Universite de Montreal

DESS in Machine Learning

Internship Final Report: Microsoft (Nuance)

Minimum Word Error Rate Training

Lucia Eve Berger

lucia.eve.berger@umontreal.ca

Due Date: January 1, 2023

Submitted: November 30, 2023

Contents

1 Abstract

This final report chronicles Lucia Eve Berger's internship project to complete her Diplôme d'Études Supérieures Spécialisées (DESS) at Universite de Montreal in Machine Learning. She conducted her internship with Microsoft(Nuance) in 6 months. Her internship spanned from June to November 2023.

She was supervised by Dermot Connolly. Dermot Connolly is a manager of research. The topic of her internship project was Minimum Word Error Rate Training, a technique to reduce Word Error Rate in Automatic Speech Recognition systems.

2 Motivation

Automatic Speech Recognition (ASR) is the process of transcribing audio to text. ASR is used in many applications including voice assistants, voice command systems, and other devices. ASR systems take in audio input and extract relevant features, such as acoustic characteristics and spectral information, to represent the speech signal effectively. ASR technology has advanced significantly over the years. This is in part to improvements in deep learning techniques and widespread availability of large datasets. Modern ASR systems use deep neural networks, such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs), to improve accuracy. Despite large improvements in ASR, there are still challenges in accuracy, particularly with out-of-domain vocabulary. In other words, ASR systems may struggle with words or terms that are not in their training data. This can be a problem for specialized or domain-specific applications, such as Interactive Voice Response (IVR) systems.

Minimum Word Error Rate Training has been used to train attention-based seq2seq models and has shown improvements with Conformer RNNTs. This report focuses on gains in the Human-to-Machine Domain. This internship report chronicles a technique for an ASR system refinement. Minimum Word Error Rate training loss performs discriminative training between hypothesizes. While other researchers at Microsoft (Nuance) had previously experimented with this technique, I integrated this training methodology in the our codebase in tensorflow and later pytorch. This internship project involved deep learning, coding and experimentation.

3 Context/Issue

Nuance Communications is a voice-technology company, based out of Burlington, Massachusetts. In 2021, Nuance Communications was acquired by Microsoft. I performed this internship with Microsoft. My job title at Microsoft is Research Scientist

Nuance technology spans healthcare technology, financial services, telecom and government. The company provides solutions that enable businesses and organizations to create more human-like interactions between people and technology.

Nuance provides various enterprise solutions, including voice biometrics for security, virtual assistants for business applications, and dictation software. The internship focused on implementing Minimum Wer training. The objective was to integrate the deep-learning feature into the Microsoft python training pipeline. The duration of this internship was 6-months. Lucia Eve Berger participated and was aided by several members of her team.

4 Timeline

The internship was conducted over a period of six months. In the first weeks of the internship, I worked on training and onboarding to the project. This included formal training sessions on relevant technologies, programming languages, and frameworks. My colleague gave me an introduction to the existing codebase and ongoing projects.

Following this, I started the initial project work. I conducted software design, memory and speed profiling. I participated in pair-programming sessions. For the internship project task, I had regular code reviews and feedback sessions. Gradually, the complexity of assigned tasks increased. Finally, I reported on my work in a objectives and key results (OKR) meeting. I presented my work weekly and had 1-on-1s with my supervisor twice-a-week. At the conclusion of the project, the code was merged for other researchers to use.

The culmination of my efforts was showcased in a meeting where I delivered a comprehensive report on the accomplished tasks and their alignment with set objectives. The completion of the internship project marked a significant milestone as the code was successfully integrated and merged. This also marks a successful application of a research project.

5 Introduction

5.1 Automatic Speech Recognition

The primary task of Automatic Speech Recognition (ASR) involves converting spoken language into written text. This technology is used in applications like transcription services and voice assistants. With ASR, traditionally statistical, language and other models were used to complete this task. As Figure 1 shows, Acoustic Models, Pronunciation Models and Language Models were built to conduct single language translation.

As of late, end-to-end models have come into prevalence. An end-to-end approach means creating a model that directly takes raw input data and generates the desired output, without relying on separate pre-processing or intermediate stages. Figure 1 shows the evolution from traditional modelling to end-to-end. We can see the final outcome is a single model for all needs.

