show a per-conversation average have a shaded section to show the standard deviation from the mean, where the mean is the coloured line on the graph.

Dashed 'random order' lines show the WER/Cost trend expected if the results were manually corrected in a random order. A performant metric for ordering results should have a plot showing a line which dips below the 'random order' line. A metric which follows the trend of the 'random order' line is thus performing the same as if the results were randomly ordered and therefore ineffective for computer-aided transcription.

6.2.1 Utterance-average confidence versus `avg_logprob`

Figure 6.1: Per-conversation average WER when evaluating in non-descending order of utterance `avg_logprob`

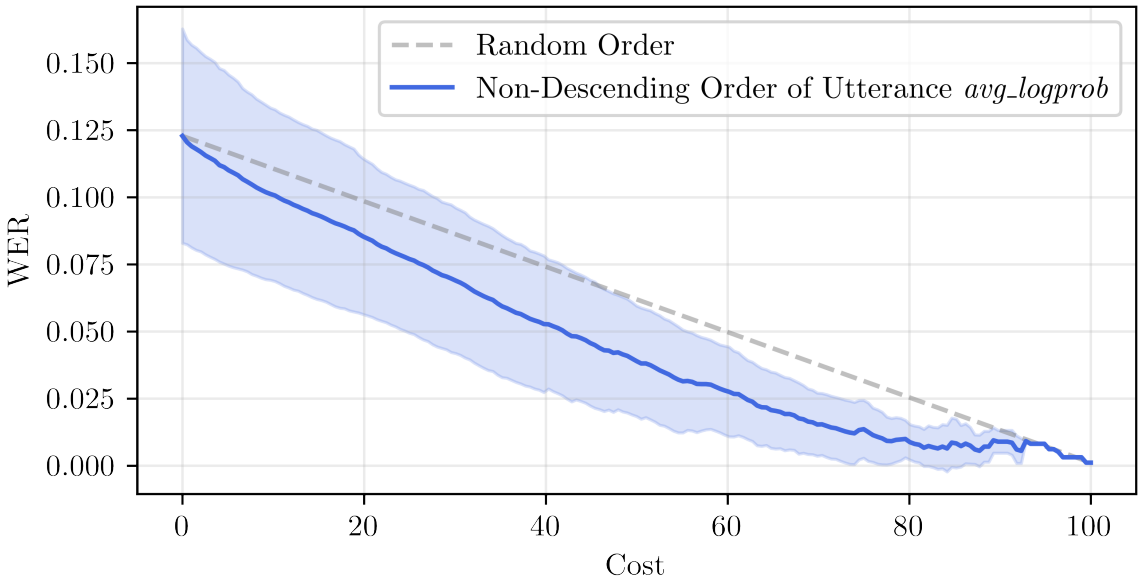


Figure 6.2: Per-conversation average WER when evaluating in non-descending order of utterance-average confidence

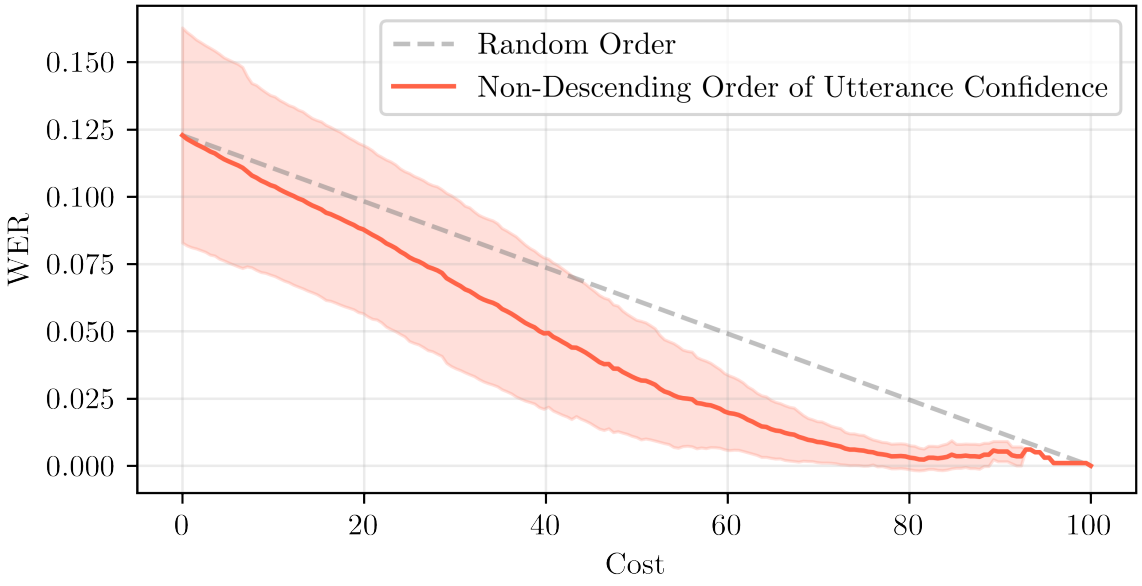


Figure 6.3: Comparing per-conversation average performance of confidence and `avg_logprob`

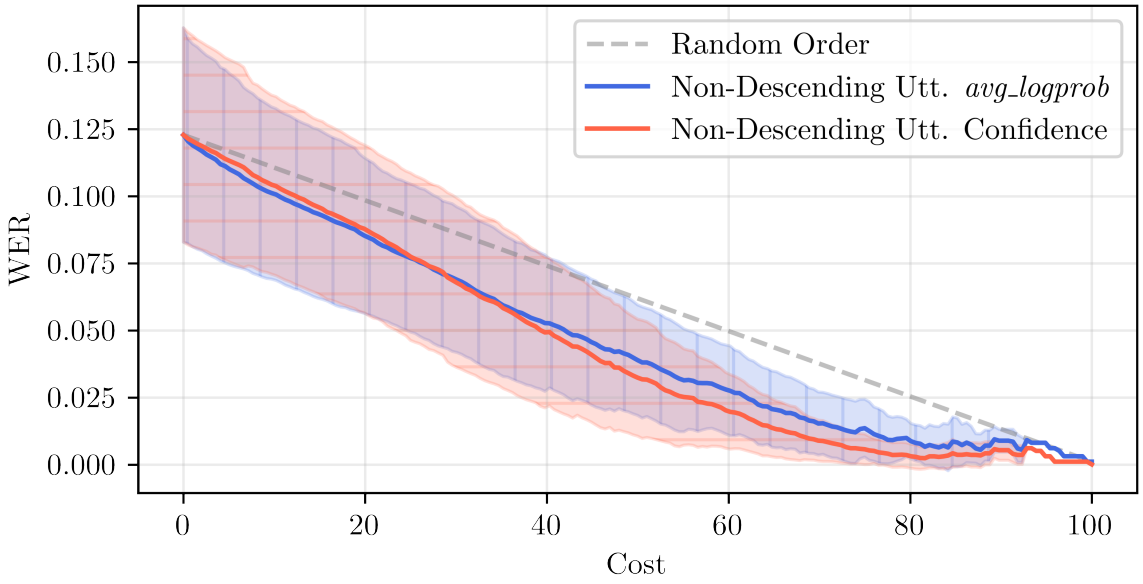
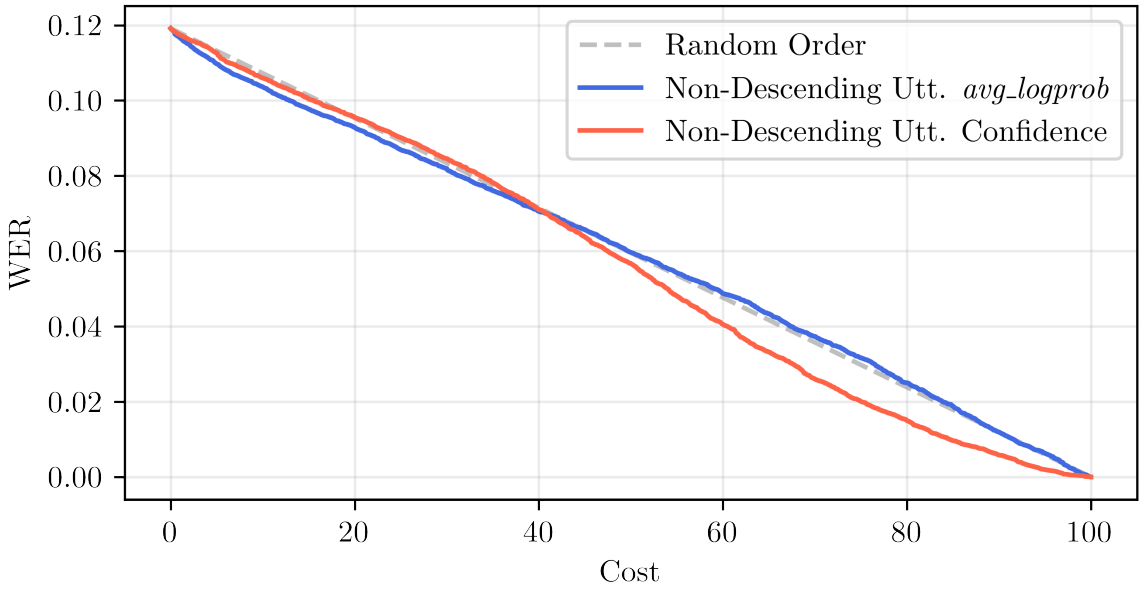


