UTRECHT UNIVERSITY

Department of Information and Computing Science

Applied Data Science master thesis

Improving the effectiveness of different Automatic Speech Recognition models with hyperparameter tuning

First examiner:
Ioana Karnstedt-Hulpus

Candidate:
Christian Acosta

Second examiner:
Vahid Shahrivari Joghan

In cooperation with:
National Police Corps

July 14, 2023

Abstract

Automatic Speech Recognition (ASR) models have shown great progress in recent years. Whisper is one of the latest models, showing state-of-the-art performance on a broad range of unseen datasets. This makes it a useful model for a broad range of applications, such as converting audio files into text transcripts. Detectives of the National Police Corps have a large amount of audio data to process for their investigations. Manual processing is tedious and resource intensive. Whisper can be a useful tool for speeding up investigations and alleviating the workload. While Whisper performs well out-of-the-box, its performance can still be further improved. Through the method of hyperparameter tuning and comparing different implementations of Whisper, the processing time, memory usage, and accuracy have been optimized. Firstly, we show that reducing computational precision improved the performance in all models tested. Secondly, reducing beam size to a more greedy strategy reduced processing time and memory usage with minimal influence on accuracy. Thirdly, larger batch sizes decreased processing time and increased accuracy, but also increased memory usage. Lastly, implementing Voice Activity Detection increased accuracy and decreased processing time without increasing memory usage. We conclude that Faster-Whisper is the overall best performing model for the current use-case. It has the best trade-off between processing time, memory usage, and accuracy. Consequently, this allows for the greatest transcription throughput when multiple instances of the model are used in parallel.

Contents

1. Introduction

1.1 Motivation and context

The National Police Corps (Korps Nationale Politie) possesses a vast amount of audio data. Data from seized devices and data storage mediums, such as voice notes from (encrypted) smartphones, must be processed before being used as evidence in an investigation. Manual analysis by listening to the audio is a tedious and highly labor-intensive process, which robs detectives of precious time that could have been spent on important tasks that can not be automated. To help alleviate the workload of detectives, a Speech to Text model can be used to develop a software tool that can automatically transcribe the audio data into a searchable and readable text format. This allows a detective to find specific audio files more quickly, listen to the relevant sections of the audio and transcribe it for use as evidence in their investigation. Due to privacy, confidentiality, and legality concerns, everything has to be hosted on-premises instead of using a cloud-based service or API. As such, there is a great need for a high-performing model that can run efficiently on relatively limited hardware capacity. The scope of this thesis is to investigate what models are available, which models perform best, what parameters return peak performance in terms of speed and accuracy, and what pre-processing can be performed in order to provide the all-round best performing model.

1.2 Literature review

1.2.1 History

Automatic Speech Recognition (ASR), also known as Speech to Text (STT/S2T) is a group of technologies and methods that enables a computer to recognize

and transcribe spoken language into text. Insights from computer science and computational linguistics are combined to realize ASR. In 1952, Bell Labs created the Automatic Digit Recognizer (AUDREY), a rudimentary machine by modern standards, but nonetheless considered to be the birth of ASR [1]. Development progressed throughout the following decades, with groups such as IBM, Carnegie Mellon [2], [3], MIT, Dragon[4], and DARPA getting involved in ASR research. A breakthrough occurred in 1987, when SPHINX was developed. This was the first system that was speaker-independent, as prior systems had to be trained to each specific speaker [5].

1.2.2 Definitions

1.2.2.1 Word Error Rate

In order to quantitatively evaluate the performance of ASR models, we require performance metrics to be defined. On the most common and important metrics is Word Error Rate (WER). WER is essentially the Levenshtein distance, but adapted to function at a word level instead of a phoneme level. Put simply, the WER is the number of edit operations, i.e. substitutions, deletions or insertions, required to change the hypothesis string into the reference string. WER will always be positive, due to the fact that addition and division operations are performed on non-negative real numbers. If the reference and hypothesis strings are identical, the WER will be exactly 0. Mathematically, WER is defined as follows:

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

where
S denotes the number of substitutions
D denotes the number of deletions
I denotes the number of insertions
N denotes the number of words in the reference.

1.2.2.2 Memory

Memory usage is another performance metric that will be used to optimize the performance, more specifically the efficiency of the ASR model. A Graphics Processing Unit (GPU) contains some amount of Video Random Access Memory (VRAM). VRAM is used as a buffer to store a model, especially the model parameters and input, in order to perform the calculations required during training and inference. The amount of memory available in a GPU is very often the bottleneck in terms of hardware requirements for running a specific model. This issue is further worsened by the fact that VRAM size is fixed in a GPU and can not be upgraded by the user. The solution is to employ a more capable, but expensive GPU that has more VRAM installed, or multiple GPUs and run them as a cluster (also expensive).

1.2.2.3 Real Time Factor

Additionally, Real Time Factor (RTF) is one more metric related to model performance, more specifically the speed of transcription. RTF describes the ratio between the length, i.e. duration of the input audio, and the processing time required to transcribe the input audio. The closer to 0 the RTF is, the faster the model is at transcribing audio. Mathematically, RTF is defined as follows:

$$RTF = \frac{P}{I}$$

where
P denotes the processing time of the given audio input
I denotes the duration of the given audio input.

1.2.2.4 Spectrograms

Spectrograms are a visual representation of a signal frequency spectrum over time. They are a good way to visually represent audio data, as can be seen in Figure 1.1.

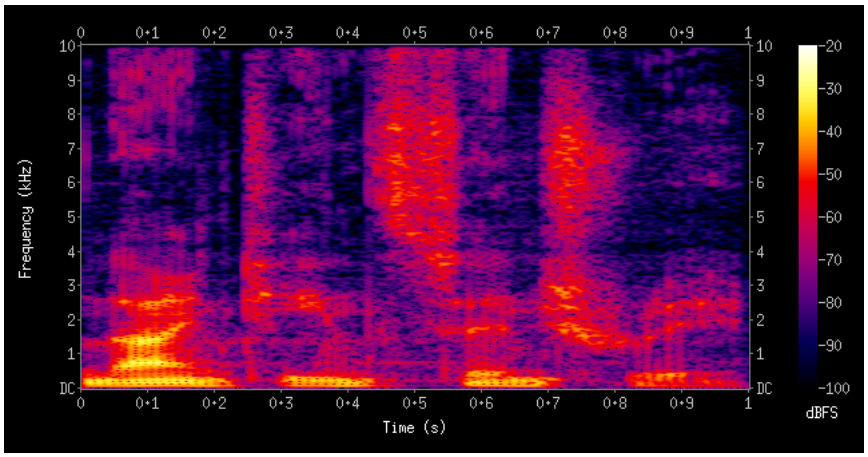

Figure 1.1: Example of a spectrogram[1]

In a regular spectrogram, frequencies are plotted on a linear scale. However, research has shown that humans perceive audio frequencies on a non-linear scale [6], [7]. One proposed solution to this is the melody (mel) scale.

1.2.2.5 Mel scale

The mel scale is based on the perception of pitch by human listeners, in other words, frequencies perceived to be equal distance from one another [8]. The mel scale mimics how the human hearing works, with high precision in the lower-end of the frequency band, where speech is located, and low precision in the higher-end of the frequency band. There are various mathematical definitions for the mel scale, but generally the formula contains a linear/logarithmic component. The differences between most formulas is the corner/break frequency. The mel scale will follow the Hertz scale linearly up to and including the corner frequency. Above the corner frequency the mel scale will increase logarithmically. This linear/logarithmic characteristic can be seen more clearly in Figure 1.2.

