Bachelor's Thesis Computing Science

Radboud University Nijmegen

Fine-tune Whisper for Speaker Recognition

Author:
Faycel Harrathi
s1013164

First supervisor/assessor:
Prof. David van Leeuwen

Second assessor:
Prof. Djoerd Hiemstra

June 17, 2024

Abstract

In this research, we jointly train openAI's speech recognition system with two closely related tasks: speaker and speech recognition. We extend the multitasking setup of the whisper network to carry out speaker recognition tasks alongside automatic speech recognition. We show that Whisper's timestamp tokens can be repurposed as speaker IDs, and that Whisper can learn speakers' discriminative features through these tokens. Furthermore, we show that the integration of these specialized tokens during the fine-tuning process of the network benefits automatic speech recognition performance.

Contents

Chapter 1

Introduction

The speaker and speech recognition market has significantly increased in the last decade and is anticipated to further increase in the coming years [1]. With the growing reliance on speaker recognition (SR) for security access to authenticate users and speech recognition (ASR) for "hands-free" interaction with devices, researchers continue to explore these two fields despite the significant improvements already achieved. One remarkable point in this progress is that these two tasks are treated separately, although closely related.

The interrelation between speaker and speech recognition stems from their mutual reliance on acoustic features extracted from the speech signal and similarities in network architecture, although for different purposes. ASR models aim to identify the sequence of words that correspond to a given speech signal. SR models aim to extract discriminative features from speech signals that describe a certain speaker. In essence, this problem is cast to extracting local and global features from the speech signal, that is, ASR tries to capture information that varies quickly in speech like words, subwords, or any speech unit. While SR tries to capture general information that describes the whole speech signal to verify or identify the speaker [18].

State-of-the-art SR systems are based on deep neural networks, such as RNNs and CNN to extract a fixed length speaker embeddings [10], [24] from a given utterance, this is usually done by extracting frame-level embeddings and then aggregate them to produce utterance-level speaker embeddings of fixed length. Similarly, ASR models used the same network as an acoustic model to extract linguistic units from a speech signal [15]. With the introduction of transformers, attention is shifted toward pre-training large models for ASR tasks [4]. This network has been successfully used for SR with a minimal change in the architecture due to the capability of transformers to extract local and global features [19]. Moreover, it has been shown that phonetic representation, usually used in ASR tasks, can enhance SR perfor-

mance [21]. Recently, Openai has introduced the state-of-the-art Whisper model [20]. The model excels in various tasks including speech activity detection, language identification, speech translation, and ASR with an impressive performance particularly in accurately transcribing English speech. The of this model stems from the large data used in the training phase and the task-specific tokens that guide the model to perform different tasks. However, The model lacks SR capability.

In this research, we further explore the Whisper network to perform SR tasks alongside ASR. We use timestamp tokens as speaker IDs to train the model to distinguish between different speakers without changing the network structure. Further, we evaluate the performance of the model trained to distinguish between different speakers in the ASR task.

The rest of this paper is structured as follows: Chapter 2 gives an overview of the concepts and metrics of deep learning used in this research. The next section explains the research done for this paper. Previous research into the subject of employing ASR and SR is delved into in section 4, and section 5 closes off with the conclusions drawn from our research.

Chapter 2

Preliminaries

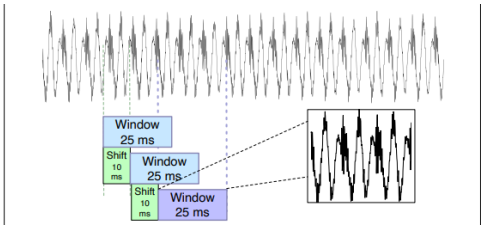
This chapter provides an overview of the basic requirements that have been used in this paper: an overview of acoustic features, transformers, automatic speech recognition (ASR), speaker recognition (SR), and the inner architecture of the Whisper model.

2.1 Feature extraction

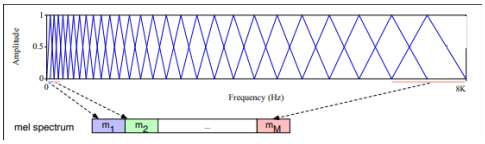
Feature extraction is converting speech signals into acoustic feature vectors that describe them over a fixed time. Logarithmic Mel spectrogram is a type of spectrogram commonly used in speech recognition [3, 9]. Generating log-mel spectrogram is created through several steps: First, the speech signal is divided into short overlapping frames (usually these frames range from 20 to 40 ms in duration, with an overlap of 50%, also known as the stride or shift, between consecutive frames). Each frame is then passed through a windowing function, typically a Hamming window [23], as shown in Figure 2.1a, to reduce spectral leakage, this crucial step as the speech signal is non-stationary.

Next, the short-time Fourier transform (SFT) is applied to each windowed frame to convert the signal from the time to frequency domain, producing a power spectrum [11]. From the power spectrum, Mel filterbanks are applied Figure 2.1b. These filterbanks are designed to mimic the non-linear human perception of sound [9], where frequencies are binned into mel frequency bins rather than linear frequency bins. Finally, we take the logarithm of each of the Mel spectrum values. The model used in this paper takes as input a log-mel spectrogram with 80 channels (bins).

[1]Source: `https://web.stanford.edu/~jurafsky/slp3/16.pdf`



(a) 25 ms rectangular window with a 10ms stride



(b) The mel filter bank Each triangular filter, spaced logarithmically along the mel scale, collects energy from a given frequency range

Figure 2.1: [1]

2.2 Convolutional layer

A convolutional layer is commonly used in neural networks to process image data. A convolutional filter, defined by the kernel size parameter, is essentially a small window that slides over the input image. At each position, the filter computes a weighted sum of the pixels it covers. The kernel size parameter specifies the dimensions of these filters. Pading and stride, are also commonly used to adjust the size of the output. While the former extends the dimension of the output, the latter reduces it by adjusting the step in which the sliding window operates. For an input with dimension (in_h,in_w), kernel of size (k_h, k_w), padding (P_h, P_w), and stride (S_h, S_w), the dimension output can be computed by:

$$Out_h \times Out_w = \lfloor \frac{in_h - k_h + P_h + 1)}{S_h} \rfloor \times \lfloor \frac{in_w - k_w + P_w + 1)}{S_w} \rfloor$$

The convolutional layer can be used to reduce the computational complexity of processing a spectrogram [3].