Figure 6.4: Comparing whole-corpus performance of confidence and `avg_logprob`



6.2.2 Per-conversation average results using word-confidence

Figure 6.5: Ordered by non-descending utterance-minimum word-confidence

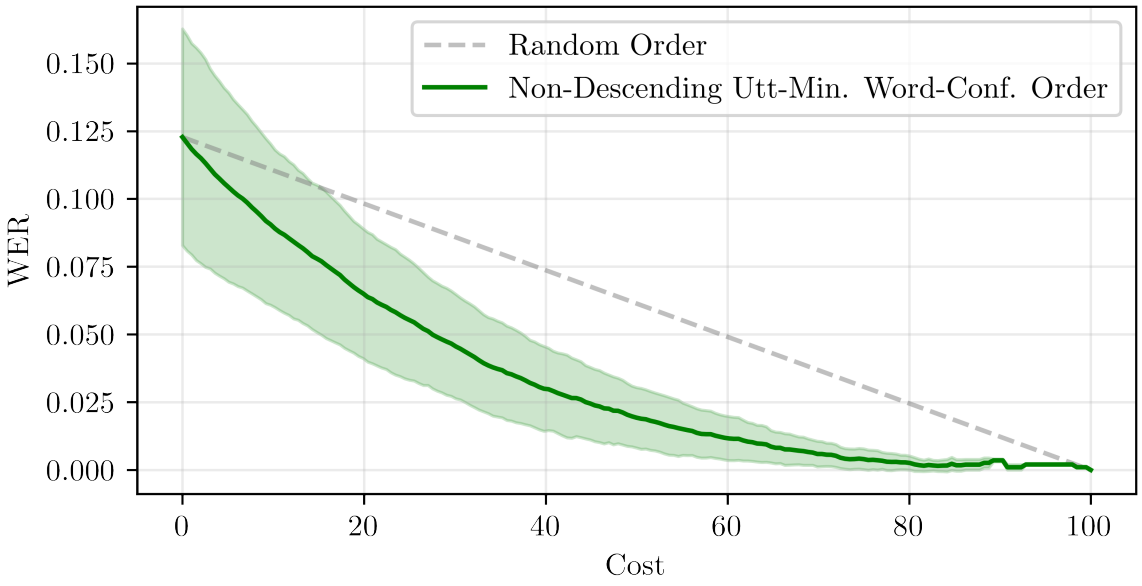


Figure 6.6: Ordered by non-ascending utterance-minimum word-confidence

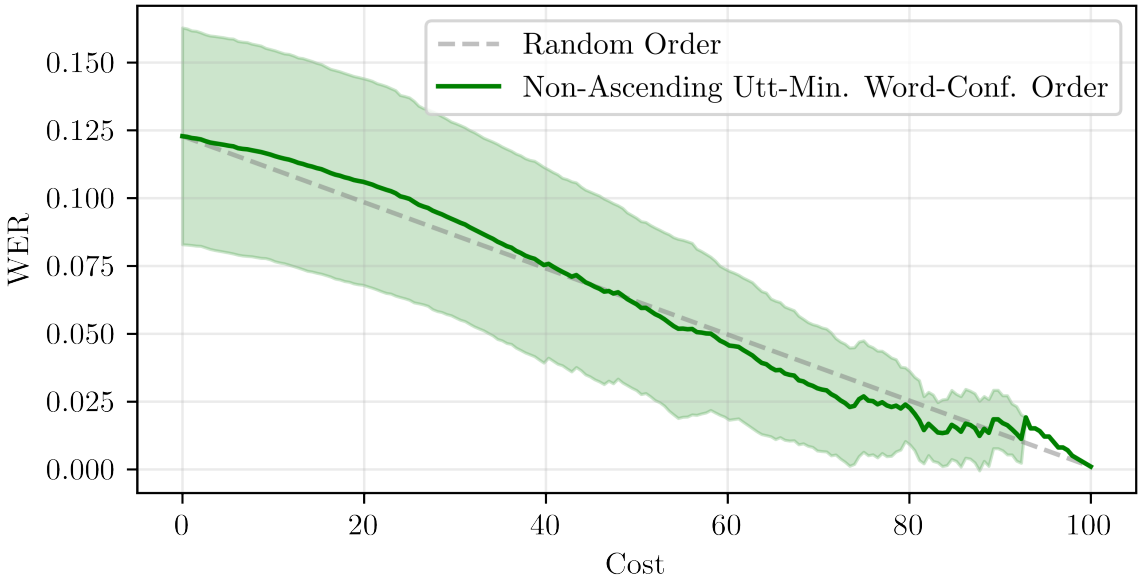


Figure 6.7: Ordered by non-descending utterance-maximum word-confidence

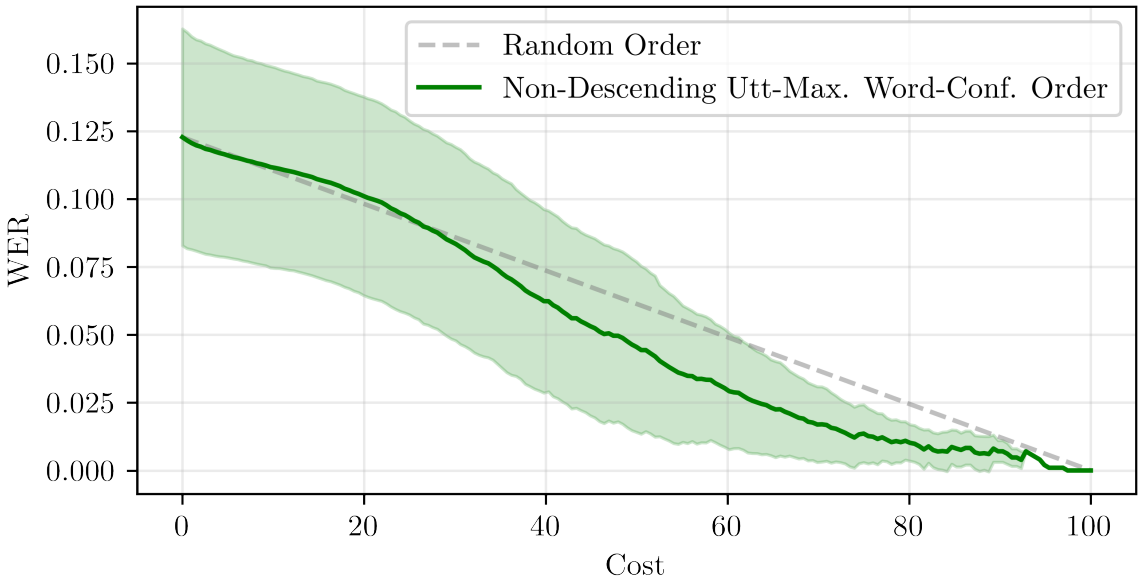
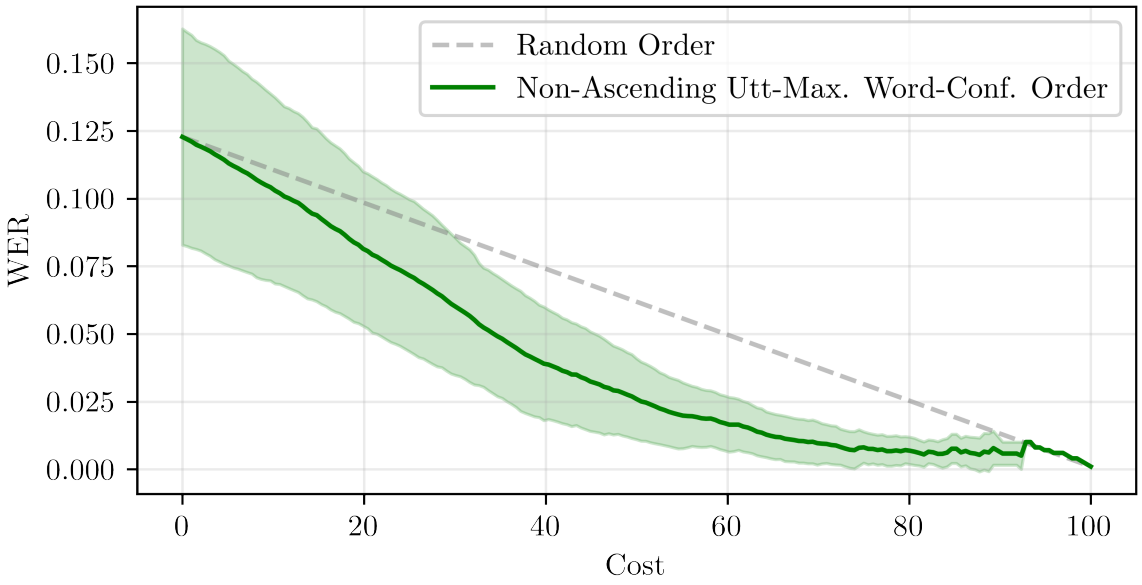


Figure 6.8: Ordered by non-ascending utterance-maximum word-confidence



6.2.3 Corpus-wide comparisons with word-level confidence ordering

Figure 6.9: Comparing whole-corpus evaluation performance with each word-confidence ordering