Usually, formulas set the corner frequency to 1000 Hz, which will then equal 1000 mels. Corner frequencies of 625 Hz [9] or 700 Hz [10] have also been proposed. For the purpose of ASR, the mel scale formula is most com-

[1]https://commons.wikimedia.org/w/index.php?curid=5544473
[2]https://commons.wikimedia.org/wiki/File:Mel_scale.PNG

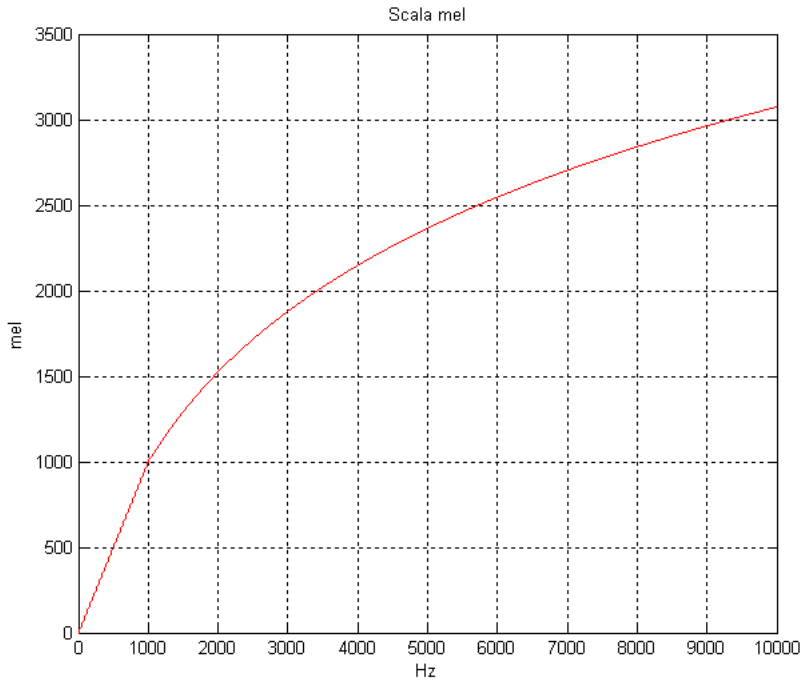

Figure 1.2: Plot of the mel scale with a corner frequency of 1000 Hz[2]

monly expressed as follows:

$$m = \frac{1000}{log2}(1 + \frac{f}{1000})$$

with a corner frequency of 1000 Hz [11]. A mel spectrogram is a spectrogram where the frequencies are plotted based on the mel scale, in contrast with the linear frequency usually seen on a 'normal' spectrogram. The advantage of the mel scale is that it is designed to mimic how humans perceive audio. Accordingly, a log-mel spectrogram has the mel scale transformed to a logarithmic scale instead of linear, as can be seen in Figure 1.3

1.2.2.6 Convolutional Neural Networks

Convolutional neural networks (CNN) are artificial neural networks that make use of a convolution function in at least one of its layers [12], [13]. The

[3]https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0

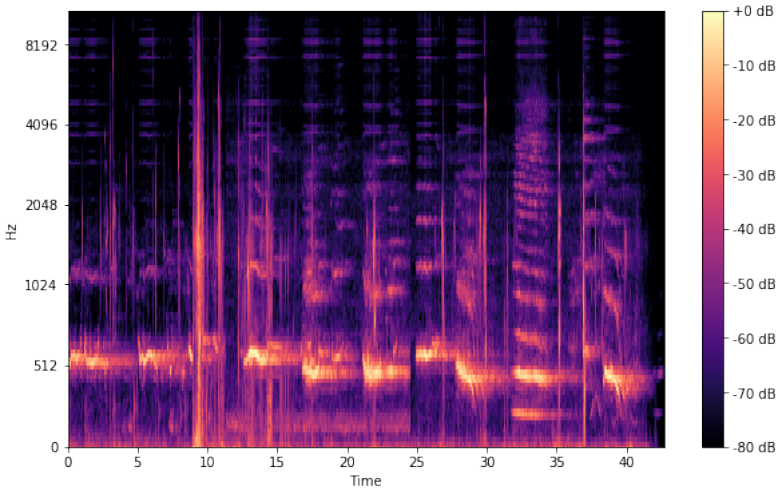

Figure 1.3: An example of a log-mel spectrogram[3]

convolutional layer applies filters to an image to extract certain features. A CNN is particularly well-suited to tasks with images, being used in many different processing and recognition tasks. CNNs are also useful for dimensionality reduction, which in turn reduces the computational complexity of processing a spectrogram.

1.2.2.7 Attention

Attention is a technique in artificial neural networks (ANN) that is intended to imitate (human) attention as understood in the domains of neuroscience and psychology. In an ANN, attention will enhance certain parts of the input sequence, while simultaneously diminishing other parts of the sequence. The idea is that the ANN should allocate more focus to a certain, important parts of the input, despite the fact that it may be a small portion of the input. In essence, certain parts of the input are 'weighted' more than others. Using gradient descent, an ANN learns that a section of the input being more important than another depends on context. This learned attention representation of which sections to focus, or give attention to, is known as an attention mask. The advantage of attention for speech tasks is that a model can flexibly utilize the most relevant parts of an arbitrary input sentence.

1.2.2.8 Self-attention

Self-attention is variant of attention. Where attention links an input sequence with an output sequence, self-attention focuses on a single input sequence. Self-attention allows an ANN to let a sequence learn information about itself.

1.2.2.9 Cross-attention

Cross-attention is yet another take on attention as a concept. Cross-attention uses the attention mask from one modality to highlight features extracted from another modality. This is in contrast to self-attention, where a modality uses its own attention mask to highlight its own features.

1.2.2.10 Transformers

Recent works with transformer architecture have achieved good performance on many tasks, including Speech to Text. In this thesis, we will investigate Speech to Text models with transformer architecture. Transformers are neural networks that implement the concept of attention and self-attention, allowing the network to process a complete input sequence at once instead of sequentially, unlike a Recurrent Neural Network (RNN) [14]. The attention mechanism provides context for any arbitrary point in the input sequence. This structure allows a transformer model to run at a much higher level of parallelization, which in turn greatly reduces training times. Brought to life in the 2017 paper titled "Attention is All You Need" [15], transformers are now considered as "foundation models" [16] due to the paradigm shift they have brought to the field of artificial intelligence. Transformers are useful due to their ability to generalize to many different applications and data types, be it computer vision (images/videos) or natural language processing (text/audio).

1.2.3 Related work

The ASR field has made remarkable progress in recent years. With the development of the wav2vec 2.0 model, self-supervised pre-training on unla-

beled speech data has been made possible [17]. Following the pre-training, a fine-tuning phase with labeled data is used to optimize the self-learned representations for character or phoneme predictions. The wav2vec model outperforms the state-of-the-art semi-supervised model, while simultaneously using 100 times less labeled data [17]. While the wav2vec model performs well, it has the limitation of needing to be fine-tuned to a specific dataset. This hampers the performance of the model on data not seen and fine-tuned to.