2.3 Byte level encoding

Byte-level BPE is a compression technique frequently employed in natural language processing (NLP) to tokenize text. It operates by iteratively merging the most common pairs of bytes until it achieves the desired vocabulary size or a specified level of compression. The tokenizer breaks down the text

into individual bytes. Each character in the text is represented by one or more bytes according to its encoding (UTF-8 is used in the case of Whisper's tokenizer).

2.4 Transformers

Transformers were first introduced in 2017 by Ashish Vaswani in the paper *Attention is all you need*. They transform a set of vectors $X = (x_1, \ldots, x_N)$ in some representation space into a corresponding set of vectors $Y = (y_1, \ldots, y_N)$, having the same dimension in some new space with richer representation. x_i can represent a word in a sentence, a patch in an image, a frame in a speech signal. The output y_i, depends on the entire input sequence X. The foundation that underpins the transformer is attention and parallelism. Attention is where the model learns to focus more on parts of the input that are more relevant to the output and "ignore" those that are not relevant. the relevance is computed by; First, creating three matrices that serve as query (Q), key (K), and value (V). These matrices are created by projecting the input X onto three learned weight matrices W_Q, W_K ,and W_V. Second, compute the attention scores which can be dot product between Q and K matrices scaled by a factor proportional to their dimension; Finally, these scores are passed through a softmax function to obtain attention weights, which are then applied to V to produce the context vectors. A single transformer layer is composed of two components: multi-head attention, which is attention applied multiple times in parallel, and a fully connected layer to adjust the output of one transformer layer to be an input for the next layer.

The transformer encoder and decoder have an equal number of transformer layers. The difference is that the transformer decoder layer employs two types of attention: self-attention and cross-attention. Self-attention is attention applied within a single sequence, where the key, query, and value are constructed from the same sequence. Cross-attention, also known as encoder-decoder attention, involves two sequences. In this case, the key and value are derived from the encoder's hidden states, whereas the query is derived from the output of the decoder's self-attention. In this case, cross-attention computes the relevance of the encoder's hidden state to the decoder's prediction. One more difference in the attention is that; attention used in an encoder's transformer layer is full, i.e., the attention is computed over the whole sequence, whereas in the decoder's layer, attention is causal, that is, at time step t, attention is computed over only the previous "past" tokens $i < t$.

2.5 Automatic Speech recognition

Automatic speech recognition (ASR) models aim to accurately map a sequence of acoustic features $O = (o_1, \ldots, o_T)$ to a sequence of words/tokens $W = (y_1, \ldots, y_n)$. These acoustic features are extracted from the audio signal as explained in section 2.3. Traditionally, ASR models are composed of three sub-models [22]: acoustic model, lexical model, and a language model. The role of the acoustic model is to map acoustic features to a linguistic unit, widelly used unit are phonemes. The lexical model which is simply a dictionary that describes how words are pronounced phonetically. Finally the langage model [7] estimates the order of this sequence of words. This approach is complicated to optimize, as each model is optimized separately.

In contrast, end-to-end ASR models map speech sequences directly to word sequences using a single network [17] where the aim is to determine a prediction $\hat{W} = \text{argmax}_W P(W|O)$. Most commonly used appraches to model $P(W|O)$ are: attention based models such as Whisper [20], Connectionist Temporal Classification (CTC) [13], and RNN-T [12]. The last is an encoder-decoder model that doesn't rely on a cross-attention mechanism to model the conditional probability distribution of predicting the output units. The network consists of several components:

- Encoder: A normal acoustic model that can be a transformer encoder.
- Decoder (known as predictor): An autoregressive language model.
- Joiner: A feedforward network that concatenates the encoder and decoder hidden states, passes them to a fully connected linear layer and then applies a softmax to output the probability distribution over all vocabulary units, including a skip token Ø.

At each time step, if the prediction is not , the predicted token is added to the decoder input, and the encoder processes the next audio frame. If not, the decoder's input remains unchanged while the encoder processes the next time step. Essentially, this approach is similar to CTC but with the ability to look at the previously generated tokens.

2.6 Whisper

Whisper [2] is a transformer encoder-decoder-based model trained in a supervised manner on 680,000 hours of multilingual audio samples. The model comes in various sizes—tiny, base, small, medium, and large—differing in the number of transformer blocks and attention heads. Using a leading set of special tokens, the model is guided to perform different tasks such as transcribing from 99 different languages to English. All pre-trained checkpoints are available in Hugging Face transformers. Training Whisper requires data

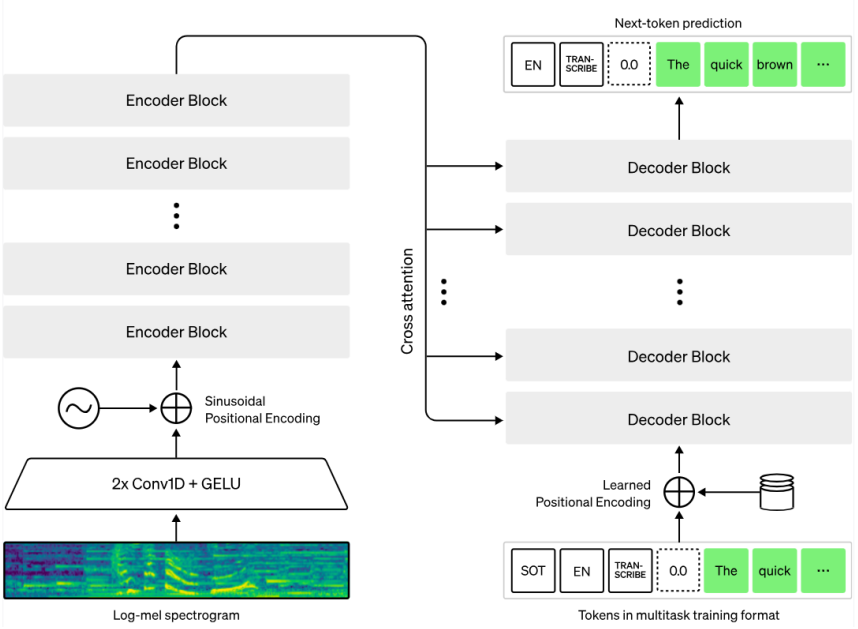


Figure 2.2: Whisper [2]

consisting of speech signals paired with their corresponding transcriptions. Training can be summarized in 6 steps:

- prepare transformer encoder input: The input of the transformer encoder is a sequence of vectors represent the speech signal of 30 seconds. the speech signal is processed by the feature extractor and produces a fixed size 2D dimension matrix of shape 80×3000, where 80 represents the number of channels and 3000 represents the time steps. This is further processed by two convolutional layers that further extract local features (extending the size of the vectors from 80 to d_{model}) which is the inner dimension of the transformer encoder, and reduces the dimension to $1500 \times d_{model}$. Adding a positional encoding results in a sequence of vectors Z of shape $1500 \times d_{model}$
- prepare transformer decoder input: The tokenizer maps the input text to a sequence of tokens in the range $[0, 51864]$. Tokens are transformed to embedding vectors from the learnable embedding, which can be seen as a look-up table over the vocabulary size; adding positional encoding. The result is forwarded to the transformer encoder
- The transformer encoder maps the sequence Z through N blocks of transformer encoder blocks to a context vectors C having the same dimension.