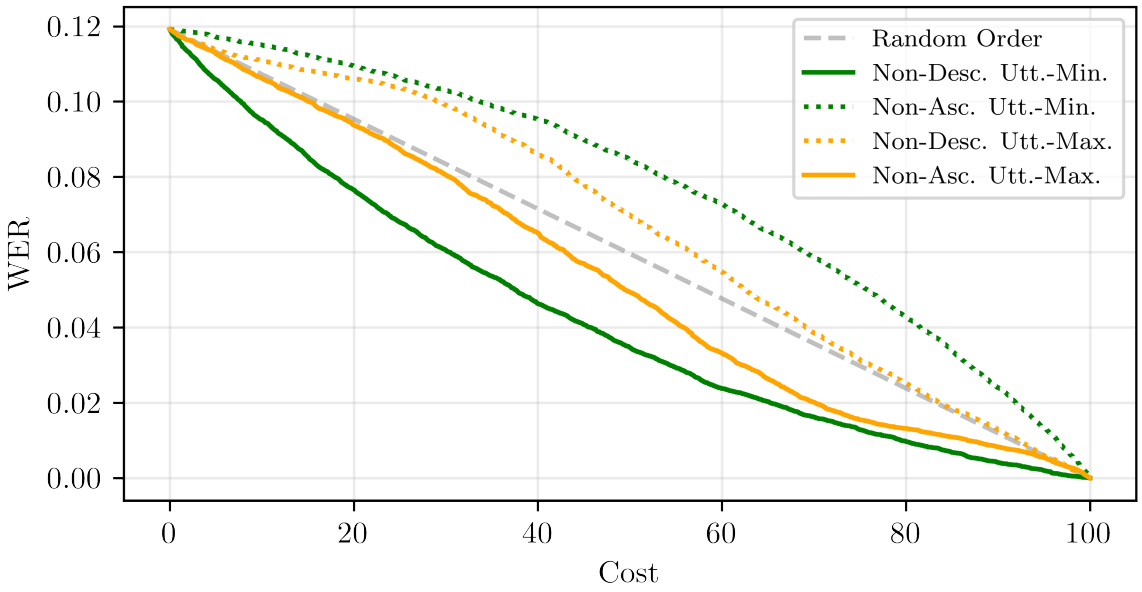
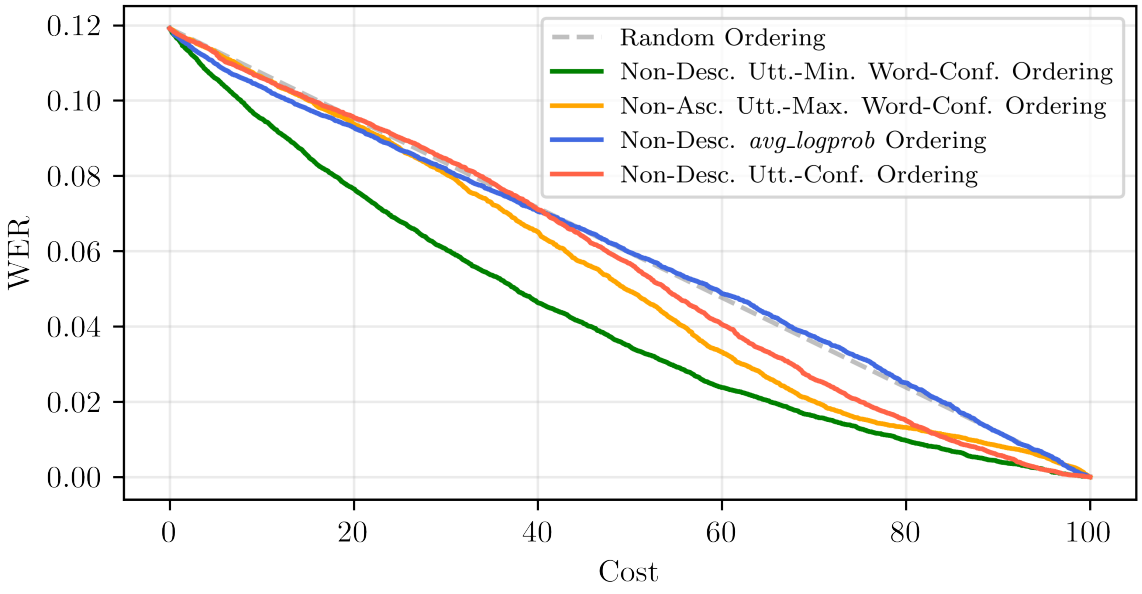


Figure 6.10: Comparing whole-corpus evaluation performance with each ordering metric



6.2.4 Different word-level confidence scoring techniques

Figure 6.11: Comparing whole-corpus evaluation performance with different word-level confidence metrics

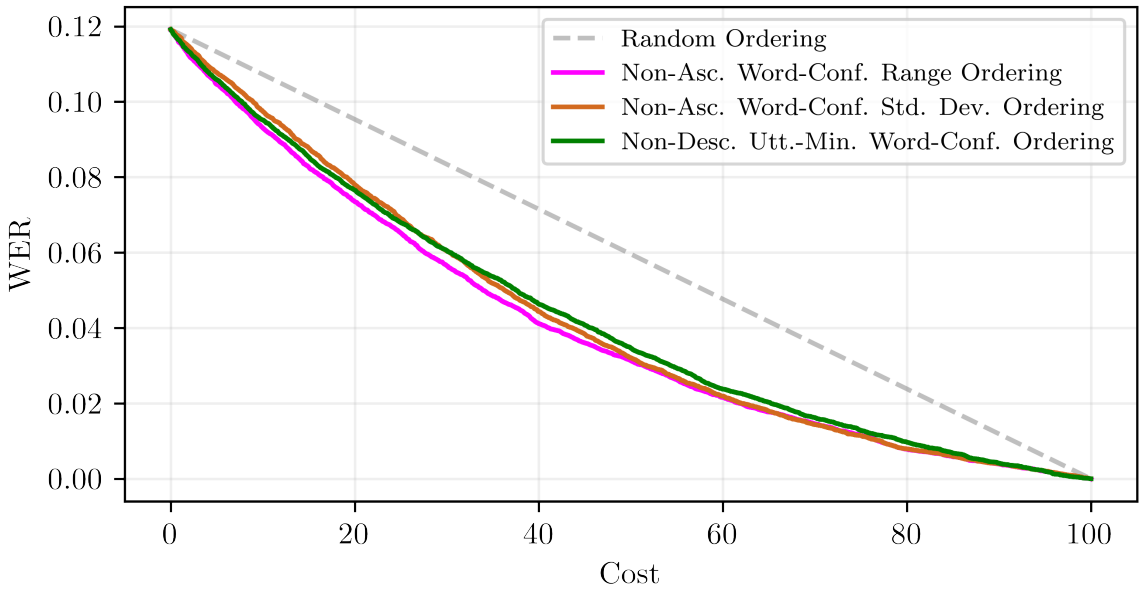
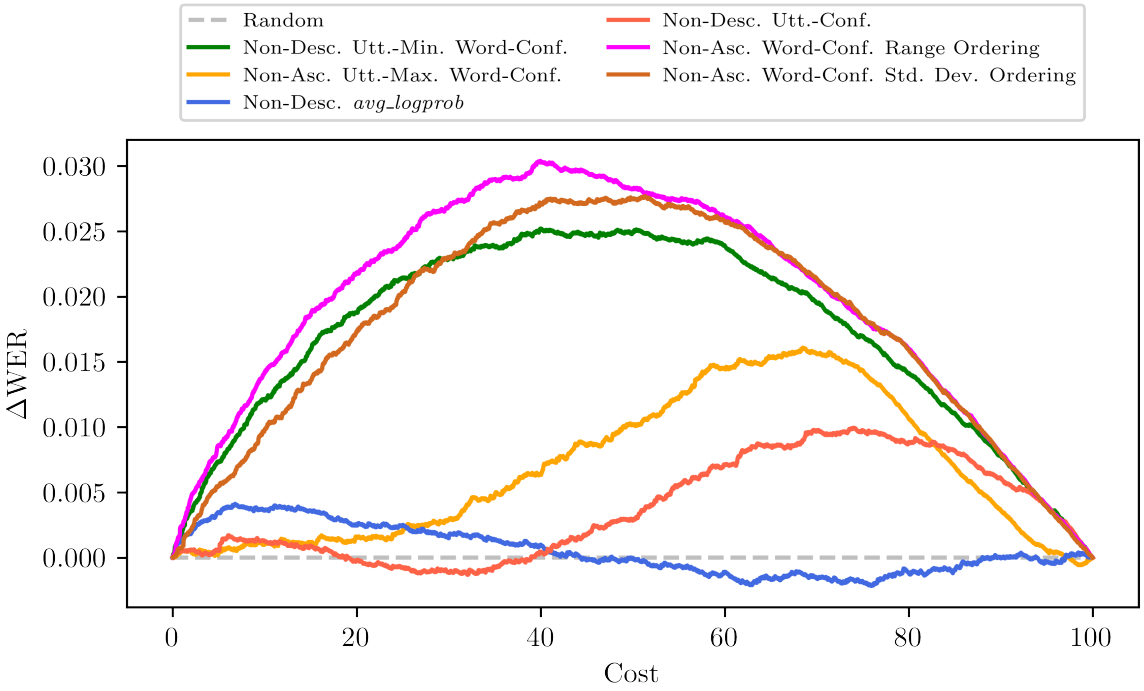


Figure 6.12: WER difference between metrics and random ordering



6.3 Discussion of Results

This section shall discuss findings which can be drawn from these results.

6.3.1 Utterance-average confidence and `avg_logprob`

The results show that `avg_logprob` and utterance-average confidence perform relatively similarly on average for a given conversation (fig. 6.3), with similarly poor performance when used to order the entire corpus (fig. 6.4). Utterance-average confidence appears to have marginally superior performance in both comparisons shown in section 6.2.2.

The similarity between the plots of the utterance-average word confidence and Whisper's `avg_logprob` output is likely due to their similarity in calculation. Confidence is calculated by *whisper-timestamped*[39] from the average of the word-level confidence scores in an utterance, which are calculated from the average of the sub-word log-probabilities for each word. Whisper[4] calculates `avg_logprob` as simply the average of the log-probabilities of all tokens in an utterance. Perhaps the slight performance increase seen when using utterance-average confidence scores could be explained due to it being calculated from word-level scores rather than token-level and the accuracy metric operating at the word-level rather than token-level too.

6.3.2 Word-level confidence metrics

The results presented in section 6.2.2 show the per-conversation average relationship between WER and Cost for the non-ascending and non-descending utterance-minimum and -maximum word-confidence metrics.

It is apparent that for the utterance-minimum word-confidence metric it is best to use non-descending (fig. 6.5) rather than non-ascending order (fig. 6.6), as non-ascending order closely follows the random order (i.e. is ineffective). This makes intuitive sense because ordering results on non-ascending minimum confidence would mean correcting the most confident results first, thus the opposite of an effective ordering method.

Notice that non-descending utterance-minimum word-confidence ordering (fig. 6.5) has the narrowest shaded area of all the plots presented, meaning it has the lowest standard deviation from the mean and therefore the most reliable representation of performance on a given conversation.

As for using utterance-maximum word-confidence ordering, non-ascending order (fig. 6.8) shows superior performance to non-descending (fig. 6.7). The reason for this is less intuitive; it would make more sense that ordering from lowest maximum confidence to highest (i.e. non-descending) would put utterances with lower confidence scores first. This could be due to word-confidence being derived from word-level probability scores, though this is an area which needs more experimentation to better understand this result.

When comparing whole-corpus evaluation results using word-confidence measures (fig. 6.9), non-descending utterance-minimum word-confidence ordering has the best (largest difference from random) and most consistent (smoothest line) performance of all the metrics, followed by non-ascending utterance-maximum word-confidence. The other two metrics are shown to

be a hindrance, achieving worse performance than if the results were evaluated in random order and should thus be ignored as metrics to use in a computer-aided transcription system.