In particular, Conformer RNNT models have state-of-the-art performance in the speech domain. The base model used is Conformer RNNT. The term *Conformer* in the model's name refers to the architecture's use of a specific type of neural network module known as the *Conformer* module. RNN-T stands for ”Recurrent Neural Network Transducer,” which is a type of ASR architecture that combines elements of recurrent neural networks (RNNs) and transformers. The Conformer module is a critical component of this architecture. It is inspired by the transformer architecture, which has been highly successful in various natural language processing tasks. The Conformer module combines self-attention mechanisms with convolutional layers and feedforward networks to capture both temporal and contextual information in speech data.

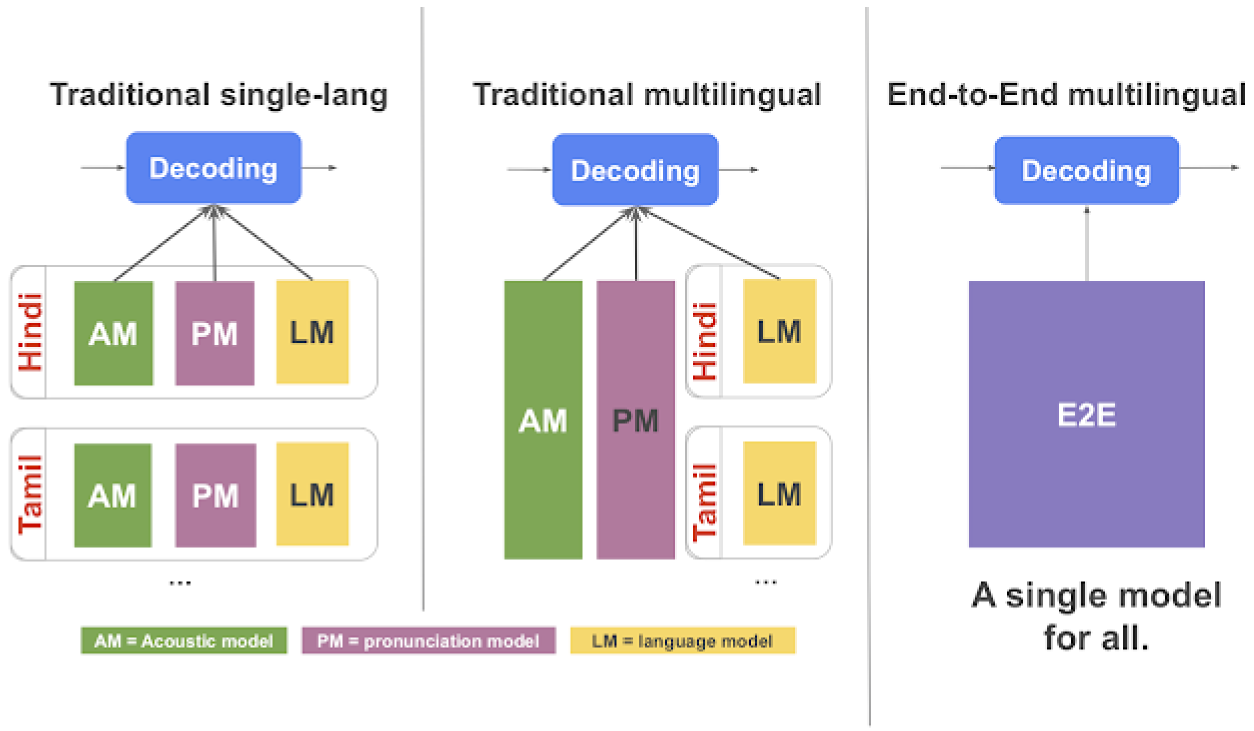


Figure 1: Large-Scale Multilingual Speech Recognition with a Streaming End-to-End Model [1]

The RNN-T part of the model stands for Recurrent Neural Network Transducer. RNN-T is an ASR architecture that uses both recurrent neural networks (RNNs) and transformers to recognize and transcribe speech. RNNs are often used to model temporal dependencies in sequential data, while transformers are excellent at capturing contextual information across sequences. The Conformer RNN-T model leverages the strengths of both the Conformer module and the RNN-T architecture to improve the accuracy and efficiency of ASR. By using the Conformer module, it can capture long-range dependencies and contextual information in the audio signal, which is crucial for understanding spoken language. The

RNN-T component helps to model sequential aspects and improves alignment between the audio and the transcription.