- transformer decoder with causal self-attention, that is, mask out the future tokens, predicts the probability over transcriptions conditioned through cross-attention on C.

- Fully connected linear layer processes the last decoder hidden states and outputs a vector of dimension the size of the vocabulary for each token, known as logits, that are transformed to probabilities using the softmax function. Finally depending on the generation strategy, where the simplest one is greedy search, the model selects the token with the highest probability.

- cross-entropy is applied to measure whether the model learns the correct alignment between the predicted tokens and the truth labels during training.

2.7 Speaker Recognition

Speaker Recognition (SR) is the problem of identifying or verifying the identity of a speaker based on their voice characteristics. In practice, standard SR protocol involves three steps [14]: training, enrollment, and evaluation. In the training step, acoustic features are extracted from the speech signal and passed through a DNN [5], to extract frame-level speaker features (embedding), followed by statistical pooling (average) to aggregate frame-level speaker feature vectors. The embeddings are subsequently projected over the speaker IDs in the training, and a loss function is used such as cross-entropy loss to penalize wrong mapping from utterance to speaker ID. The aim of this phase is to train the model to extract compact embeddings to be able to generalize to new speakers. At the enrollment phase, the model is trained on a specific set of utterances belonging to new speakers. Finally the model is evaluated on a pair of utterances, one from the enrolment set and a verification utterance. Evaluation is based on comparing the similarity of embedding pairs using a similarity function such as a cosine function. The resulting score is then compared to a threshold to decide if they belong to the same speaker or by different speakers. This approach requires that the model has a prior knowledge of the speakers used during evaluation at the training time. In this paper, we evaluate Whisper on trials involving speakers who were unseen during the training phase. Two types of errors can represent the performance of the SR system; False Acceptance Rate (FAR), and False Rejection Rate (FRR). FAR occurs when the system classifies a pair of utterances uttered by two different speakers to be uttered by the same speaker. FRR, on the other hand, is the frequency of incorrectly classifying pairs of utterances uttered by the same speaker to be uttered by different speakers.

This approach requires that the model has a prior knowledge of the

speakers used during evaluation at the training time. In this paper, we evaluate Whisper on trials involving speakers who were unseen during the training phase.

The performance of SR system can be represented by two types of errors; False Acceptance Rate (FAR), and False Rejection Rate(FRR). FAR occurs when the system classifies a pair of utterances uttered by two different speakers to be uttered by the same speaker. FRR, on the other hand, is the frequency of incorrectly classifying pairs of utterances uttered by the same speaker to be uttered by different speakers.

The receiving operating characteristic (ROC) curve is an evaluation metric for the binary classifier that plots the probability of correct acceptance (1-FRR) against FPR at various decision threshold values. Adjusting the threshold in a binary classification model impacts the trade-off between False Acceptance Rate (FAR) and False Rejection Rate (FRR). Increasing the threshold tends to decrease the FAR while increasing the FRR, and vice versa. The value for equal FRR and FAR is called the equal error rate (EER). Evaluating SR model using EER requires many pairs of speech segments.

2.7.1 Word Error rate

The word error rate (WER) is a metric used to evaluate the performance of ASR models by comparing the aligned actual transcription of the audio (reference) and the generated output sentence (hypothesis). The WER calculation can be expressed as follows:

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

Here, in the formula, S represents the number of substitutions, D represents deletions, I represents insertions, and N is the total number of words in the reference sentence. The closer the WER value is to 0, the fewer errors there are between the reference and hypothesis, indicating better ASR model performance.

Chapter 3

Research

In this chapter, we will first provide an overview of the data pipeline, explain the steps used to fine-tune the Whisper models with and without speaker IDs. and finally, evaluate the performance of the models in both tasks as well as the effect of finetuning using speaker ids on the ASR task.

3.1 Data

Our research employs the clean Librispeech dataset[2] for both tasks: speaker verification and speech recognition. The data is a corpus of English speech derived from audiobooks designed to train and evaluate ASR systems. Each audio file in the data set is in FLAC format. The datasets use a specific naming for each audio file, for example, the file named 1034-121119-0000.flac. Here:

- Speaker ID: 1034
- Book ID is: 121119
- Utterance ID: 0000

Each speech signal corresponds to a sentence spoken by the speaker, which we use as a label. We use the three partitions of this dataset: training set, development set, and testing set. We further divide the training set into training and validation subsets to train and validate the loss in the SR task. Specifically, we split the training set into an 80% training subset and a 20% validation subset. This is achieved by grouping the training set by speaker IDs and then randomly selecting 20% from each group for the validation set. Thus, the train and validation set have the same set of speakers. All datasets are stored in CSV files. Each row in the file consists of:

- Path to the FLAC file: The file path to the audio file.
- Speaker ID: The identifier for the speaker in the audio file.

- Utterance ID: The identifier for the specific utterance.
- Transcription: The textual transcription of the spoken utterance.

Dataset	samples	speakers	av.length
train	22725	251	12.68
validation	5814	251	12.71
development	2692	40	7.07
test	2609	40	7.32

Table 3.1: Number of samples, number of unique speakers, and average length of utterances for each of the datasets we consider for our experiments

3.2 Pre-processing

To fine-tune multilingual Whisper models. The data requires pre-processing steps. Fine-tuning requires pairs of audio and labels; the maximum audio length that Whisper can handle is 30 seconds; the maximum length of labels is 448. The encoder stack takes as input a sequence of acoustic features extracted from the audio signal, while the decoder takes as input a sequence of word tokens, in the range of the vocabulary size, extracted from the labels. Both audio and labels have to be processed separately before starting the training process.