6.3.3 Comparing word-level and utterance-level metrics

Figure 6.10 presents a comparison between the performance of each metric when evaluating the whole corpus. The metric with the best performance is clearly non-descending utterance-minimum word-confidence; it is much more consistent than the others and remains much further from random ordering, meaning it has the highest WER reduction for the same cost as all other metrics.

Utterance Ordering Metric	**Cost to reduce WER by 50%**
Random	50.0%
Non-descending utterance-minimum word-confidence	30.7%
Non-ascending word-confidence range	**27.9%**
Non-ascending word-confidence standard deviation	30.6%
Non-descending `avg_logprob`	50.1%
Non-descending utterance-confidence	47.7%
Non-descending utterance-maximum word-confidence	43.2%

Table 6.1: Cost required to halve WER.

Figure 6.11 shows the performance of non-ascending ordering using the both range scores and standard deviation of word-confidence for each utterance. Notice the slight performance gains over non-descending utterance-minimum word-confidence.

These metrics show slight performance gains over non-descending utterance-minimum word-confidence, as illustrated in figure 6.12 which plots the difference in WER from using random ordering (ΔWER) against Cost.

Table 6.3.3 presents the cost required to halve WER, with non-ascending word-confidence range being the best performer. According to these results, a computer-aided transcription system using this metric would require only 28% of the results to be checked in order to halve the WER of the ASR output, in this case falling to approximately 6% WER. Considering that the corpus has been segmented into 7312 utterances, fewer than 2050 of them would require manually correcting to reduce the WER by half when ordered using this metric. For reference, to achieve this result by correcting the results in random order would require checking over 3600 pieces of audio, or 44% more.

6.4 Future Work

This section shall highlight some of the areas which this project has failed to adequately address and present potential avenues for further work.

6.4.1 Increase test corpus

This work has focused on only one speech corpus, *LifeLUCID*[34]. Though it presents a diverse range of speaker ages, it has made for a relatively small test sample (less than 9 hours total). Thorough testing on a variety of speech corpora would provide validation to the results presented earlier in this chapter. Other languages could be used to increase the diversity of the test corpus because Whisper is capable of operating on various different languages (though is trained on 65% English data[4]).

Despite the limited test corpus used in this work, the results of ordering utterances using metrics based on word-level confidence scores show a very clear benefit to the efficiency of a semi-automatic transcription system. This benefit may differ in magnitude across corpora, though it seems highly unlikely that it would not provide a benefit to other similar corpora (i.e. consisting of English speakers).

6.4.2 Acquire results from human transcribers

The rudimentary simulation presented in section 6.1 is built on the assumption that a result is free from errors once human-corrected. In reality, this is not always the case; humans often make mistakes, and therefore would produce different results from what was presented in this chapter.

An experiment using human participants would require a working computer-aided transcription software; the software demonstration presented for this project is not fully-featured and would require a small amount of modification before it could be used for transcription. Specifically, it is not capable of taking text input from the user, though it can order, highlight and blank-out utterances, as well as play accompanying audio files.

6.4.3 Experiment with other derivations of confidence

This work has used confidence scoring derived from a single unmodified output from Whisper (`avg_logprob`) and internal probability scoring. While ordering utterances based on minimum word-level confidence scores derived from internal probability scoring has shown a degree of effectiveness, there is expansive literature discussing methods of neural network confidence measures which have not been implemented in this work.

For example, use of a neural network (e.g. a multilayer perception) which takes Whisper's output as input and outputs a confidence metric could be shown to provide a reliable metric for confidence. Similar methods are proposed in the literature, such as using recurrent neural networks[48, 49] or a naïve Bayes classifier[50]. The benefit of this method would be that Whisper doesn't require any (or minimal) modification to apply them.

To avoid spending time experimenting with modifications to Whisper's architecture, a mildly modified version of Whisper called *whisper-timestamped*[39] was used in this project. An 'entropy-based' approach for calculating word-confidence is proposed in the literature[51, 52], though implementation would require considerable modification to the Transformer which Whisper uses.

A solution to this problem may be to use a different model than Whisper, though at the time of writing there doesn't appear to be another free and open-source ASR model with similar performance.

Chapter 7

Conclusions

In conclusion, this work has presented a method for calculating confidence scores from Whisper and produced a method to order results to corrected by a human. Three of the metrics on which transcript utterances may be ordered have been presented to provide a considerable benefit to a human transcriber; they reduce the number of results which must be manually checked to reduce the WER. In fact, according to the simulation used to generate results, 44% fewer utterances need to be checked using the best metric than if the results weren't checked in order to achieve a halving of WER.

A piece of software to demonstrate the possibility to aid a human transcriber has been produced. Despite not being fully-featured, it would not take much modification to transform it into a more capable system to be used to produce accurate transcripts more quickly than fully-manual transcription. All software libraries used in development are free and open-source, meaning the proposed system is accessible to anyone who wishes to quickly produce accurate transcripts without incurring large costs.

While confidence measures which seem to perform well were produced, literature on the subject point to many other methods of deriving system confidence which were not explored. Given further work, an exploration of other methods for measuring Whisper's confidence may find more performative results.

The LifeLUCID speech corpus was used to produce all results. It contains a diverse range of speaker ages but other demographic information about the speakers is unavailable. The results presented in this report should be treated as preliminary; further testing across corpora should aim to validate results, as well as perform testing with real human subjects.

Bibliography

[1] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 12 449–12 460. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/92d1e1eb1cd6f9fba3227870bb6d7f07-Paper.pdf

[2] P. Szymański, P. Żelasko, M. Morzy, A. Szymczak, M. Żyła-Hoppe, J. Banaszczak, L. Augustyniak, J. Mizgajski, and Y. Carmiel, "Wer we are and wer we think we are," *arXiv preprint arXiv:2010.03432*, 2020.

[3] Rev, "Transcribe Speech to Text | Rev," May 2023, [Online; accessed 24. May 2023]. [Online]. Available: https://www.rev.com

[4] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," arXiv:2212.04356, 2022.

[5] L. R. Rabiner, "Automatic Speech Recognition - A Brief History of the Technology Development," *Scinapse*, Jan. 2004. [Online]. Available: https://www.scinapse.io/papers/187290754

[6] D. W. H., "The vocoder," *Bell. Labs. Rec.*, vol. 18, p. 122, 1939. [Online]. Available: https://cir.nii.ac.jp/crid/1572261551231523968

[7] K. H. Davis, R. Biddulph, and S. Balashek, "Automatic recognition of spoken digits," *The Journal of the Acoustical Society of America*, vol. 24, no. 6, pp. 637–642, 1952. [Online]. Available: https://doi.org/10.1121/1.1906946

[8] J. K. Baker, *Stochastic modeling as a means of automatic speech recognition.* Carnegie Mellon University, 1975.

[9] Y. Bengio *et al.*, "Markovian models for sequential data," *Neural computing surveys*, vol. 2, no. 199, pp. 129–162, 1999.

[10] X. Dong, W. Cao, H. Cheng, and T. Zhang, "Hidden markov model-driven speech recognition for power dispatch," in *Tenth International Conference on Applications*

and Techniques in Cyber Intelligence (ICATCI 2022), J. H. Abawajy, Z. Xu, M. Atiquzzaman, and X. Zhang, Eds. Cham: Springer International Publishing, 2023, pp. 760–768.

[11] E. B. Dynkin, *Theory of Markov Processes*, T. Kóváry, Ed. Pergamon Press, 1960.

[12] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. IEEE*, vol. 77, no. 2, pp. 257–286, Feb. 1989.

[13] D. Wang, X. Wang, and S. Lv, "An overview of end-to-end automatic speech recognition," *Symmetry*, vol. 11, no. 8, p. 1018, 2019.

[14] M. K. Mustafa, T. Allen, and K. Appiah, "A comparative review of dynamic neural networks and hidden markov model methods for mobile on-device speech recognition," *Neural Computing and Applications*, vol. 31, pp. 891–899, 2019.

[15] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International conference on machine learning*. PMLR, 2016, pp. 173–182.

[16] T. Hori, S. Watanabe, Y. Zhang, and W. Chan, "Advances in joint ctc-attention based end-to-end speech recognition with a deep cnn encoder and rnn-lm," *arXiv preprint arXiv:1706.02737*, 2017.

[17] S. Kim, T. Hori, and S. Watanabe, "Joint CTC-attention based end-to-end speech recognition using multi-task learning," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, Mar. 2017, pp. 4835–4839.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[19] N. Zeghidour, Q. Xu, V. Liptchinsky, N. Usunier, G. Synnaeve, and R. Collobert, "Fully convolutional speech recognition," *arXiv preprint arXiv:1812.06864*, 2018.

[20] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *International conference on machine learning*. PMLR, 2014, pp. 1764–1772.

[21] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," *arXiv preprint arXiv:1803.02155*, 2018.

[22] A. Fang, S. Filice, N. Limsopatham, and O. Rokhlenko, "Using phoneme representations to build predictive models robust to asr errors," in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser.

SIGIR '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 699–708. [Online]. Available: https://doi.org/10.1145/3397271.3401050

[23] S. Nießen, F. J. Och, G. Leusch, H. Ney *et al.*, "An evaluation tool for machine translation: Fast evaluation for mt research." in *LREC*, 2000.

[24] Y. Park, S. Patwardhan, K. Visweswariah, and S. C. Gates, "An empirical analysis of word error rate and keyword error rate." in *INTERSPEECH*, vol. 2008, 2008, pp. 2070–2073.

[25] H. B. Pasandi and H. B. Pasandi, "Evaluation of automated speech recognition systems for conversational speech: A linguistic perspective," *arXiv preprint arXiv:2211.02812*, 2022.