Researchers from OpenAI have made an attempt to counteract the weaknesses of the wav2vec model by developing a weakly-supervised transformer model named Whisper [18]. An overview of the model architecture is shown in Figure 1.4. The input for the model is an audio file, segmented into 30-second chunks, which are accordingly transformed into log-mel spectrograms. Afterwards, two one-dimensional convolutional layers with a Gaussian Error Linear Unit (GELU) [19] activation function are used to compress the spectrograms to a lower spatial size, which greatly reduces the computational complexity. Hereafter, sinusoidal position embeddings are added the output of the previous step and passed on to the encoder transformer blocks. Using the attention mechanism and encoded information, the decoder transformer blocks predict which word follows the current word being transcribed. The overall model architecture is based on the 'classic' encoder-decoder architecture [15]. The encoder-decoder architecture consists of a stack of 6 identical encoder layers, followed by a stack of 6 identical decoder layers. The final output of the model is a text transcript of the input audio, including timestamps.

This model requires no dataset specific fine-tuning to achieve good results in terms of WER, which we discussed in the definitions section, and accordingly improving the zero-shot performance. In essence, the Whisper model works as-intended "out of the box" on any data it is given. This performance has been obtained by massively scaling up the quantity of training data, on the order of magnitudes, compared to previous efforts. The typical supervised model uses around 1.000 hours of training data, which is significantly less than what is possible with unsupervised models [18].

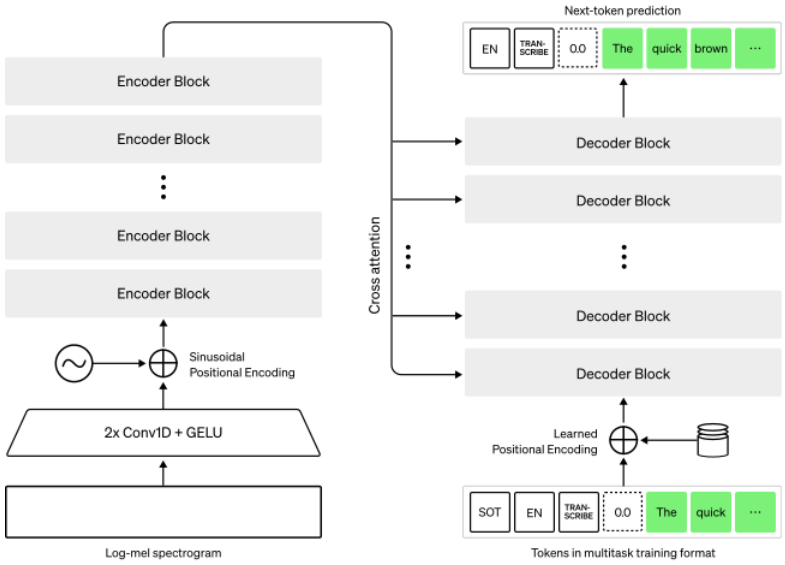

Figure 1.4: Overview of the Whisper model architecture[4]

Whisper uses around 680.000 hours of training data, which is much closer to the scale of unsupervised models. The Whisper authors suggest that "simple scaling of weakly supervised pre-training has been underappreciated so far for speech recognition" [18]. Their work has shown that trading off quantity over quality can be the right choice, and that it increases the generalization and robustness of the model. Another advantage of the Whisper model is that is also trained with languages other than English, with approximately 117.000 hours spread between 96 languages. As impressive as this all is, there are still some weaknesses. The largest Whisper model has approximately 1550 million parameters, and consequently is highly resource-intensive and relatively slow during transcription. This is further worsened by the characteristic that transcription process of the audio runs sequentially in the order that the it is spoken. Another downside is that timestamps tend to only be accurate on an utterance or phrase level, not a word or phoneme level. The sequential nature of the transcription process means that if the timestamp of the initial chunk is inaccurate, that following chunks will drift further away from the correct timing. In certain scenarios the Whisper model can exhibit hallucinations, i.e. putting words

[4]https://openai.com/research/whisper

in the transcript that were not present in the audio file.

While the Whisper model is highly impressive, there are still weaknesses and areas where it can be improved upon. Multiple variants and re-implementations of Whisper have been developed since the original publishing, and are still being improved upon.

1.2.4 Whisper variants

The implementation of Whisper available in the Hugging Face Transformers library[5] uses a batching algorithm for the transcription process, where audio chunks are grouped together into batches and processed in parallel. The addition of batching alone has shown great transcription speed increases in benchmarks, with 8 times gain being observed.

faster-whisper[6] is an implementation that uses the CTranslate2[7] backend , which is a library optimized for fast and efficient inference with Transformer models . This implementation runs up to 4 times faster than the original OpenAI implementation, while at the same time reducing memory consumption. Furthermore, Silero Voice Activity Detection (VAD)[8] is used to remove parts of the audio that do not contain speech. This step should help increase transcription speed, as the audio chunks to be transcribed will contain as little silence as possible, which in turn can reduce hallucinations as well.

The third variation of the Whisper model is WhisperJAX[9]. This implementation of Whisper is modified to run on the JAX framework, whereas the original OpenAI implementation runs on the PyTorch framework. A batching algorithm for transcription is also used in the WhisperJAX implementation. In addition to support for the usual GPU acceleration, Tensor Processing Unit (TPU) acceleration is also possible. The WhisperJAX developers claim that it is currently the fastest Whisper implementation, with it

[5]https://github.com/huggingface/transformers
[6]https://github.com/guillaumekln/faster-whisper
[7]https://github.com/OpenNMT/CTranslate2
[8]https://github.com/snakers4/silero-vad
[9]https://github.com/sanchit-gandhi/whisper-jax

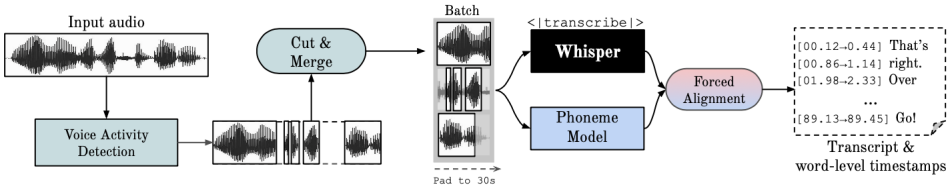

Figure 1.5: Overview of the WhisperX model architecture[11]

being able to transcribe up to 70 times faster than the original Whisper implementation when using TPU acceleration, and more than 13 times faster when using GPU acceleration.

One of the latest attempts at resolving the issues from the original Whisper implementation is the WhisperX[10] model [20]. This model combines the CTranslate2 back-end from the faster-whisper implementation with the wav2vec 2.0 model. An overview of the model architecture can be seen in Figure 1.5

Using forced alignment, referring to the process where transcripts in the native spelling of the target language are aligned to audio data, transcripts can be segmented on a phoneme level. This allows one to obtain highly accurate (to the order of milliseconds) timestamps on either a word or phoneme level. This model also supports VAD and speaker diarization out-of-the-box using pyannote-audio[12]. In order to improve batching performance and reduce hallucinations, VAD pre-processing is added in WhisperX. Further adaptations to reduce hallucinations are present in the form of parameter modifications in Whisper's transcribe function, where the –condition_on_prev_text parameter is modified. Conditioning on previous text is a process where the output of the previous chunk is used as context for the current chunk. The idea is that this will improve consistency of the transcription between chunks and improve the 'flow' of the text. In Whisper, this parameter is True by default. However, setting it to False, which is the case in WhisperX, reduces hallucinations and occurrences of the model getting stuck in a loop of repetitive text.