The pre-trained Whisper checkpoints provided by HuggingFace[1] Transformers are used in this research. The complete code is shared on GitHub[2]

3.2.1 Filtering the data

Audio files that are longer than 30 seconds are removed from the data. we use Liberosa to load and convert the audio files from FLAC to waveform with a 16000 sampling rate. The output of this function is an array that represents frequency information in the audio file. The duration of the audio can be determined as:

$$d = \frac{N}{R} \tag{3.1}$$

where N is the total number of samples in the waveform, and R is the sampling rate

[1]Whisper: `https://huggingface.co/docs/transformers/model_doc/whisper`
[2]GitHub: `https://github.com/faycel-dev/bachelor_thesis`

3.2.2 Feature-extraction

Feature extraction is performed using Librosa, which processes the waveform array extracted from a speech signal into a log-mel spectrogram representation. This function performs the following:

- Padding: The waveform representation is padded to a fixed length of 30 seconds by appending zeros at the end of the array. These zeros represent silence in the frequency representation of speech.
- Log-Mel Spectrogram Transformation: The padded waveform is then transformed into a log-mel spectrogram representation, as explained in section 1. The sampling rate parameter is set to 16,000 Hz

3.2.3 Label tokenization

Whisper's tokenizer encodes and decodes the labels for both the SR and ASR tasks. The tokenizer can be seen as a bijective function that maps a sequence of words to a sequence of integers within the range of 0 to 51,464. Similarly, the tokenizer decoder performs the inverse mapping. The output of the tokenizer is a dictionary with input IDs and an attention mask, unlike the input speech, the transformer encoder requires attention mask information as the labels are of variable lengths. In the ASR task, encoding the labels is straightforward, however, encoding the speaker IDs is challenging as we need to keep track of these tokens in the last hidden layers. For example, a speaker with the id 134 is divided $[13, 4]$, then it maps them to $[4762, 18]$. We want all speaker ID information to be encoded in a single embedding, so we need a single-token encoding of the speaker ID. Therefore we used the time stamp tokens, i.e. [50364,..,501464], as speaker IDs. This is done by sorting speaker IDs in data and incrementing each speaker ID by its index in the list and 50363. To fine-tune the model on the ASR task, we utilize the encoded labels generated by the Whisper tokenizer. In both SR and ASR tasks, the speaker ID is encoded in the token sequence. Specifically, the speaker ID is included as the third position in the tokenized labels array. Table 3.2 provides an example. To this point, the class LibriSpeechDataset is used for the previously explained processing steps, the class initializes the parameters by parsing a CSV file and uses Liberosa to load the audio files. Get Item to process feature extraction and tokenization as well as encoding the speaker IDs in the tokenized labels. Returns a dictionary with keys input features, input-ids, and speaker-ids and values array.

3.2.4 Shift-right and padding

The model internally appends a token ID of the start of the sentence to the input IDS of the decoder; hence, to preserve alignment between the true

type	SID	label	decoder input IDs
1	50366	"Hi"	[<SOT>, 50259, 50359, 50363, 17155, 50257]
2	50366	"Hi"	[<SOT>, 50259, 50359, 50363, 50366,17155, 50257]

Table 3.2: decoder input IDs used during fine-tuning ASR (type 1) task and jointly ASR and SR (type 2). SID denotes speaker ID.

labels and the predicted labels, removing the start of the transcription token from the true label is required to keep alignment between the true labels and the predicted labels. The model is trained to minimize the cross entropy loss between the ground-truth labels and the predicted labels. Since the labels are of different lengths, padding to the maximum length in a mini-batch is required. Whisper encodes padding tokens as -100, which are mapped internally to <|endoftext|>, but are ignored in computing the loss.

3.3 Finetune Whisper models

We fine-tune four whisper models: tiny, base, small, and medium with and without freezing the encoder parameters. We use PyTorch Lightning to simplify the fine-tuning process. We train all models for 6000 steps (approximately 4 epochs). We used a two-stage learning rate scheduler with ADAM optimizer [16], we make use of 10% of the training steps for warmup, followed by exponentially decreasing in the remaining training steps. the maximum learning rate and the proportion of the steps used during training are summarized in table A.

We use an effective batch size of 16 for all models; depending on the batch size that we can use for each model, we increase the gradient accumulation steps to mimic training with a batch size of 16 for bigger models; that is, the model weights will be updated after accumulating the gradients of 16 data samples.

3.4 Evaluation

The fine-tuned models are evaluated based on their performance on ASR and SR tasks. Evaluation is done by comparing the decoded output by the model (also known as the hypothesis) with the decoded ground-truth labels (also known as the reference).

To evaluate the performance of the models on ASR task, we use the WER metric. The WER penalizes ASR models for trivial differences, such as punctuation and casing. To ensure that these are not mistakenly con-

sidered errors in the ASR task, we normalize the models' output before computing the WER. Furthermore, we remove speaker identities from the models' output and labels before decoding them to the text format.

Performance on the SR task is evaluated using the equal error rate (EER). Computing EER requires extracting embedding for each speaker in the development and test sets. Whisper predicts the probability distribution for the next token Y_i by conditioning on all previous tokens and the encoder's last hidden states

$$P(Y_i|C) = \prod_{t=1}^{i} P(Y_i|Y_{t\leq i-1}, C)$$

where C is the encoder's last hidden states, Y is the decoder input IDs with length T, and $Y_0 =$ `<|startoftranscript|>`.Since speaker ID tokens are encoded at the third time step in the decoder input IDs, the model is designed to predict the token at that time step. We assume that,

$$\text{argmax}P(Y_3|Y_0, Y_1, Y_2, C) \in [50364, \ldots, 51464]$$

And since $P(Y3|Y0, Y1, Y2, C)$ is just the projection of the decoder's last hidden states over the vocabulary size, which is the logit at the third time step

$$P(Y3|Y0, Y1, Y2, C) = \text{argmax}(logits(t_3))$$

, then embedding at index three of the decoder's last hidden state is the most representative of the speaker ID token. To verify this assumption, we examined the number of samples in the development and test datasets where the speaker ID token is correctly predicted at the third time step. We found that in the validation set, all speaker IDs were predicted correctly at time step three. In the development and test sets, the speaker tokens appeared correctly in all samples for the tiny model. However, for the other models, there was a small number of samples where the speaker tokens did not appear, as shown in Table A.1