[26] Y. Zhang, D. S. Park, W. Han, J. Qin, A. Gulati, J. Shor, A. Jansen, Y. Xu, Y. Huang, S. Wang, Z. Zhou, B. Li, M. Ma, W. Chan, J. Yu, Y. Wang, L. Cao, K. C. Sim, B. Ramabhadran, T. N. Sainath, F. Beaufays, Z. Chen, Q. V. Le, C.-C. Chiu, R. Pang, and Y. Wu, "BigSSL: Exploring the frontier of large-scale semi-supervised learning for automatic speech recognition," *IEEE Journal of Selected Topics in Signal Processing*, vol. 16, no. 6, pp. 1519–1532, oct 2022. [Online]. Available: https://doi.org/10.1109%2Fjstsp.2022.3182537

[27] Y.-A. Chung, Y. Zhang, W. Han, C.-C. Chiu, J. Qin, R. Pang, and Y. Wu, "W2v-bert: Combining contrastive learning and masked language modeling for self-supervised speech pre-training," 2021. [Online]. Available: https://arxiv.org/abs/2108.06209

[28] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An asr corpus based on public domain audio books," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 5206–5210.

[29] Y. Zhang, J. Qin, D. S. Park, W. Han, C.-C. Chiu, R. Pang, Q. V. Le, and Y. Wu, "Pushing the limits of semi-supervised learning for automatic speech recognition," 2020.

[30] E. Shriberg, A. Stolcke, and D. Baron, "Observations on overlap: findings and implications for automatic processing of multi-party conversation." in *Interspeech*. Citeseer, 2001, pp. 1359–1362.

[31] A. Koenecke, A. Nam, E. Lake, J. Nudell, M. Quartey, Z. Mengesha, C. Toups, J. R. Rickford, D. Jurafsky, and S. Goel, "Racial disparities in automated speech recognition," *Proceedings of the National Academy of Sciences*, vol. 117, no. 14, pp. 7684–7689, 2020.

[32] A. Baevski, W.-N. Hsu, A. CONNEAU, and M. Auli, "Unsupervised speech recognition," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 27 826–27 839. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/ea159dc9788ffac311592613b7f71fbb-Paper.pdf

[33] W. S. Horton, D. H. Spieler, and E. Shriberg, "A corpus analysis of patterns of age-related change in conversational speech." *Psychol. Aging*, vol. 25, no. 3, p. 708, 2010.

[34] O. Tuomainen, L. Taschenberger, and V. Hazan, "LifeLUCID Corpus: Recordings of Speakers Aged 8 to 85 Years Engaged in Interactive Task in the Presence of Energetic and Informational Masking, 2017-2020," *UK Data Service*, May 2021. [Online]. Available: https://reshare.ukdataservice.ac.uk/854350

[35] "Praat: doing Phonetics by Computer," Mar. 2023, [Online; accessed 8. Apr. 2023]. [Online]. Available: https://www.fon.hum.uva.nl/praat

[36] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, "Comparison of json and xml data interchange formats: a case study." *Caine*, vol. 9, pp. 157–162, 2009.

[37] Y.-A. Wang and Y.-N. Chen, "What do position embeddings learn? an empirical study of pre-trained language model positional encoding," *arXiv preprint arXiv:2010.04903*, 2020.

[38] openai, "tiktoken," Apr. 2023, [Online; accessed 24. April 2023]. [Online]. Available: https://github.com/openai/tiktoken

[39] J. Louradour, "whisper-timestamped," https://github.com/linto-ai/whisper-timestamped, 2023.

[40] T. Giorgino, "Computing and visualizing dynamic time warping alignments in r: The dtw package," *Journal of Statistical Software*, vol. 31, no. 7, 2009.

[41] Textualize, "textual," Apr. 2023, [Online; accessed 24. April 2023]. [Online]. Available: https://github.com/Textualize/textual

[42] B. Bechtold, "python-soundfile," 2013, [Online; accessed 8. Apr. 2023]. [Online]. Available: https://github.com/bastibe/python-soundfile

[43] spatialaudio, "python-sounddevice," Apr. 2023, [Online; accessed 24. April 2023]. [Online]. Available: https://github.com/spatialaudio/python-sounddevice

[44] Textualize, "rich," Apr. 2023, [Online; accessed 25. April 2023]. [Online]. Available: https://github.com/Textualize/rich

[45] "High Performance Computing at Sheffield — Sheffield HPC Documentation," May 2023, [Online; accessed 6. May 2023]. [Online]. Available: https://docs.hpc.shef.ac.uk/en/latest

[46] jitsi, "jiwer," Apr. 2023, [Online; accessed 26. April 2023]. [Online]. Available: https://github.com/jitsi/jiwer

[47] "Pyqt," May 2023, [Online; accessed 24. May 2023]. [Online]. Available: https://www.riverbankcomputing.com/software/pyqt

[48] P.-S. Huang, K. Kumar, C. Liu, Y. Gong, and L. Deng, "Predicting speech recognition confidence using deep learning with word identity and score features," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 7413–7417.

[49] K. Kalgaonkar, C. Liu, Y. Gong, and K. Yao, "Estimating confidence scores on asr results using recurrent neural networks," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 4999–5003.

[50] A. Sanchis, A. Juan, and E. Vidal, "A word-based naïve bayes classifier for confidence estimation in speech recognition," *IEEE transactions on audio, speech, and language processing*, vol. 20, no. 2, pp. 565–574, 2011.

[51] A. Laptev and B. Ginsburg, "Fast entropy-based methods of word-level confidence estimation for end-to-end automatic speech recognition," in *2022 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2023, pp. 152–159.

[52] D. Qiu, Q. Li, Y. He, Y. Zhang, B. Li, L. Cao, R. Prabhavalkar, D. Bhatia, W. Li, K. Hu *et al.*, "Learning word-level confidence for subword end-to-end asr," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 6393–6397.

Appendices

Appendix A

Example of the TextGrid Format

This example serves to illustrate the lack of readability of a TextGrid file. To aid formatting, it is displayed in two columns, though the original file is a long series of *intervals*.

It represents a piece of audio which is under three seconds in length (from a 10 minute-long recording) and consists of the text "a bush with a yellow duck on top".

```
intervals [12]:
  xmin = 20.899
  xmax = 20.971783458461772
  text = "SIL"
intervals [13]:
  xmin = 20.971783458461772
  xmax = 21.05
  text = "a"
intervals [14]:
  xmin = 21.05
  xmax = 21.47
  text = "BUSH"
intervals [15]:
  xmin = 21.47
  xmax = 21.66
  text = "with"
intervals [16]:
  xmin = 21.66
  xmax = 21.720024609817834
  text = "A"
intervals [17]:
  xmin = 21.720024609817834
  xmax = 22.1
  text = "SIL"
intervals [18]:
  xmin = 22.1
  xmax = 22.49
  text = "yellow"
intervals [19]:
  xmin = 22.49
  xmax = 22.84
  text = "duck"
intervals [20]:
  xmin = 22.84
  xmax = 23.06
  text = "ON"
intervals [21]:
  xmin = 23.06
  xmax = 23.769
  text = "top"
```

Appendix B

get_utterances

```
#!/usr/bin/env python
"""
get_utterances

Given a directory of Praat TextGrid files, this program segments them all
into utterances according to the documentation provided by with the LifeLUCID
corpus (V.Hazan et al.).

The utterances are written to JSON files in a given output directory.

Copyright Adam Spencer, 03/2023
"""

__author__ = 'Adam Spencer'

import argparse
import json
from pathlib import Path
from typing import Union

import textgrid as tg

# As described in documentation
BREAK_TOKENS = {'SILP', '<GA>'}
JUNK_TOKENS = {
    'SIL', '<BELL>', '<bell>', '<lg>', '<LG>', '<br>', '<BR>', 'er', 'erm',
    'ermm', 'um', 'umm', 'uhm', 'uhmm', 'uh', 'uhh', 'uhhh', 'ER', 'ERM',
    'ERMM', 'UM', 'UMM', 'UHM', 'UHMM', 'UH', 'UHH', 'UHHH'
}
```

```
def find_utterances(grid: tg.TextGrid, /, normalise: bool, min_break: float,
                    max_pause: float) -> dict[int, dict[str, Union[float, str]]]:
    """
    Find all utterances in a Praat TextGrid file, returning start and end times
    and the transcription as provided in the TextGrid.

    Utterance ends and beginnings are found using the `BREAK_TOKENS`, and any
    non-speaking tokens (as defined in `JUNK_TOKENS`) are removed.

    :param grid: The Praat TextGrid to segment into utterances.
    :param normalise: Enable lowercase text normalisation.
    :param min_break: Minimum time between tokens required for break.
    :param max_pause: Maximum allowed pause time within an utterance.
    :returns: dict of structure
              { segment_num -> { start_time, end_time, transcript } }
    """
    segments = dict()
    start_time = float()
    seg_words = list()
    seg_counter = 0
    seg_ongoing = False
    interval_tier = grid[0]

    for i in range(0, len(interval_tier)):
        break_now = False
        interval = interval_tier[i]
        duration = interval.duration()

        # Find non-speech tokens and decide whether or not to end the utterance
        # NOTE: `continue` statments are used to prevent writing non-speech
        # tokens to the transcript
        if interval.mark in JUNK_TOKENS:
            if seg_ongoing and duration > max_pause:
                break_now = True
            else:
                continue
        elif interval.mark in BREAK_TOKENS:
            if seg_ongoing and duration > min_break:
                break_now = True
```

```
            else:
                continue

        # End the current utterance
        if break_now:
            segments[seg_counter] = {'start': start_time, 'end': interval.minTime,
                                     'transcript': ' '.join(seg_words)}
            seg_words = []
            seg_counter += 1
            seg_ongoing = False
            continue

        # Begin a new segment
        if not seg_ongoing:
            start_time = interval.minTime
            seg_ongoing = True
        if normalise:
            interval.mark = interval.mark.lower()
        seg_words.append(interval.mark)
    return segments


def write_to_json(utterances: dict, output_file: Path) -> None:
    """
    Write utterances out to the given output directory in JSON format.

    :param utterances: Utterances dict
    structure { segment_num -> { start_time, end_time, transcript } }
    :param output_file: Path to output file.
    """
    with open(output_file, 'w') as f:
        json.dump(utterances, f, indent=2)


def main():
    # Parse Arguments
    parser = argparse.ArgumentParser()
    parser.add_argument('in_dir', help='Name of dir to find textgrids in')
    parser.add_argument('out_dir', help='Name of dir to write JSON files to')
    parser.add_argument('--no-normalise', '-n', action='store_true',
                        help='Disable case normalisation')
```

```
    parser.add_argument('--min-break-threshold', '-b', type=float, default=None,
                        help='Minimum time between tokens required for break')
    parser.add_argument('--max-pause-threshold', '-p', type=float, default=None,
                        help='Maximum allowed pause in a single utterance')
    args = parser.parse_args()
    normalise = not args.no_normalise

    # New Path objects @ specified paths
    in_dir_path = Path(args.in_dir)
    out_dir_path = Path(args.out_dir)
    for p in [in_dir_path, out_dir_path]:
        if not p.is_dir():
            raise ValueError(f'{p} is Not a directory!')

    # Find utterences in input dir, output JSON to output dir
    for file in in_dir_path.iterdir():
        if '.TextGrid' not in file.name:
            continue
        grid = tg.TextGrid.fromFile(file)
        utterances = find_utterances(grid, normalise=normalise,
                                     min_break=args.min_break_threshold,
                                     max_pause=args.max_pause_threshold)
        out_path = out_dir_path / file.name.replace('TextGrid', 'json')
        write_to_json(utterances, out_path)


if __name__ == '__main__':
    main()
```