[10]https://github.com/m-bain/whisperX
[11]https://arxiv.org/abs/2303.00747
[12]https://github.com/pyannote/pyannote-audio

1.3 Research question

Considering that the National Police Corps has a large and burgeoning amount of audio data that needs to be processed, it follows that there is a great motivation for the realisation of a faster Speech to Text model. The literature review suggest that there are variants of the Whisper model that improve on the original implementation on all aspects of performance. Furthermore, we believe that these variants can be optimized even further by tweaking their parameters.

With the knowledge from related works, and taking the research motivation and context into account, we arrive at the following research question:

What optimizations can be made to improve the transcription performance and efficiency of Whisper and its variants?

From the main research question we derive three subquestions, all of which play a role in answering the main research question. Each subquestion concerns a different aspect of the performance of the Whisper model and its variants that will be investigated through multiple experiments, and are divided as follows:
Subquestion 1: What is the effect of batch size on memory usage of WhisperX?
Subquestion 2: How can the RTF be decreased for all Whisper variants?
Subquestion 3: How does VAD affect Faster-Whisper's WER?

2. Data

2.1 Description of the data

The dataset used as reference for analysis and comparisons of model performance is the Corpus Gesproken Nederlands (CGN) dataset [21]. In total, CGN contains roughly 900 hours, or almost 9 million words, of modern Dutch, spoken by both Dutch and Flemish people. The the total file size for the dataset is 129,1 GB in uncompressed form. The dataset consists of 14 components, ranging from component A to component O. All audio files are in WAVE (.wav) format. An overview of the components and the type of audio they contain is provided in Table 2.1.

Component	Type of audio
A	Spontaneous conversations ('face-to-face')
B	Interviews with Dutch teachers
C	Telephone dialogues recorded at switchboard
D	Telephone dialogues recorded using mini disc recorder
E	Business negotiations
F	Interviews and discussion broadcast on radio and television
G	Discussion, debates, meetings (particularly political)
H	Lessons
I	Spontaneous comments (including sports) broadcast on radio and television
J	Current affairs sections and reports broadcast on radio and television
K	News bulletins broadcast on radio and television
L	Reflections and commentaries broadcast on radio and television
M	Masses, lectures, solemn speeches
N	Lectures, lectures, readings
O	Pre-read texts

Table 2.1: Overview of dataset components

2.2 Data exploration results

Initial exploratory analysis has been performed to gain a general overview of the contents, characteristics and descriptives of the CGN dataset. For each component, the number of audio files, mean duration of audio files

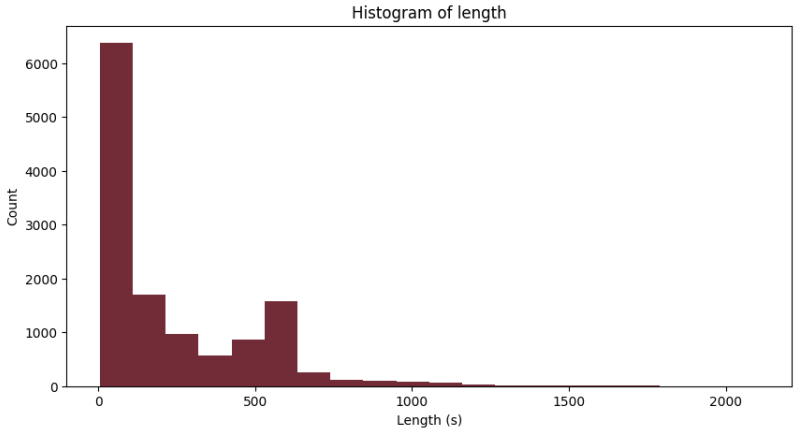

Figure 2.1: Histogram of the duration of audio files (seconds) in the CGN dataset

(minutes), total duration of audio files (minutes), and number of speakers are recorded. The findings from the exploratory analysis are show in Table 2.2. It can be noted that component K has noticeably different descriptives than the other components, with a very large number of files of relatively short duration. This can be more clearly observed in Figure 2.1.

Component	File count	Mean duration	Total duration	Speakers
A	1537	8.77	13478.93	2+
B	160	18.99	3037.72	2
C	649	8.65	5611.23	2
D	581	6.63	3851.12	2
E	67	9.57	640.95	2+
F	642	5.97	3829.73	2+
G	248	8.83	2189.35	2+
H	265	9.86	2613.40	2+
I	325	3.82	1240.49	1
J	506	2.06	1041.35	1+
K	5581	0.38	2136.61	1
L	365	2.42	881.03	1
M	16	8.14	130.20	1
N	78	12.01	936.78	1+
O	1761	3.54	6228.24	1

Table 2.2: Overview of file descriptives for each component

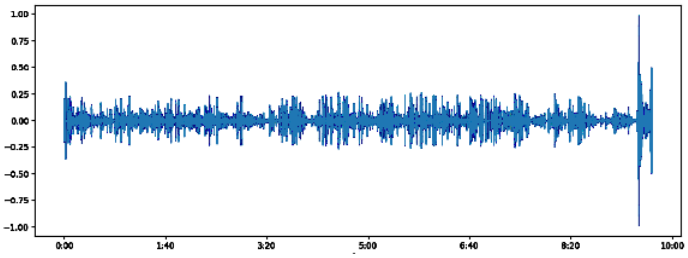

Figure 2.2: Waveform of the full-length audio sample

2.3 Preparation of the data

Data to be used for transcription by the Whisper model (or any of its variants) does not require any specific preparation to be performed. File types supported are m4a, mp3, webm, mp4, mpga, wav, and mpeg. Internally, Whisper uses ffmpeg to read the input audio files and resample to 16KHz. Next, the input audio is split into 30-second chunks. Subsequently, the audio is converted to a log-mel spectrogram, which we discuss in the next section.

2.3.1 Log-Mel spectrogram

This section demonstrates the computation and visualization of a log-mel spectrogram. Whisper uses a different approach for computing the Log-Mel spectrogram, namely by applying a Short-Time Fourier Transformation (STFT) on the 30-second audio chunks and afterwards using a mel filterbank matrix to project the STFT into a mel spectrogram.

Reference file in Appendix C is used for the following demonstration. Figure 2.2 shows a waveform of the audio file in its entirety, totaling roughly 10 minutes.

Next, Figure 2.3 shows the first 30-second audio chunk taken from the beginning of the file.

Afterwards, the audio chunk is converted into a 'regular' spectrogram, that is with the y-axis linearly displaying frequency as Hz. It can be noted that it is quite hard to see any meaningful audio activity in Figure 2.4, and thus little useful information can be derived.

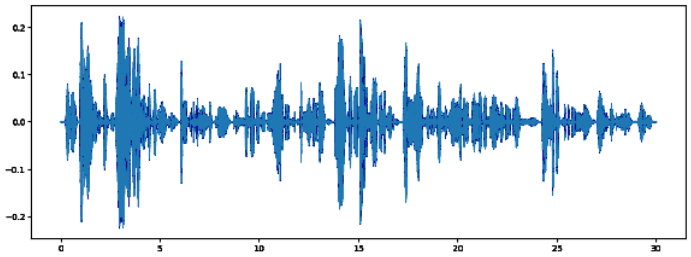

Figure 2.3: Waveform of the first 30-second chunk from the audio sample

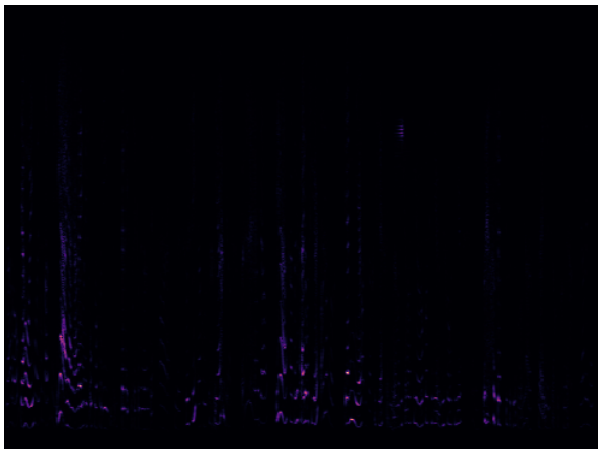

Figure 2.4: Frequency (Hz) spectrogram of the chunk

Converting the frequency spectrogram to display the mel scale, one can notice that more activity can be seen, thus more information can be derived from Figure 2.5.