SR models are evaluated by giving them many pairs of speech segments. However, creating pairs from the test dataset can be redundant. Assume we have N audio samples, for each sample i, we extract speaker embedding vector e_i. Let $E \in R^N \times d$, be the matrix of embeddings, where d is the dimension of each embedding vector.

$$E = \begin{bmatrix} e_1 \\ e_2 \\ .. \\ .. \\ e_N \end{bmatrix}$$

First, normalize each embedding vector $\mathbf{e}_i$ to have unit length by dividing it by its Euclidean norm. Compute the cosine similarity matrix between all pairs of embeddings. Let E be the matrix of normalized embeddings, where each row represents an embedding.

$$S = EE^{\top} \tag{3.2}$$

Here, S is the similarity matrix, where S_{ij} is the cosine similarity score between the embeddings $_i$ and e_j. Since cosine similarity is symmetric, S is also symmetric and the diagonal will be 1 (since the similarity of a vector with itself is 1). Therefore, for efficient computation of the EER, we can consider only the upper triangle matrix.

$$\text{UpperTri}(S) = \{S_{ij} \mid i < j\} \tag{3.3}$$

Similarly, we generate a binary label matrix L, where $L_{ij} = 1$ if the embeddings i and j belong to the same speaker and $L_{ij} = 0$ otherwise, and we consider only the upper triangle matrix. Flattening these two matrices results in two arrays: one for labels and one for scores, where each score is labeled with 1 for the same speaker and 0 for different speakers. These arrays are then used to compute the EER.

3.5 Experiments & Results

In this section, we evaluate the performance of the fine-tuned models. The experiments are conducted in the Radboud cluster. We fine-tuned each model separately for ASR and jointly for ASR and SR using the same hyperparameters. Checkpoints were saved at the end of each training epoch. We evaluated the performance of the saved checkpoints using the development set. The checkpoint with the lowest EER was selected as the fine-tuned model, which was then used to test the performance on the test dataset.

Model	**Tiny**		**Base**		**Small**		**Medium**	
	Dev	**Test**	**Dev**	**Test**	**Dev**	**Test**	**Dev**	**Test**
HF	8.16	7.55	6.01	5.04	3.41	3.5	3.44	2.88
ASR	6.67	6.82	4.76	4.6	3.23	3.37	2.32	2.56
ASR + SR	6.75	6.9	4.78	4.76	3.21	3.44	2.32	2.55
EER	11.4	12.6	10.81	9.83	10	10.4	7.5	7.9

Table 3.3: compare WER and EER on the development and test data sets for models fine tuned on ASR, ASR + SR. HF refers to the base multilangual chekpoints (not fine-tuned).

3.5.1 Speaker recognition

In this Section, to answer our first research question, from the results in Table 3.3, we observe that the models fine-tuned using speaker ID tokens are comfortably under 50%. This indicates that Whisper can perform SR tasks. The lowest EER achieved is for the medium model with 7.5 and 7.9 for the development and the test datasets, respectively. Whisper is originally pre-trained for ASR task, therefore, it may be possible that the model loses speaker features before reaching the last layer.

By further inspecting speaker embedding from different layers of the decoder stack but in a similar way, i.e., third index in the decoder hidden layer. The results in Table 3.3 show that the EER for almost all models is increasing in the top layer compared to the lower layers. Therefore, extracting speaker embeddings from lower layer is more beneficial for SR.

3.5.2 Speech recgnition

Fine-tuning Whisper multilingual checkpoints indeed enhanced the models' capability and improved the WER scores. Results in Table 3.3 show significant improvement for the tiny model, with a reduction from 7.55% to 6.82%. The overall best performance is achieved by the larger models, with the Medium model achieving 2.32% and 2.55% for the development and test datasets, respectively.

Fine-tuning using speaker ID tokens shows a slight degradation in the models' performance. This might be because we trained the models with the same training effort. However, the performance of the medium model did not degrade and even showed a slight improvement.

decoder layer	Tiny		Base		Small		Small	
	Dev	**Test**	**Dev**	**Test**	**Dev**	**Test**	**Dev**	**Test**
0	50	50	50	50	50	50	50	50
1	16.04	14.29	12.25	11.14	10.84	10.47	36	38.2
2	13.27	13.67	13.43	11.65	10.30	8.33	27.1	30.5
3	12.6	12.59	12.72	11.1	10.76	8.48	24.5	27.6
4	11	12.6	12.96	11.28	8.45	6.87	22.6	23.7
5	-	-	11.38	9.78	7.78	6.84	12.8	14.4
6	-	-	10.81	9.83	7.38	7.03	12.6	13.9
7	-	-	-	-	7.42	7.32	13.8	15.1
8	-	-	-	-	7.88	7.99	14.5	15.6
9	-	-	-	-	6.73	6.94	14.2	15.4
10	-	-	-	-	8.10	6.85	12.5	14.9
11	-	-	-	-	7.72	7	13	15.2
12	-	-	-	-	10.04	10.35	15	16.9
13	-	-	-	-	–	–	14.5	15.6
14	-	-	-	-	–	–	13.1	14
15	-	-	-	-	–	–	8.4	8.5
16	-	-	-	-	–	–	8.1	8.5
17	-	-	-	-	–	–	7.7	8.5
18	-	-	-	-	–	–	7.9	8.6
19	-	-	-	-	–	–	7.5	7.7
20	-	-	-	-	–	–	7.8	8.2
21	-	-	-	-	–	–	8	8
22	-	-	-	-	–	–	8	8.1
23	-	-	-	-	–	–	8	8.2
24	-	-	-	-	–	–	7.5	7.9

Table 3.4: EER for development and test sets for different models. Speaker embeddings are extracted from all decoder's output layers at the third time step

Chapter 4

Related Work

In this section, we will discuss some of the previous work that has been done in joint training of ASR and SR recognition. As we mentioned in the introduction, ASR and SR have been treated as two separate research areas.

Tijn [6] showed that the wave2vec.2.0 network, a self-supervised pre-trained model, can perform three tasks: ASR, SR, and speaker change detection. Tijn showed that by introducing speaker class tokens in the target label, the network could simultaneously learn the phoneme tokens and the speakers/speaker-change tokens. Using this approach, he showed that the network's performance on ASR slightly degraded when combining ASR and SR for a single utterance. Using trials, the knowledge learned by the network about the speakers enhanced its performance in ASR task
Besides this, most papers are inducting fine-tunig large model for a specific task rather than multitasking.