Appendix C

segment_audio

```
#!/usr/bin/env python
"""
segment_audio

Generate audio segments to some specification described in a JSON file.

Copyright Adam Spencer, 2023.
"""

__author__ = 'Adam Spencer'

import soundfile as sf
import json
import argparse
from pathlib import Path

from utils import vprint

def generate_segments(filename:Path, spec_file:Path, out_dir:Path, channel:int,
                      v:bool) -> None:
  """
  Generate audio segments given an audio file and segment metadata.

  :param filename: Path to input file.
  :param spec_file: Path to JSON segment metadata.
  :param channel: Which audio channel is being segmented.
  :param v: Verbose output.
  :returns: List containing segmented audio represented by Numpy arrays.
  """
```

```
  vprint(v, 'Loading metadata...')
  with open(spec_file) as f:
    seg_data = json.load(f)

  vprint(v, 'Loading audio...')
  sig, fs = sf.read(filename)
  sig = sig[:, channel]

  vprint(v, 'Beginning segmentation...')
  for seg_n, seg_meta in seg_data.items():
    slice_start = int(fs * seg_meta['start'])
    slice_end = int(fs * seg_meta['end'])
    outpath = out_dir / f'{int(seg_n):03d}.wav'
    sf.write(outpath, sig[slice_start:slice_end], fs)
  vprint(v, 'Done!')

def run_over_dir(audio_dir:Path, spec_dir:Path, out_dir:Path,
                 channel:int, v:bool) -> None:

  # IDEALLY THIS WOULD WORK, BUT IT DOESN'T!
  # Must find the matching spec and audio files, then generate a new output
  # subdirectory, then pass these into generate_segments() in a loop.
  # if channel == 0:
  #   spec_ext = 'Ac.json'
  # else:
  #   spec_ext = 'Bc.json'

  spec_ext = 'Ac.json'

  vprint(v, 'Looping over files...')
  counter = 0
  total_files = 0
  if v:
    total_files = len(list(audio_dir.glob('*')))

  for audio_file in audio_dir.iterdir():
    if v:
      counter += 1
      print(f'Progress: {counter} / {total_files}', end='\r')
    purename = audio_file.name.removesuffix('.wav')
    spec_file = spec_dir / f'{purename}_{spec_ext}'
```

```
    if spec_file.exists():
      out_subdir = out_dir / purename
      out_subdir.mkdir(exist_ok=True)
      generate_segments(audio_file, spec_file, out_subdir, channel, False)
    else:
      vprint(v, f"Couldn't find a spec file for {purename}!")

def main() -> None:
  parser = argparse.ArgumentParser()
  parser.add_argument('audio_file', help='Input audio file/directory')
  parser.add_argument('spec_file',
                      help='Input JSON file/directory containing segment \
                      specification')
  parser.add_argument('out_dir', help='Output directory')
  parser.add_argument('--channel', '-c', choices=[0,1], type=int,
                      help='Specify channel, 0 = L, 1 = R',
                      default=0)
  parser.add_argument('--dir-mode', '-d', action='store_true',
                      help='Activate directory mode')
  parser.add_argument('--verbose', '-v', action='store_true',
                      help='Activate verbose output')
  args = parser.parse_args()

  audio_file = Path(args.audio_file)
  spec_file = Path(args.spec_file)
  out_dir = Path(args.out_dir)

  # This is mildly janky; there must be a better way!
  if out_dir.is_dir():
    if args.dir_mode and audio_file.is_dir() and spec_file.is_dir():
      run_over_dir(audio_file, spec_file, out_dir, args.channel, args.verbose)
      return
    elif not args.dir_mode and not audio_file.is_dir() and not spec_file.is_dir():
      generate_segments(audio_file, spec_file, out_dir, args.channel, args.verbose)
      return
  raise ValueError('Incorrect path specification!')

if __name__ == '__main__':
  main()
```

Appendix D

do_whisper_confidence

```
#!/usr/bin/env python
"""
do_whisper_confidence

Given a directory of audio segment subdirectories and accompanying directory of
JSON specification files, run `whisper-timestamped` on the audio and write the output to
a specified destination.

Copyright Adam Spencer, 03/2023
"""

__author__ = 'Adam Spencer'

import argparse
import json
import re
from pathlib import Path

import whisper_timestamped as whisper

WAVFILE = re.compile(r'.+?(?=.wav)')


def run_whisper(model: whisper.Whisper, audio_dir: Path,
                opts: dict, v: bool
                ) -> dict[str, whisper.DecodingResult]:
    """
    Run Whisper over a directory of audio segments and return the results.
```

```
    :param model: Whisper model to use.
    :param audio_dir: Directory containing audio segments.
    :param opts: Decoding options for Whisper to use.
    :returns: Dict of structure: { key -> result }
    """
    result_dict = dict()
    for audio_file in audio_dir.iterdir():
        if fname := WAVFILE.match(audio_file.name):
            fname = fname.group().lstrip('0')
            if fname == '':
                fname = '0'
            result_dict[fname] = whisper.transcribe(
                model, str(audio_file), **opts)
            vprint(v, result_dict[fname]['text'])
    return result_dict


def write_to_json(whisper_result: dict, spec_file: Path, out_file: Path
                  ) -> None:
    """
    Write whisper result and utterence spec to a new JSON file at the output
    path.

    :param whisper_result: Results of running Whisper.
    :param spec_file: Path to JSON specification file.
    :param out_file: Path to output file.
    """
    with open(spec_file) as f:
        spec_dict = json.load(f)
    for k in spec_dict.keys():
        spec_dict[k].update(whisper=whisper_result[k])
    with open(out_file, 'w') as f:
        json.dump(spec_dict, f, indent=2, ensure_ascii=False)


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('audio_in',
                        help='Directory containing audio segment \
                        subdirectories')
    parser.add_argument('spec_dir',
```

```
                        help='Directory containing segment specification \
                            files')
    parser.add_argument('out_dir', help='Output directory')
    parser.add_argument('--model', default='base.en',
                        help='Whisper model to use, default `base.en`')
    parser.add_argument('--device', default='cpu',
                        help='Device for Whisper to use, default `cpu`')
    parser.add_argument('--no-fp16', action='store_true',
                        help='Disable FP16, useful on M1 Mac')
    parser.add_argument('--verbose', '-v', action='store_true',
                        help='Enable verbose output')
    args = parser.parse_args()

    audio_superdir = Path(args.audio_in)
    spec_dir = Path(args.spec_dir)
    out_dir = Path(args.out_dir)

    out_dir.mkdir(exist_ok=True)  # Create output directory
    if not (audio_superdir.is_dir() and spec_dir.is_dir()):
        raise ValueError(
            'One or more specified directories is not a directory!')

    model = whisper.load_model(args.model).to(args.device)
    opts = dict(language='en', fp16=(not args.no_fp16), best_of=5,
                beam_size=5, temperature=(0.0, 0.2, 0.4, 0.6, 0.8, 1.0))

    for audio_subdir in audio_superdir.iterdir():
        vprint(args.verbose, f'At folder: {audio_subdir}')
        if not audio_subdir.is_dir():
            continue
        if (spec_file := spec_dir / f'{audio_subdir.name}_Ac.json').exists():
            result = run_whisper(model, audio_subdir, opts, args.verbose)
            out_path = out_dir / f'{audio_subdir.name}_asr.json'
            vprint(args.verbose, 'Writing to JSON...')
            write_to_json(result, spec_file, out_path)


def vprint(verbose: bool, msg: str) -> None:
    """
    Verbose mode printing.
```

```
    :param verbose: Verbose mode bool.
    :param msg: Message to print.
    """
    if verbose:
        print(msg)


if __name__ == '__main__':
    main()
```