Finally, the mel spectrogram is modified to show the mel spectrogram on a decibel (logarithmic) scale. One can observe much more meaningful visual activity in Figure 2.6, which is exactly why the log-mel transformation is an important step.

2.3.2 Choosing a suitable dataset subset

A subset of the CGN dataset, namely the component C directory, has been chosen to be used for experimentation. The component c directory has the most similar characteristics to the NFI-FRITS dataset, which is the dataset used internally by the National Police Corps for experimentation and validation purposes [22]. Both component C and NFI-FRITS consist of telephone conversations recorded at a switchboard. These similarities mean that component C can be considered as most representative for experimen-

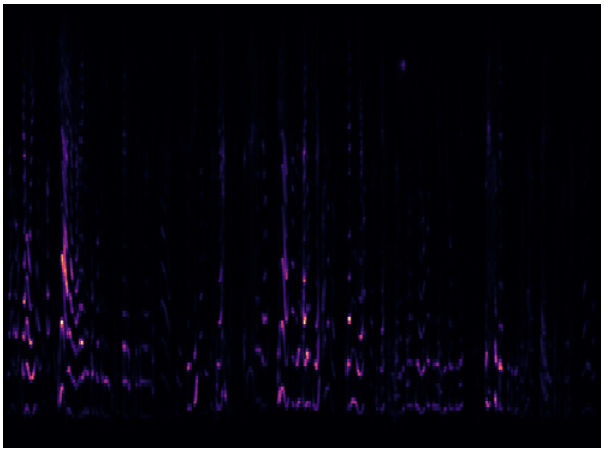

Figure 2.5: Mel spectrogram of the chunk

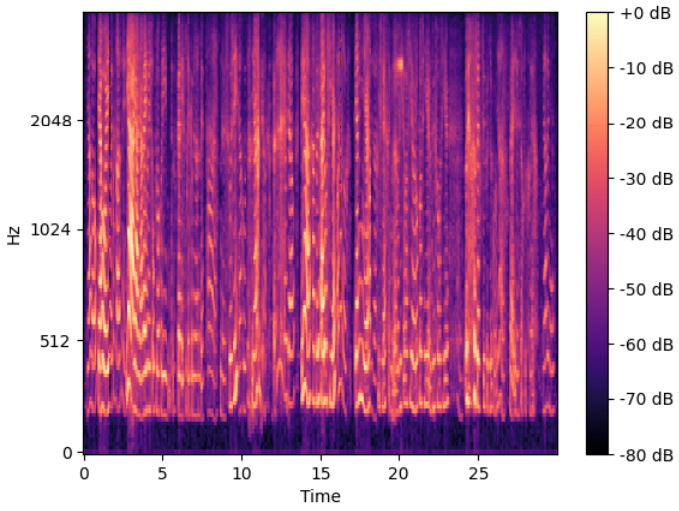

Figure 2.6: Log-mel spectrogram of the chunk

tation purposes. The similarity between datasets should give a higher likelihood of the findings from this report being applicable to the the National Police Corps' use-case and data.

Another important aspect to note is that samples are taken from component C. This is done to reduce the computational requirements and experiment running times to an acceptable and workable level. An approximately 5% random sample is taken from the NL subdirectory, which equals 18 files. As explained in Appendix B, we randomly select 18 files from a given directory and move them to a specified subdirectory. This subdirectory is used for further experimentation and remains fixed. The sampling is done without replacement, meaning the 5% sample is used as the development subset. A 10% sample from the remaining files of component C are used as the test subset later on, as to avoid overlap. Table 2.3 provides an overview of the number of files in each subset.

Subset	Number of files
Development	18
Test	36

Table 2.3: Overview of dataset subsets created

2.4 Ethical and legal considerations of the data

The project that developed the CGN dataset was funded by the Dutch and Flemish governments, and by the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO/Dutch Research Council). The rights to the CGN research and dataset are held by the Nederlandse Taalunie (Dutch Language Union). The corpus is licensed to be freely accessible for scientific usage. For the purposes of this thesis there are no legal objections.

3. Method

3.1 Overview

The general method is the comparison of different variants of the Whisper model and the tuning of parameters to find the best trade-off between performance metrics for the use-case presented in this thesis. Figure 3.1 provides a visual overview of the method.

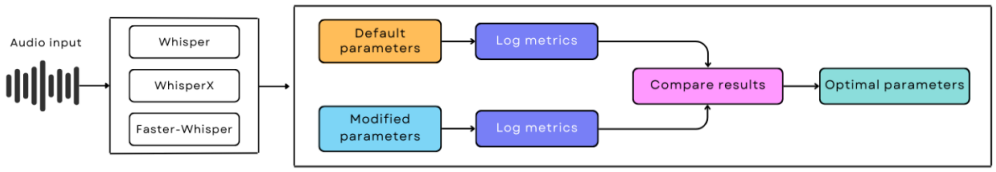

Figure 3.1: Visualization of the method

3.2 Variables

For each Whisper model variant, there are multiple parameters that can be specified for use during the model run. If not otherwise specified, default values will be used. The models provide many parameters, of which only a select portion will be used to tune the performance of the model. The selection of which parameters to use for tuning is based on the model papers, code, documentation, and theory. An overview of the parameters and a short explanation of each parameter is provided in Table 3.1.

Parameter	Meaning
dtype	Data type of the computation. Specifies the inference precision.
compute_type	Same as dtype
beam_size	Specifies the 'width', i.e. N candidates at each split during beam search
vad_filter	Specifies if VAD is enabled during transcription
batch_size	Number of samples propagated through the network per forward pass

Table 3.1: Overview of parameters and their meaning

3.2.1 dtype & compute_type

Both dtype and compute_type parameters control the exact same thing, namely the amount of precision used during model inference. One can conceptualize this as the number of decimals used during calculation. More decimals equals a more accurate calculation, but will require more computing resources and use more memory. For the Whisper models, the expectation is that lower precision will reduce memory usage and computing requirements, without greatly reducing the accuracy of the models' transcription.

3.2.2 beam_size

The beam_size parameter is directly related to the beam search algorithm used in Whisper. This parameter controls the width of the beam, that is the number of candidates (ν, κ, or β) kept at each split. For example, a beam size of 2 implies that only the 2 best candidates are considered at each split, from which they will expanded upon at the next split. The remaining candidates are disregarded and will not be expanded upon. A visual representation of a the beam search algorithm with a beam size of 2 is provided in Figure 3.2. The expectation is that a smaller beam size will increase transcription speed, as fewer candidates are considered at each split, thus requiring less computation. The downside of a smaller beam-size is that memory usage will increase, and potentially accuracy will reduce. The trade-off between increased speed and increased memory usage will be investigated. With beam size set to 1, the beam search algorithm is effectively equivalent to the best-first search algorithm. Here, the algorithm only selects the best or most promising candidate at each split and disregards all other candidates. The algorithm becomes greedy at this point.