[8] explored the representations extracted from transformer encoder models, specifically Wav2Vec 2.0, HuBERT, and UniSpeech-SAT to verify the speaker. In this approach, Chen employed the ECAPA-TDNN architecture on top of these pre-trained models to extract speaker embeddings, utilizing input from various layers of the pre-trained models. Despite achieving commendable results, Chen's experiments indicated an interesting finding: the lower layers of the pre-trained models exhibited a greater ability to capture speaker-related information than their higher-layer counterparts. This observation suggests that the early layers in the transformer-based models are more effective in capturing features relevant to speaker verification.

Contribution: The main contribution of this paper is integrating speaker recognition (SR) capability into the multitasking stack of the Whisper architecture. We demonstrate that the Medium model can learn speaker discriminative embeddings without degrading its performance on the ASR task.

Chapter 5

Conclusions

We have shown that we can build a speaker recognition system alongside speech recognition using a whisper network. By incorporating a speaker ID token into the decoder input IDs, Whisper can learn to associate speaker features with the corresponding speaker's ID. Fine-tuning Whisper for joint tasks has slightly increased the model's performance in ASR tasks. Furthermore, richer speaker features can be extracted from lower decoder layers.

Although we have shown that Whisper can perform SR alongside ASR, the data wasn't suitable for both tasks. Librispeech consists primarily of audiobook readings, resulting in a lack of spontaneity and natural conversational speech patterns. Consequently, SR models can't learn much from this data. This issue can be mitigated by using a dataset specifically collected for SR, such as Voxceleb, but this data lacks transcriptions, posing a challenge for our task. Future work could, for example, use Whisper to transcribe Voxceleb.

Bibliography

[1] Voice and speech recognition market size, share trends analysis report by function (speech recognition, voice recognition). 2023. https://www.grandviewresearch.com/industry-analysis/voice-recognition-market.

[2] Tao Xu Greg Brockman Christine McLeavey Ilya Sutskever Alec Radford, Jong Wook Kim. Robust Speech Recognition via Large-Scale Weak Supervision. *Electrical Engineering and Systems Science*, 2022.

[3] Abdul Malik Badshah, Jamil Ahmad, Nasir Rahim, and Sung Wook Baik. Speech emotion recognition from spectrograms with deep convolutional neural network. In *2017 International Conference on Platform Technology and Service (PlatCon)*, pages 1–5, 2017.

[4] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems*, 33:12449–12460, 2020.

[5] Zhongxin Bai and Xiao-Lei Zhang. Speaker recognition based on deep learning: An overview. *Neural Networks*, 140:65–99, 2021.

[6] Tijn Berns, Nik Vaessen, and David A van Leeuwen. Speaker and language change detection using wav2vec2 and whisper. *arXiv preprint arXiv:2302.09381*, 2023.

[7] Peter F Brown, Vincent J Della Pietra, Peter V Desouza, Jennifer C Lai, and Robert L Mercer. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–480, 1992.

[8] Zhengyang Chen, Sanyuan Chen, Yu Wu, Yao Qian, Chengyi Wang, Shujie Liu, Yanmin Qian, and Michael Zeng. Large-scale self-supervised speech representation learning for automatic speaker verification. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6147–6151. IEEE, 2022.

[9] Steven Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE transactions on acoustics, speech, and signal processing*, 28(4):357–366, 1980.

[10] Shaojin Ding, Tianlong Chen, Xinyu Gong, Weiwei Zha, and Zhangyang Wang. Autospeech: Neural architecture search for speaker recognition. *arXiv preprint arXiv:2005.03215*, 2020.

[11] Lutfiye Durak and Orhan Arikan. Short-time fourier transform: two fundamental properties and an optimal implementation. *IEEE Transactions on Signal Processing*, 51(5):1231–1242, 2003.

[12] Alex Graves. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*, 2012.

[13] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376, 2006.

[14] Georg Heigold, Ignacio Moreno, Samy Bengio, and Noam Shazeer. End-to-end text-dependent speaker verification. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5115–5119. IEEE, 2016.

[15] Takaaki Hori, Jaejin Cho, and Shinji Watanabe. End-to-end speech recognition with word-based rnn language models. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 389–396. IEEE, 2018.

[16] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[17] Jinyu Li et al. Recent advances in end-to-end automatic speech recognition. *APSIPA Transactions on Signal and Information Processing*, 11(1), 2022.

[18] Rafizah Mohd Hanifa, Khalid Isa, and Shamsul Mohamad. A review on speaker recognition: Technology and challenges. *Computers Electrical Engineering*, 90:107005, 2021.

[19] Sergey Novoselov, Galina Lavrentyeva, Anastasia Avdeeva, Vladimir Volokhov, and Aleksei Gusev. Robust speaker recognition with transformers using wav2vec 2.0. *arXiv preprint arXiv:2203.15095*, 2022.

[20] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International Conference on Machine Learning*, pages 28492–28518. PMLR, 2023.

[21] Md Hafizur Rahman, Ivan Himawan, Mitchell McLaren, Clinton Fookes, and Sridha Sridharan. Employing phonetic information in dnn speaker embeddings to improve speaker recognition performance. In *Proceedings of the 19th Annual Conference of the International Speech Communication Association (INTERSPEECH 2018)*, pages 3593–3597. International Speech Communication Association (ISCA), 2018.

[22] Ramya Rasipuram and Mathew Magimai-Doss. Acoustic and lexical resource constrained asr using language-independent acoustic model and language-dependent probabilistic lexical model. *Speech Communication*, 68:23–40, 2015.

[23] A. Testa, D. Gallo, and R. Langella. On the processing of harmonics and interharmonics: using hanning window in standard framework. *IEEE Transactions on Power Delivery*, 19(1):28–34, 2004.

[24] Li Wan, Quan Wang, Alan Papir, and Ignacio Lopez Moreno. Generalized end-to-end loss for speaker verification. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4879–4883. IEEE, 2018.