Appendix E

normalise_text

```
#!/usr/bin/env python
"""
normalise_text

Normalise the text output by Whisper.

Normalisation is done using `whisper.normalizer.EnglishTextNormalizer` and
removing hesitations as defined in `HESITATION_TOKENS`.
"""
import json
from argparse import ArgumentParser
from pathlib import Path

from whisper.normalizers import EnglishTextNormalizer

normaliser = EnglishTextNormalizer()

HESITATION_TOKENS = {
    'er', 'erm', 'ermm', 'um', 'umm', 'uhm', 'uhmm', 'uh', 'uhh', 'uhhh', 'ER',
    'ERM', 'ERMM', 'UM', 'UMM', 'UHM', 'UHMM', 'UH', 'UHH', 'UHHH'
}


def normalise_string(inp: str) -> str:
    """Remove hesitations and normalise a string"""
    inp_norm = map(normaliser, inp.split())
    to_return = []
    for token in inp_norm:
        if token in HESITATION_TOKENS:
```

```
            continue
        # This next step joins together tokens which have been normalised into
        # multiple words with an underscore, in order to facilitate mapping
        # confidence scores onto those words correctly
        elif len(post_norm_words := token.split()) > 1:
            to_return.append('_'.join(post_norm_words))
        else:
            to_return.append(token)
    return ' '.join(to_return)


def handle_confidence(segments: dict) -> dict:
    """
    Handle the confidence scores output by `whisper_timestamped`.

    :param segments: `utterance['whisper']['segments']`
    :returns: Confidence segments to be added to output dict
    """
    words_list = []
    confidence_list = []
    total_utterance_confidence = 0.0
    for segment in segments:
        if not (seg_conf := segment.get('confidence')):
            continue
        total_utterance_confidence += seg_conf
        for word in segment['words']:
            text = normalise_string(word['text'])
            if len(text) > 0:
                words_list.append(text)
                confidence_list.append(word['confidence'])
    words = ' '.join(words_list)
    avg_utt_confidence = total_utterance_confidence / len(segments)
    return {
        'words': words,
        'confidence_scores': confidence_list,
        'utterance_confidence': avg_utt_confidence,
    }


def normalise_file(file: Path, confidence_mode: bool) -> dict:
    """Load a whisper output file and normalise the text within"""
```

```
    with open(file) as f:
        data = json.load(f)
    for utterance in data.values():
        utterance['transcript'] = normalise_string(utterance['transcript'])
        utterance['whisper']['text'] = normalise_string(
            utterance['whisper']['text'])
        if confidence_mode and (segs := utterance['whisper'].get('segments')):
            utterance['confidence_scoring'] = handle_confidence(segs)
    return data


def write_to_output(file: Path, data: dict) -> None:
    """Write the normalised data to a JSON file as output"""
    with open(file, 'w') as f:
        json.dump(data, f, indent=2)


def main():
    parser = ArgumentParser()
    parser.add_argument('input', help='Input file or directory')
    parser.add_argument('output', help='Output directory')
    parser.add_argument('--confidence-scores', '-c', action='store_true',
                        help='Input contains confidence scores')
    parser.add_argument('--verbose', '-v', action='store_true',
                        help='Activate verbose output')
    args = parser.parse_args()

    input_path = Path(args.input)
    output_path = Path(args.output)
    output_path.mkdir(exist_ok=True)

    confidence_mode = args.confidence_scores
    verbose = args.verbose

    if input_path.is_dir():
        in_files = [f for f in input_path.iterdir()]
        n_files = len(in_files)
        for idx, file in enumerate(in_files):
            if verbose:
                print(f'Progress: {idx:3} / {n_files:3}', end='\r')
            if not (fname := file.name).endswith('.json'):
```

```
                continue
            data = normalise_file(file, confidence_mode)
            file_out = output_path / fname.replace('.json', '_norm.json')
            write_to_output(file_out, data)
    else:
        data = normalise_file(input_path, confidence_mode)
        file_out = output_path / input_path.name.replace('.json', '_norm.json')
        write_to_output(file_out, data)


if __name__ == "__main__":
    main()
```

Appendix F

calculate_wer

```
#!/usr/bin/env python
"""
calculate_wer

Calculate the WER of a file output by `do_whisper` and either print the result
to stdout or write in JSON format to some output location.

Copyright Adam Spencer, 03/2023
"""
__author__ = 'Adam Spencer'

import argparse
import json
import re
from pathlib import Path

import jiwer as jw
from whisper.normalizers import EnglishTextNormalizer

# This normalises strings for better comparison
normalise = EnglishTextNormalizer()

# Used to ensure only utterance entries (numerical idx) in the transcript file
# are parsed.
only_digits = re.compile(r'^\d+$')


def get_avg_logprob(data: dict) -> float:
    """
```

```
    Get an average value for the log posterior probability.

    :param data: Data for single utterance.
    :returns: Average log posterior probability / number of utterance segments
    """
    n_segs = 0.0
    total_logprob = 0.0
    for seg in data['whisper']['segments']:
        n_segs += 1.0
        total_logprob += seg['avg_logprob']
    if n_segs <= 0.0:
        return None
    return total_logprob / n_segs


def get_file_wer(transcript_file: Path, file_measures: bool
                 ) -> (dict, list, list):
    """
    Get WER from a transcript JSON file as returned by do_whisper.

    :param transcript_file: Path to transcript file.
    :param file_measures: True to calculate measures for entire file and write
        to the first entry in the output.
    :returns: Dictionary representation of file with added WER, along with lists
        containing hypotheses and references.
    """
    with open(transcript_file) as file:
        data_dict = json.load(file)

    hypotheses = []
    references = []
    n_utterances = 0
    for idx, data in data_dict.items():
        if not only_digits.match(idx):
            continue
        ref = normalise(data['transcript'])
        hyp = normalise(data['whisper']['text'])
        if ref == '' or hyp == '':
            continue
        else:
            hypotheses.append(hyp)
```

```
            references.append(ref)
            wer = jw.wer(ref, hyp)
        data_dict[idx]['wer'] = wer
        data_dict[idx]['avg_logprob'] = get_avg_logprob(data)
        n_utterances += 1

    if file_measures:  # Calculate WER for entire file
        file_wer = jw.wer(hypotheses, references)
        data_dict['0']['file_measures'] = {'wer': file_wer}

    return data_dict, hypotheses, references


def write_out(wer_dict: dict, /, out_path: Path = None,
              stdout: bool = False) -> None:
    """
    Write the WER output either to a specified file or to stdout.
    Output is in JSON format.

    :param wer_dict: WER dictionary as output by `get_file_wer()`
    :param out_path: Path to write output to.
    :param stdout: Write output to stdout rather than to a file.
    """
    if out_path is None and not stdout:
        raise ValueError(
            'Must either supply a path or choose to print to stdout!')
    if stdout:
        print(json.dumps(wer_dict, indent=2))
    else:
        with open(out_path, 'w') as f:
            json.dump(wer_dict, f, indent=2)


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('in_file', help='Input file')
    parser.add_argument('--output-dir', '-o',
                        help='Output dir, if none is supplied then result is \
                       printed to stdout')
    parser.add_argument('--file-measures', '-m', action='store_true',
                        help='Calculate measures for entire input files and \
```

```
                            store in the first entry of the output file.')
    parser.add_argument('--corpus-measures', '-M', action='store_true',
                        help='Caclulate measures for the entire corpus, output \
                            is written to a file named "measures.json" in \
                            the output dir')
    parser.add_argument('--verbose', '-v', action='store_true',
                        help='Activate verbose output')
    args = parser.parse_args()

    file_measures = args.file_measures
    corpus_measures = args.corpus_measures
    verbose = args.verbose
    in_file_path = Path(args.in_file)
    if save_output := args.output_dir is not None:
        out_path = Path(args.output_dir)
        out_path.mkdir(exist_ok=True)

    if corpus_measures:
        corpus_hyps = []
        corpus_refs = []

    if in_file_path.is_dir():
        if not save_output:
            raise ValueError('Must supply an output dir when input is dir!')
        files = [f for f in in_file_path.iterdir()]
        n_files = len(files)
        for idx, file in enumerate(files):
            if not file.name.endswith('.json'):
                continue
            if verbose:
                print(f'Progress : {idx:3} / {n_files:3}', end='\r')
            wer_dict, hyps, refs = get_file_wer(file, file_measures)
            if corpus_measures:
                corpus_hyps.extend(hyps)
                corpus_refs.extend(refs)
            write_path = out_path / file.name.replace('.json', '_wer.json')
            write_out(wer_dict, out_path=write_path)

    else:
        wer_dict, hyps, refs = get_file_wer(in_file_path, file_measures)
        if corpus_measures:
```

```
            corpus_hyps.extend(hyps)
            corpus_refs.extend(refs)
        if save_output:
            write_path = (out_path /
                          in_file_path.name.replace('.json', '_wer.json'))
            write_out(wer_dict, out_path=write_path)
        else:
            write_out(wer_dict, stdout=True)

    if corpus_measures:
        corpus_wer = jw.wer(corpus_refs, corpus_hyps)
        corpus_measures_dict = {'wer': corpus_wer}
        write_path = out_path / 'corpus_measures.json'
        write_out(corpus_measures_dict,
                  out_path=write_path, stdout=(not save_output))


if __name__ == '__main__':
    main()
```