3.2.3 vad_filter

In Faster-Whisper, Voice Activity Detection (VAD) can be enabled by setting the vad_filter parameter to True. The expectation is that enabling VAD will increase transcription accuracy, as background-noise and other non-speech

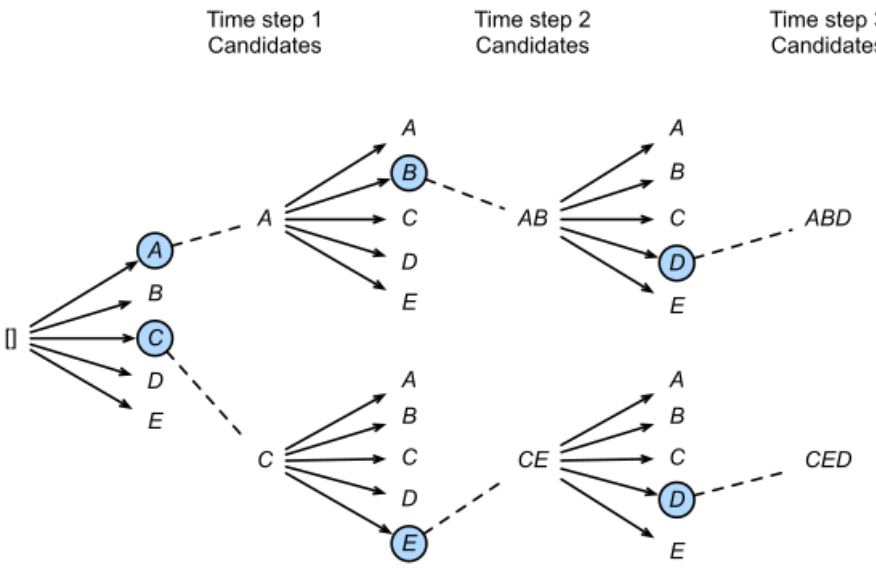

Figure 3.2: A visualization of the beam search algorithm, with a beam size of 2[1]

activity will have been removed, and only segments with speech activity are processed through the transcription pipeline.

3.2.4 batch_size

Batch size refers to the number of samples that are propagated through the neural network in each (forward) pass. The higher the batch size, the more memory is required. In theory, a larger batch size should also provide higher accuracy, as the gradient descent estimate is more accurate due to having a larger sample size, thus more information to consider during calculation. The expectation is that a smaller batch size will be beneficial, as it will reduce memory usage. With memory usage reduced enough, it can be possible to run parallel instances of the model, which can increase overall transcription throughput considerably.

3.2.5 Default values

Table 3.2 provides an overview of the selected tuning parameters used for each model, and the accompanying default value.

[1]https://d2l.ai/chapter_recurrent-modern/beam-search.html

Model	Parameter	Default value
Whisper	dtype	float32
Faster-Whisper	compute_type	float16
Faster-Whisper	beam_size	5
Faster-Whisper	vad_filter	False
WhisperX	compute_type	float16
WhisperX	batch_size	16

Table 3.2: Overview of default model parameter values

Although not clearly specified as a model parameter, both Whisper and WhisperX implicitly contain the beam_size parameter. By default, in Whisper and WhisperX beam_size is set to 5.

3.3 Experiments

3.3.1 Baseline

The first of the experiments to be performed is the establishment of baseline values for the performance metrics. To perform this experiment, a development subset is to be transcribed using the Whisper, Faster-Whisper, and WhisperX models set to the default parameter values shown in Table 3.2.

3.3.2 Transcription speed

Next, transcription speed will be optimized in the second experiment. Using the selected parameters for each model shown in Table 3.2 and the hypotheses proposed in subsection 3.2, the parameter values will be modified and the performance metrics during transcription of the development subset will be logged.

3.3.3 Memory usage

In the third experiment, memory usage will be optimized. This experiment is similar in setup to the second experiment. The parameters from Table 3.2 and hypotheses from subsection 3.2 are used as guidance. Once again, the development subset will be transcribed and performance metrics logged.

3.3.4 Test subset

As a final experiment, the parameters returning the best performance on the reference file are selected for further experimentation on a 10% test subset B of the component C directory of the CGN dataset.

3.4 Calculation of performance metrics

3.4.1 WER

Calculation of the Word Error Rate (WER) for analysis in the results section of this research paper will be performed using the JiWER package[2]. The reference and hypothesis strings are standardized, i.e. lowercased and removal of non-words, before the WER is calculated. Standardization is also handled by JiWER.

3.4.2 Memory usage

Next, memory usage of all model runs are tracked using Weights & Biases[3] (WandB). While memory usage is logged in bytes, it is converted to gigabytes for easier interpretation of results.

3.4.3 RTF

Lastly, processing time for each model run is also tracked using WandB. Dividing the processing time by the total duration of audio files processed returns the RTF.

[2]https://github.com/jitsi/jiwer
[3]https://wandb.ai/site

3.5 A proposal for the modification of Whisper pre-processing

In the pre-processing/pre-transcription phase of Whisper, the audio file to be transcribed is split into 30 second segments and converted to a Log-Mel spectrogram. In an attempt to speed up this process, the feasibility of switching to a Mel-frequency cepstral coefficients (MFCC) spectrogram was investigated. An MFCC spectrogram is computed by taking the logarithm of a Log-Mel spectrogram and then performing a discrete cosine transformation (DCT). MFCC is a more compressed form of Log-Mel, akin to a 'spectrum of a spectrum'. A MFCC spectrogram usually has 13 or 20 coefficients instead of the usual 32-64 coefficients seen in Log-Mel. Following this logic, a MFCC spectrogram should, in theory, be more efficient to process than Log-Mel spectrogram. However, there are downsides to MFCC. The biggest downside in the scope of this thesis is that a Whisper model modified to use MFCC would require retraining. Retraining requires a lot of time and massive resources that are not, as of writing, at my disposal. Another downside is that MFCC is less strongly correlated, and in turn still inferior to Log-Mel when used on strong classifiers or learners with a large amount of training data available. Ultimately, it can be concluded that the downsides outweigh the benefits for the Whisper use-case.

4. Results

4.1 Overview of the results

The performance of the models tested using the 5% development subset is shown in Table 4.1. Results from experiment 1 are presented at the first row for each model. These are the baseline values for each model. Results from experiment 2 and 3 are shown in columns RTF and memory usage.

For all tested models, it can be observed that setting the dtype/compute_-type parameter to a lower setting reduces memory usage and RTF, while having negligible influence on WER.

Regarding Faster-Whisper, setting the beam_size to 1 returns the lowest RTF and lowest memory usage, while increasing WER by approximately 2.90% compared to baseline. The largest reductions are seen in RTF, with reductions of 81.25% measured. Further, setting the vad_filter to True decreases the WER by about 3.29%, bringing the accuracy back closer to baseline. RTF is further reduced by 1.82%, and memory usage by 2.62%.

Concerning WhisperX, it can be noted that batch_size has minimal influence on WER, with at most 0.61% difference observed. Increasing the batch size reduces RTF, but increases memory usage as well. Decreasing the batch size reduces memory usage, but increases RTF instead.