Appendix A

Appendix

A.1 Trining hyperparameters

Model	Max lr_rate	Batch_size	Gradient_accumulation
Tiny	5e-5	16	1
Base	3e-5	8	2
Small	1.5e-5	2	8
Medium	1e-5	1	16

The learning rates are chosen after a hyperparameter search for a peak learning rate, as we are using a two-stage learning rate schedule. For each model, we test learning rates in the range $\{5, 3, 1.5, 1\} \times 10^{\{-4,-5,-6\}}$. We then select the learning rate that produces the lowest WER in the development set. This selected learning rate is then used to train the model on both speaker and speech recognition tasks

Model	validation	development	test
Tiny	0	0	0
Base	0	1	0
Small	0	1	0
Medium	0	10	32

Table A.1: number speaker ID tokens that didn't apear at the third time step of the models' output tokens. These tokens didn't appear at all in any other time step

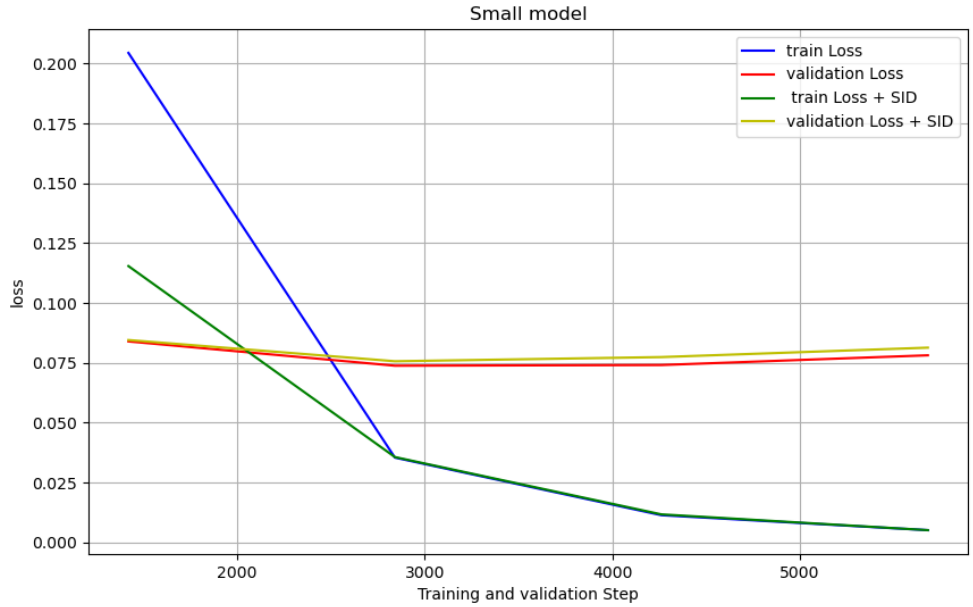


Figure A.1: Loss

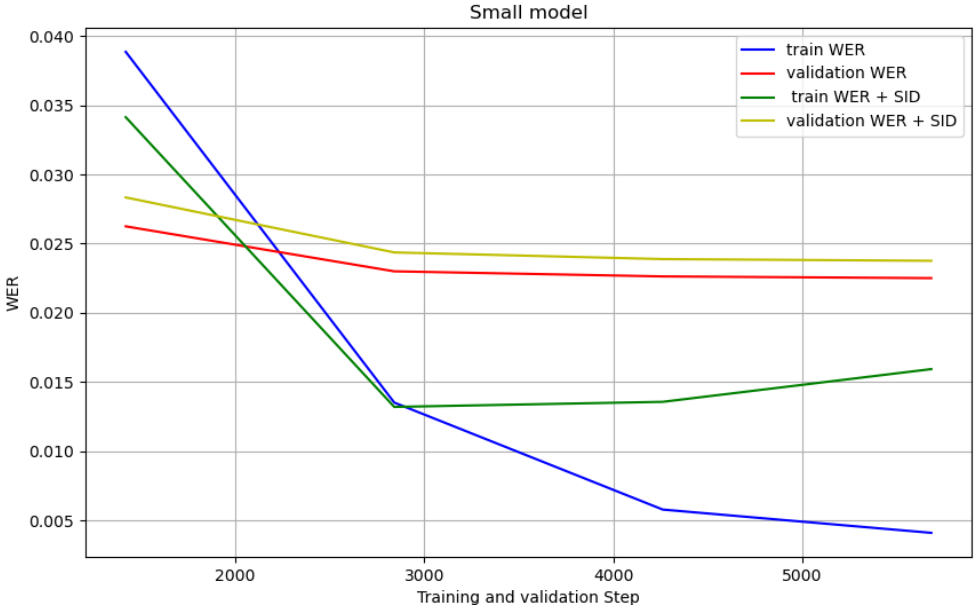


Figure A.2: WER

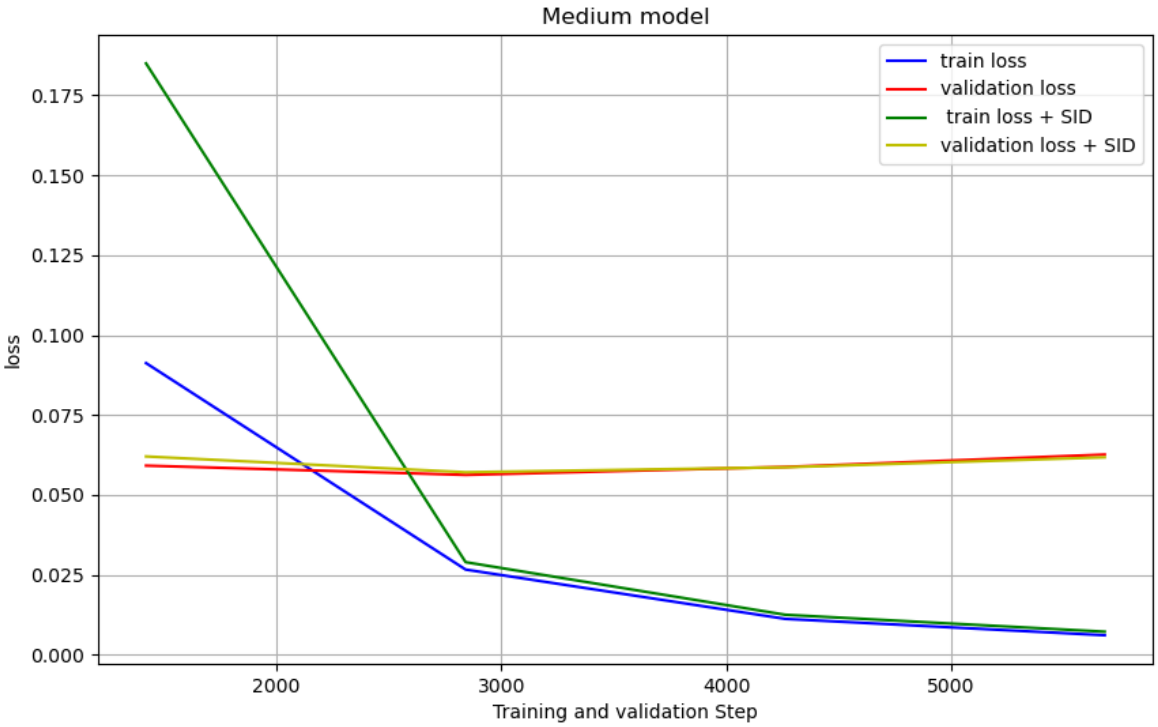


Figure A.3: Loss training and validation

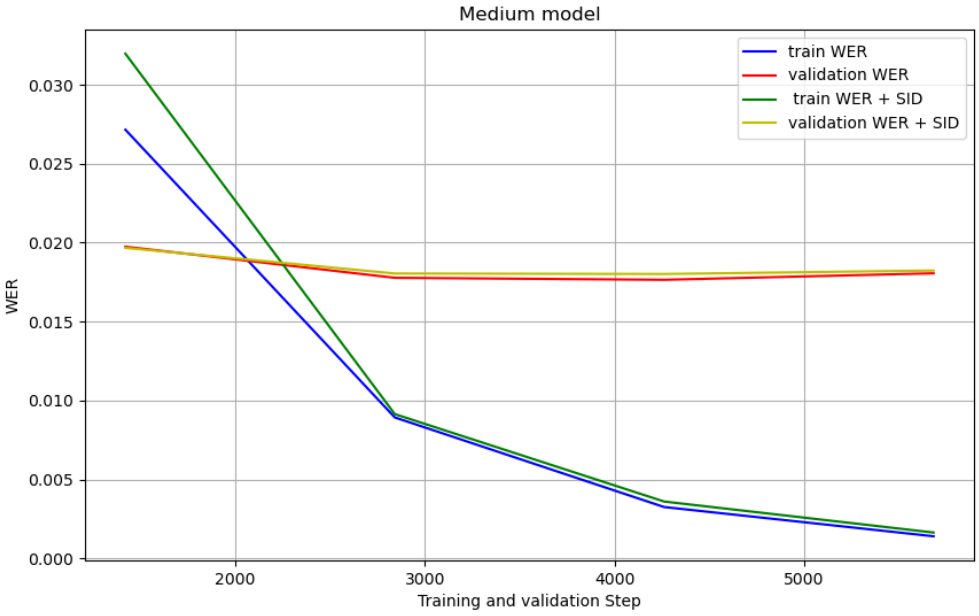


Figure A.4: Enter Caption

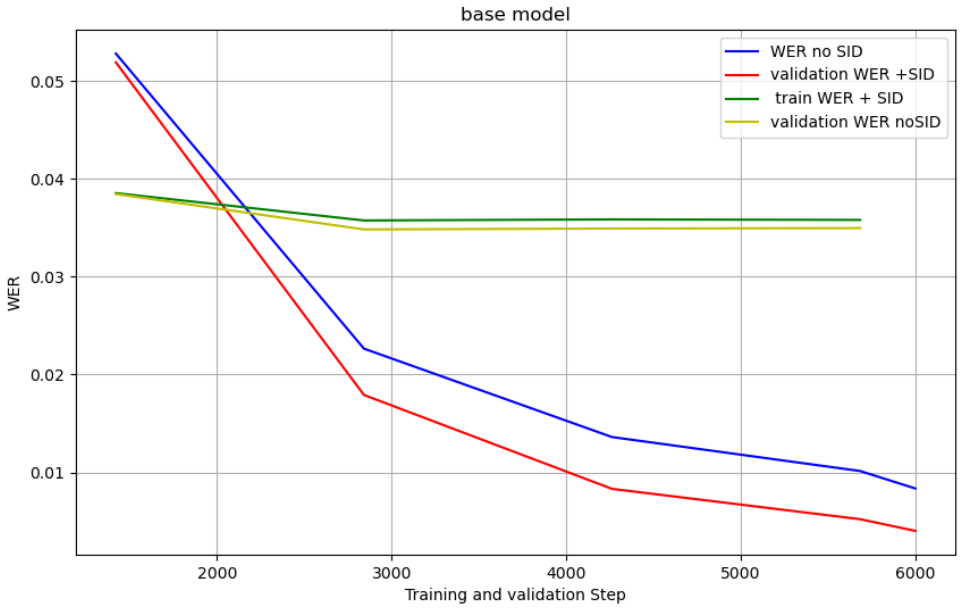


Figure A.5: WER

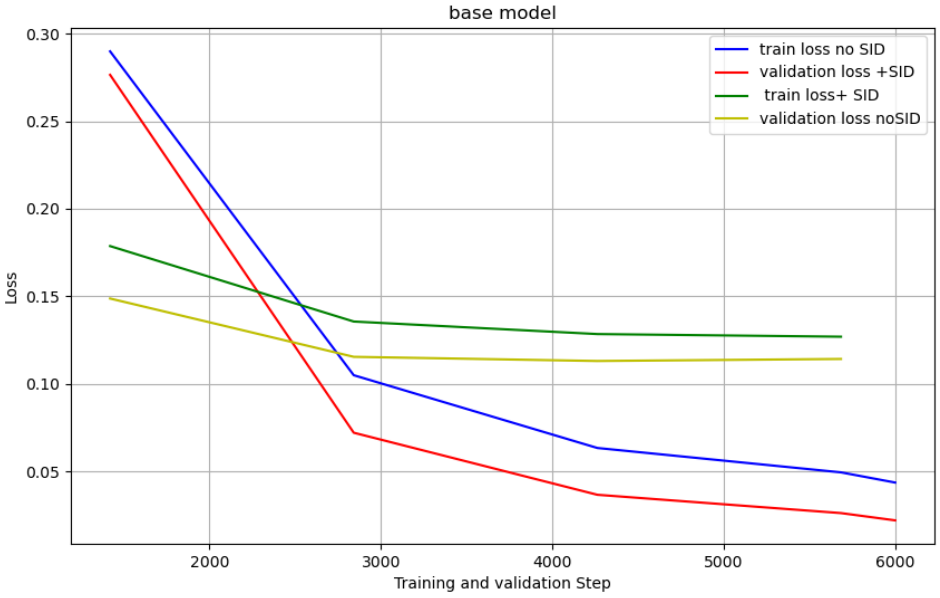


Figure A.6: loss