Appendix G

asr_app.py

```
import cmd
import shutil
from pathlib import Path

from audio_data_link import AudioDataLinker
from textual.app import App, ComposeResult
from textual.coordinate import Coordinate
from textual.widgets import DataTable, Footer

ROW_HEIGHT = 4


class TranscriptionApp(App):
    """
    A Textual app to demonstrate semi-automatic transcription.
    """
    BINDINGS = [
        ('p', 'play_sound', 'Play Audio'),
        ('s', 'sort_id', 'Sort ID'),
        ('w', 'sort_wer', 'Sort WER'),
        ('u', 'sort_utt_conf', 'Sort Utterance Conf'),
        ('>', 'sort_max_conf', 'Sort Max Conf.'),
        ('<', 'sort_min_conf', 'Sort Min Conf.'),
        ('a', 'sort_prob', 'Sort Av. Log Prob.'),
    ]

    def compose(self) -> ComposeResult:
        """
        Create child widgets for the app.
```

```
        """
        yield DataTable(fixed_columns=1, zebra_stripes=True)
        yield Footer()

    def on_mount(self) -> None:
        """This is run upon starting the App."""
        self.current_sort_col = 'ID'
        self.reversed_sort = False
        table = self.query_one(DataTable)
        rows = iter(self.data_in)
        for col_name in next(rows):
            table.add_column(col_name, key=col_name)
        for row in rows:
            table.add_row(*row, height=ROW_HEIGHT)

    def populate_data(self, data_linker: AudioDataLinker):
        """Populate table data."""
        self.data_in = data_linker.data_for_table()
        self.data_linker = data_linker

    def action_play_sound(self):
        """Play the utterance."""
        if self.selected_idx is not None:
            self.data_linker.play_audio(self.selected_idx)

    def sort_fn(self, col: str):
        """Function to sort a given column."""
        rev = False
        if self.current_sort_col == col and not self.reversed_sort:
            rev = True
        self.current_sort_col = col
        self.reversed_sort = rev
        dt = self.query_one(DataTable)
        dt.sort(col, reverse=rev)

    def action_sort_id(self):
        self.sort_fn('ID')

    def action_sort_wer(self):
        self.sort_fn('WER')
```

```
    def action_sort_prob(self):
        if not self.data_linker.confidence_mode:
            self.sort_fn('Average Log Probability')

    def action_sort_utt_conf(self):
        if self.data_linker.confidence_mode:
            self.sort_fn('Utterance Confidence')

    def action_sort_max_conf(self):
        if self.data_linker.confidence_mode:
            self.sort_fn('Max Conf.')

    def action_sort_min_conf(self):
        if self.data_linker.confidence_mode:
            self.sort_fn('Min Conf.')

    def on_data_table_cell_highlighted(self, message: DataTable.CellHighlighted):
        """Keeps track of current highlighted cell to enable playing of sound."""
        row_num = message.coordinate.row
        idx_coord = Coordinate(row_num, 0)
        self.selected_idx = self.query_one(DataTable).get_cell_at(idx_coord)


def launcher() -> (Path, Path, bool, bool, bool):
    """
    Launcher for my ASR App!

    In order for this to work, asr_app.py must be run from the `asr-app/src`
    directory.

    It will open up a list of conversatiosn to choose from, take a user's input
    as a number corresponding to a conversation and then the app will launch.

    :returns: something...
    """
    print("Would you like to use data with confidence scores? [y/n]")
    use_confidence_inp = input("\n-> ")
    use_confidence = False
    text_highlight = False
    text_blanking = False
    blanking_threshold = 0.0
```

```
    if use_confidence_inp.lower() == 'y':
        use_confidence = True
        print('\nWould you like to use text blanking, highlighting, or neither?'
              + '[b/h/n]\n')
        conf_opt = input('-> ').lower()
        if conf_opt == 'b':
            text_blanking = True
            blanking_threshold = float(input('Threshold : '))
        elif conf_opt == 'h':
            text_highlight = True

    data_path = Path('../data')
    if use_confidence:
        asr_data_path = data_path / 'confidence-data'
    else:
        asr_data_path = data_path / 'asr-out'
    audio_dirs_list = sorted(
        [i for i in (data_path / 'audio').iterdir() if i.is_dir()])

    # Present the conversations by code and allow the user to choose
    convos = [f'{idx:2} : {d.name} ' for idx, d in enumerate(audio_dirs_list)]
    cli = cmd.Cmd()
    print('\nChoose a conversation:\n')
    cli.columnize(convos, displaywidth=shutil.get_terminal_size().columns)
    chosen_audio = audio_dirs_list[int(input('\n-> '))]

    # Find the data file corresponding to the chosen conversation
    chosen_data = None
    for f in asr_data_path.iterdir():
        if chosen_audio.name in f.name:
            chosen_data = f
    if chosen_data:
        return (chosen_data, chosen_audio, use_confidence,
                text_blanking, blanking_threshold, text_highlight)
    raise IndexError('Can\'t find data for selected conversation')


if __name__ == "__main__":
    (data_file, audio_dir, confidence_mode, text_blanking,
     blanking_threshold, text_highlight) = launcher()
    data_linker = AudioDataLinker(
```

```
        audio_dir, data_file, confidence_mode, text_blanking, blanking_threshold,
        text_highlight)
    app = TranscriptionApp()
    app.populate_data(data_linker)
    app.run()
```

Appendix H

audio_data_link.py

```
import json
import sys
import textwrap
from pathlib import Path

import sounddevice as sd
import soundfile as sf
from rich.color import Color
from rich.style import Style
from rich.text import Text

WRAP_WIDTH = 40
BLOCK = '\u2588'  # Unicode block character for blanking out words


class AudioDataLinker:
    """
    Object to aid linking audio and transcript data
    """

    def __init__(self, audio_dir: Path, data_file: Path, confidence_mode: bool,
                 text_blanking: bool = False, blanking_threshold: float = 0.5,
                 text_highlight: bool = False) -> None:
        """
        Create a new AudioDataLinker.

        :param audio_dir: Directory within which audio files are stored.
        :param data_file: File containing transcription data.
        :param confidence_mode: True if confidence scores are being used.
```

```
        :param text_blanking: True to enable text blanking mode.
        :param text_highlight: True to enable text highlighting based on confidence.
        """
        with open(data_file) as f:
            self.data = json.load(f)
        self.audio_dir = audio_dir
        self.confidence_mode = confidence_mode
        if text_blanking or text_highlight:
            if text_blanking and text_highlight:
                print('Error! Can\'t blank and highlight at the same time :<')
                sys.exit(1)
        self.text_blanking = text_blanking
        self.blanking_threshold = blanking_threshold
        self.text_highlight = text_highlight
        self.row_data = self.init_row_data()

    def play_audio(self, idx: int) -> bool:
        """
        Play audio correlated with an utterance.

        :param idx: Index of utterance.
        :returns: True if audio file exists, false otherwise.
        """
        audio_file = self.audio_dir / f'{idx:03d}.wav'
        if not audio_file.exists():
            return False
        sd.play(*sf.read(audio_file))

    def init_row_data(self) -> list:
        """
        Get data in format required to be printed in DataTable object.

        :returns: List containing table rows.
        """
        row_data = []
        for idx, utterance in self.data.items():
            kwgs = dict()
            kwgs['idx'] = idx
            kwgs['hyp'] = wrap_and_format(utterance['whisper']['text'])
            kwgs['ref'] = wrap_and_format(utterance['transcript'])
            kwgs['wer'] = utterance['wer']
```

```
            if self.confidence_mode:
                if conf_scoring := utterance.get('confidence_scoring'):
                    kwgs['conf_scoring'] = conf_scoring
                else:
                    continue
            else:
                kwgs['avg_logprob'] = utterance['avg_logprob']
            row = TableRow(self.confidence_mode,
                           self.text_blanking, self.text_highlight,
                           self.blanking_threshold, **kwgs)
            row_data.append(row)
        return row_data

    def data_for_table(self) -> list:
        table_rows = [row.for_table() for row in self.row_data]
        table_rows.insert(0, self.row_data[0].get_header())
        return table_rows


class TableRow:
    """
    Object to represent table row.

    Using OOP simplifies text manipulation in table.
    """

    def __init__(self, confidence_mode: bool, text_blanking: bool = False,
                 text_highlight: bool = False, blanking_threshold: float = 0.5,
                 **kwargs):
        """
        Create a new TableRow.

        :param confidence_mode: True if using confidence mode.
        :param text_blanking: True to enable text blanking.
        :param text_highlight: True to enable text highlighting.
        :param blanking_threshold: Set the confidence threshold to blank text.
        :param kwargs: Data to insert into table.
            `kwargs` allows different options based on confidence mode.
        """
        self.confidence_mode = confidence_mode
        self.highlight_mode = text_highlight
```

```
            self.idx = int(kwargs['idx'])
            self.hyp = kwargs['hyp']
            self.ref = kwargs['ref']
            self.wer = kwargs['wer']
            if confidence_mode:
                conf_scoring = kwargs['conf_scoring']
                scores = conf_scoring['confidence_scores']
                self.token_conf_pair = list(zip(
                    conf_scoring['words'].split(), scores))
                if len(scores) > 0:
                    self.max_conf = max(scores)
                    self.min_conf = min(scores)
                else:
                    self.max_conf = 0.0
                    self.min_conf = 0.0
                self.utt_conf = conf_scoring['utterance_confidence']
            else:
                self.avg_logprob = kwargs['avg_logprob']

            if text_blanking:
                self.hyp_to_print = wrap_and_format(
                    self.text_blanking(blanking_threshold))
            elif text_highlight:
                self.hyp_to_print = self.text_highlighting()
            else:
                self.hyp_to_print = wrap_and_format(self.hyp)

        def for_table(self):
            """Get row data in correct format for insertion into table."""
            if self.confidence_mode:
                return (int(self.idx), self.hyp_to_print, self.ref,
                        float(self.utt_conf), float(self.max_conf),
                        float(self.min_conf), float(self.wer))
            else:
                (int(self.idx), self.hyp_to_print, self.ref, float(
                    self.avg_logprob), float(self.wer))

        def get_header(self):
            """Get header row for table."""
            if self.confidence_mode:
                return ['ID', 'Hypothesis', 'Reference',
```

```
                    'Utterance Confidence', 'Max Conf.', 'Min Conf.', 'WER']
        else:
            return ['ID', 'Hypothesis', 'Reference',
                    'Average Log Probability', 'WER']

    def text_highlighting(self) -> Text:
        """
        Highlight text based on confidence measure.

        :returns: Text with colour between red and green.
        """
        text_list = []
        for word, conf in self.token_conf_pair:
            text_style = compute_colour(conf)
            # Highlight text and replace underscores with space
            text_obj = Text(word.replace('_', ' '), style=text_style)
            text_list.append(text_obj)
        # Wrap text to be printed properly
        lines = Text(' ').join(text_list).wrap(None, WRAP_WIDTH)
        return Text('\n').join(lines)

    def text_blanking(self, threshold: float) -> str:
        """Blank out text based on confidence threshold."""
        text_list = []
        for word, conf in self.token_conf_pair:
            if conf < threshold:
                # Removes underscores when printing blocks
                for subword in word.split('_'):
                    text_list.append(len(subword) * BLOCK)
            else:
                text_list.append(word)
        return ' '.join(text_list)


def compute_colour(conf: float) -> Style:
    """
    Compute a colour between red and green for highlighting text.

    :param conf: Confidence score between 0 and 1.
    :returns: Style to highlight text with.
    """
```

```
    r1, g1, b1 = (255, 0, 0)  # red
    r2, g2, b2 = (0, 255, 0)  # green
    rn, gn, bn = (
        r1 + conf * (r2 - r1),
        g1 + conf * (g2 - g1),
        b1 + conf * (b2 - b1)
    )
    return Style(color=Color.from_rgb(rn, gn, bn))


def wrap_and_format(string: str) -> str:
    """Apply text wrapping to a string and remove underscores."""
    string_no_uscore = string.replace('_', ' ')
    return '\n'.join(textwrap.wrap(string_no_uscore, width=WRAP_WIDTH))
```