In absolute terms, WhisperX is the fastest model, as the RTF is the lowest. The caveat is that WhisperX has considerably higher memory usage than Faster-Whisper.

The most promising parameters from experiments 1,2, and 3 with the development subset are chosen for further experimentation on an independent test subset of the dataset. The findings from experiment 4 are shown in Table 4.2. The findings here follow the main trend seen with the development subset, with WhisperX being faster, but using more memory.

Model	Parameter	Value	WER	RTF	Memory usage (mean)
Whisper	dtype	float32	**0.349**	0.450	11.50 GB
Whisper	dtype	float16	0.353	**0.378**	**10.73** GB
Faster-Whisper	compute_type	float16	0.335	0.203	5.11 GB
Faster-Whisper	compute_type	int8	**0.316**	0.175	3.32 GB
Faster-Whisper	beam_size	5	**0.316**	0.175	3.32 GB
Faster-Whisper	beam_size	4	0.320	0.156	3.20 GB
Faster-Whisper	beam_size	3	0.328	0.146	3.16 GB
Faster-Whisper	beam_size	2	0.337	0.135	3.17 GB
Faster-Whisper	beam_size	1	0.345	0.112	3.13 GB
Faster-Whisper	vad_filter	True	0.334	**0.110**	**3.05** GB
WhisperX	compute_type	float16	0.330	0.068	9.64 GB
WhisperX	compute_type	int8	0.330	0.064	8.17 GB
WhisperX	batch_size	40	0.330	**0.062**	8.51 GB
WhisperX	batch_size	32	0.330	0.063	8.52 GB
WhisperX	batch_size	24	0.330	0.073	7.81 GB
WhisperX	batch_size	16	0.330	0.064	8.17 GB
WhisperX	batch_size	8	0.330	0.067	7.50 GB
WhisperX	batch_size	4	0.330	0.106	4.53 GB
WhisperX	batch_size	2	**0.329**	0.122	4.39 GB
WhisperX	batch_size	1	0.332	0.136	**4.22** GB

Table 4.1: Overview of the findings for all model and parameter combinations tested on the development subset

Model	Parameter	Value	WER	RTF	Memory usage (mean)
Faster-Whisper	beam_size	5	**0.333**	0.172	3.54 GB
Faster-Whisper	beam_size	1	0.364	0.131	3.48 GB
Faster-Whisper	vad_filter	True	0.335	**0.127**	**3.41** GB
WhisperX	batch_size	16	**0.328**	**0.065**	7.16 GB
WhisperX	batch_size	8	**0.328**	**0.065**	5.99 GB
WhisperX	batch_size	4	**0.328**	0.105	**4.97** GB

Table 4.2: Overview of the findings for most promising model and parameter combinations tested on the test subset

4.2 Alternative approaches

4.2.1 Whisper-JAX

During initial testing of different Whisper variants, it was observed that one model performed so poorly that further experimentation would be futile. The Whisper-JAX implementation was riddled with bugs and issues related to stability and performance. VRAM usage was also unusually high at approximately 11.9 GB, which meant that only the medium model could be

tested on the available hardware (NVIDIA T4). Attempts to run the large-v2 model were unsuccessful, as loading the model would increase VRAM usage to the point that it exceeds limits of available compute resources for this thesis. In accordance with the Whisper-JAX documentation, reduced precision and smaller batch size parameter combinations were attempted in an effort to curb VRAM usage, but to no avail. Initial results of Whisper-JAX with the parameters shown in Table 4.3 on the reference file are shown in Table 4.4.

Parameter	Value
Precision	float16
Model size	medium
batch_size	1

Table 4.3: Initial Whisper-JAX parameter values

Run	Elapsed time
1 (warm-up)	158s
2	102s
3	102s

Table 4.4: Initial Whisper-JAX results on reference file

It can be observed that run 1 is slower approximately 1.55 times slower than the succeeding two runs. This observation is explained by the fact that on the first run, the model has to initialise and compile all the required components *Just In Time* (JIT), i.e. at the first moment the components are called. Once compiled, subsequent model runs use the compiled components from cache, which allows faster run times. The described phenomenon is more commonly known as "warm-up".

4.2.2 Hugging Face Transformers

Another discontinued implementation is the Whisper model provided via the Hugging Face Transformers library[1]. This implementation was not considered for further experimentation as inference is only supported for short-

[1]https://huggingface.co/docs/transformers/model_doc/whisper

form audio, that is to say, audio files shorter than 30 seconds. With a suitably configured pipeline, one could in theory use this implementation for long-form audio of arbitrary length, as long as the pipeline handles segmentation of the input audio into 30 second chunks.

5. Conclusion

In this study, we investigate how the performance of Whisper can be improved. Through the comparison of different Whisper variants and the tuning of hyperparameters, we answer the research question: "What optimizations can be made to improve the transcription performance and efficiency of Whisper and its variants?".

Firstly, experiments 1,3, and 4 show that memory usage can be reduced through multiple strategic choices. Reducing the level of computational precision decreases memory usage on all models tested. Decreasing the batch size reduces memory usage in WhisperX, as do smaller beam sizes in Faster-Whisper.

Subsequently, experiments 1,2, and 4 show that transcription speed will increase by reducing the level of computational precision. Larger batch sizes have also been shown to increase transcription speed. Reducing beam size increases transcription speed too.

Furthermore, experiments 1 to 4 show that enabling Voice Activity Detection (VAD) decreases the Word Error Rate (WER), thus improving the accuracy of the model. The effects on memory usage and speed are also favorable, with memory usage reducing and speed increasing. There are thus no downsides to enabling VAD.

As is often the case, there are trade-offs between model performance and accuracy. Larger batch sizes increase transcription speed, but increase memory usage. Reducing the beam size will increase transcription speed and reduce memory usage, but will reduce accuracy by a small amount. The choices one makes will depend on the constraints set by the specific use-case and implementation.

In conclusion, implementing Faster-Whisper with compute_type set to int8, beam_size set to 1, and vad_filter set to True will give the best results.

This model with the aforementioned parameters values will return the highest overall transcription throughput, as the lower memory usage will permit the usage of multiple, parallel instances of the model. In our findings, Faster-Whisper offers the best trade-off between model performance and accuracy. For the problem statement described in the introduction of this thesis, Faster-Whisper is the best choice.

5.1 Discussion

5.1.1 Implications

Contrary to our initial assumptions, WhisperX may not always be the best model to use. While the initial expectation was that WhisperX would perform better on all metrics due it being newer and having more features than Faster-Whisper, our findings showed that this is not always the case. Model choice depends on the demands and constraints of the specific use-case, and no one-size-fits-all solution exists at this time. For the National Police Corps, implementing either WhisperX or Faster-Whisper is a large improvement over the original Whisper model. The higher processing speed and lower WER lead to better results, both in terms of accuracy and quantity. Consequently, this leads to higher quality investigations.

5.1.2 Limitations

A limitation of this study is that only Dutch language audio has been used. Although Whisper has been designed as a multilingual model, it cannot be guaranteed that the findings of this study will be directly applicable to other languages.

Another limitation is that the subset of the dataset used during experimentation consists exclusively of telephone dialogue. Guarantees cannot be given as to whether the results shown in this study generalize to other forms of speech or conversations.

The relatively small amount of data used in this study is also a limitation.

The component C directory contains only 55 hours of audio. It remains to be seen if the results observed here remain scaling linearly when the scale of data is in the hundreds or even thousands of hours.

One other limitation is that model runs have been performed over the course of multiple days. This was necessary due to Google Colab's usage limits blocking access to GPU resources after a certain threshold of usage was surpassed. Another downside of using Google Colab is that there are no guarantees for runtime priority or resource availability. As such, there are backend factors out of our control that may influence the results of the model performance.

Lastly, while the beam_size parameter was only tested on Faster-Whisper, WhisperX also contains this parameter. Unfortunately, at time of writing, WhisperX contains a bug that prevents the model from responding to certain parameter modifications in the transcribe function. As such, the effects of beam_size on WhisperX could not be tested.

5.1.3 Ethical implications and considerations

As it stands, the Whisper model will not be used to create evidence or make some other type of decision that will influence the outcome of a legal proceeding. The model is used solely as a tool to transcribe confiscated audio data, which can then be more easily searched through by detectives. Legal procedures and due diligence remain as they were previously, as current laws do not give any form of extra leniency to AI tools. The manual listening and eventual correction by a detective is mandatory if a transcript will be used as evidence in a case. As such, we see no ethical implications of this research.

5.1.4 Future research

Due to legal constraints, an essential requirement from the National Police Corps is that the Whisper models tested must be able to run locally. For use-cases where no such requirement is present, it could be worthwhile to experiment and evaluate the performance of WhisperJAX with Tensor Pro-

cessing Unit (TPU) acceleration instead of GPU acceleration. The caveat of using TPUs is that the hardware is proprietary and used by Google exclusively. The only way one can use TPUs is through Google's cloud computing services. Nevertheless, the claimed performance gains are so attractive that one should attempt to use TPUs if the use-case allow it.

At time of writing, WhisperX uses pyannote.audio for Voice Activity Detection (VAD). Another VAD implementation that shows promising result is NVIDIA NeMo. Future research could investigate whether the use of NeMo could improve the overall performance of WhisperX with regard to speed, memory usage and accuracy.

As was discussed in the limitations, the effects of certain parameter modifications could not be tested on WhisperX. Future research should investigate if the results from Faster-Whisper apply to WhisperX as well.

6. Acknowledgements

We would like to thank Vahid Shahrivari Joghan and Duygu Islakoglu for their guidance and wisdom as daily Utrecht University supervisors during this research project, Nils Hulzebosch for his help and cooperation as external supervisor, and Arjen van Vliet and Albert Gatt for their help with organisational matters.

Appendices

A. GitHub repository

Repository containing code created for this thesis

B. Scripts

```
shuf -zen18 nl/* | xargs -0 mv -t nl/sample_5_dev
```

Script for creating the 5% development subset

```
shuf -zen36 nl/* | xargs -0 mv -t nl/sample_10_test
```

Script for creating the 10% test subset

C. Reference audio file

Value	Metric
Name	fn008000
Duration	580s
Format	wav
Dataset	CGN
Directory	component c
Language	Dutch
Speakers	2

Table C.1: Description of reference audio file

Bibliography

[1] R. Pieraccini and I. Director, "From audrey to siri," *Is speech recognition a solved problem*, vol. 23, 2012.

[2] Reddy, Erman, Fennell, and Neely, "The hearsay-i speech understanding system: An example of the recognition process," *IEEE Transactions on Computers*, vol. C-25, no. 4, pp. 422–431, 1976. DOI: 10.1109/TC.1976.1674624.

[3] B. P. Lowerre and B. R. Reddy, "Harpy, a connected speech recognition system," *The Journal of the Acoustical Society of America*, vol. 59, no. S1, S97–S97, Apr. 1976. DOI: 10.1121/1.2003013. [Online]. Available: https://doi.org/10.1121/1.2003013.

[4] J. Baker, "The dragon system–an overview," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 23, no. 1, pp. 24–29, 1975. DOI: 10.1109/TASSP.1975.1162650.

[5] K.-F. Lee, H.-W. Hon, and R. Reddy, "An overview of the sphinx speech recognition system," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, no. 1, pp. 35–45, 1990.

[6] A. J. Oxenham, "How we hear: The perception and neural coding of sound," *Annual review of psychology*, vol. 69, pp. 27–50, Jan. 2018, ISSN: 0066-4308. DOI: 10.1146/annurev-psych-122216-011635.

[7] L. H. Carney and J. M. McDonough, "Nonlinear auditory models yield new insights into representations of vowels," en, *Attention, Perception, & Psychophysics*, vol. 81, no. 4, pp. 1034–1046, May 2019, ISSN: 1943-393X. DOI: 10.3758/s13414-018-01644-w.

[8] P. Pedersen, "The mel scale," *Journal of Music Theory*, vol. 9, no. 2, pp. 295–308, 1965.

[9] P. Lindsay and D. Norman, *Human Information Processing: An Introduction to Psychology*. Academic Press, 1977, ISBN: 9780124509603. [Online]. Available: https://books.google.nl/books?id=6d9OAAAAMAAJ.

[10] J. Makhoul and L. Cosell, "Lpcw: An lpc vocoder with linear predictive spectral warping," in *ICASSP '76. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, 1976, pp. 466–469. DOI: 10.1109/ICASSP.1976.1170013.

[11] J. Harrington and S. Cassidy, *Techniques in Speech Acoustics*. Springer Netherlands, 1999. DOI: 10.1007/978-94-011-4657-9. [Online]. Available: https://doi.org/10.1007/978-94-011-4657-9.

[12] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *CoRR*, vol. abs/1511.08458, 2015. arXiv: 1511.08458. [Online]. Available: http://arxiv.org/abs/1511.08458.

[13] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 international conference on engineering and technology (ICET)*, Ieee, 2017, pp. 1–6.

[14] L. R. Medsker and L. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, no. 64-67, p. 2, 2001.

[15] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," no. arXiv:1706.03762, Dec. 2017, arXiv:1706.03762 [cs]. DOI: 10.48550/arXiv.1706.03762. [Online]. Available: http://arxiv.org/abs/1706.03762.

[16] R. Bommasani, D. A. Hudson, E. Adeli, *et al.*, "On the opportunities and risks of foundation models," *CoRR*, vol. abs/2108.07258, 2021. arXiv: 2108.07258. [Online]. Available: https://arxiv.org/abs/2108.07258.

[17] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, *Wav2vec 2.0: A framework for self-supervised learning of speech representations*, 2020. arXiv: 2006.11477 [cs.CL].

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

[19] D. Hendrycks and K. Gimpel, "Bridging nonlinearities and stochastic regularizers with gaussian error linear units," *CoRR*, vol. abs/1606.08415, 2016. arXiv: 1606.08415. [Online]. Available: http://arxiv.org/abs/1606.08415.

[20] M. Bain, J. Huh, T. Han, and A. Zisserman, *Whisperx: Time-accurate speech transcription of long-form audio*, 2023. arXiv: 2303.00747 [cs.SD].

[21] ivdnt.org, *Corpus Gesproken Nederlands (CGN)*, nl-NL. [Online]. Available: https://taalmaterialen.ivdnt.org/download/tstc-corpus-gesproken-nederlands/ (visited on 06/25/2023).

[22] D. van der Vloed, J. Bouten, and D. van Leeuwen, "NFI-FRITS: A forensic speaker recognition database and some first experiments," in *Proc. The Speaker and Language Recognition Workshop (Odyssey 2014)*, 2014, pp. 6–13. DOI: 10.21437/Odyssey.2014-2.