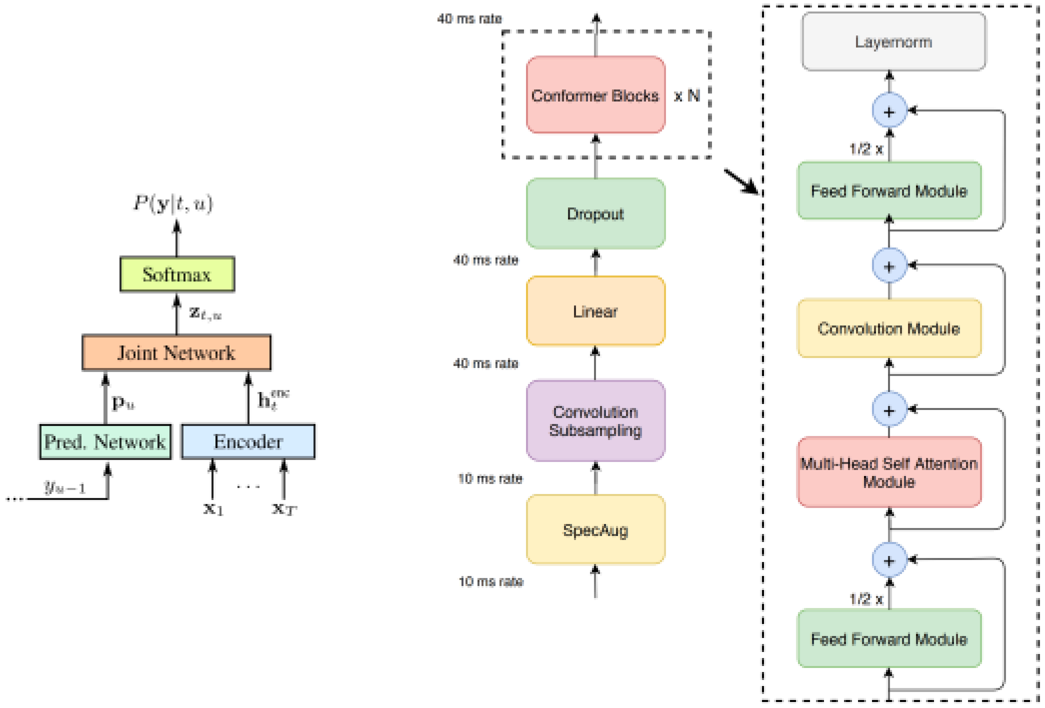


Figure 2: Conformer RNN-T Architecture [2].

Figure 2 depicts a Conformer RNNT Architecture. Microsoft (Nuance) researchers experiment with variants of this model, changing the Conformer blocks as as the depth and width of each layer. The Conformer RNNT is a popular choice for many in industry. It is considered state-of-the-art due to its attention mechanisms, efficient modeling of long sequences and generalization. They can be fine-tuned to meet the needs of different ASR tasks.

It should be noted the work in this paper focuses on streaming ASR models. Such models are designed to process and transcribe audio input into text almost instantly as the audio is being spoken, providing a continuous stream of text output. This is in contrast to offline models.

5.2 Evaluation of Automatic Speech Recognition

With modelling technology advancing quickly, it is important for researchers to have a common metric for measuring ASR performance. Word Error Rate (WER) is frequently used. WER is used to evaluate the accuracy of the ASR system's transcriptions by measuring the difference between the recognized words and the reference words. The reference is also called the ground-truth. A lower WER indicates better accuracy in the ASR system because it represents a smaller difference between the ASR output and the reference transcription.

6 Algorithm

6.1 Background and Inspiration

Minimum Word Error Rate has been used to train attention-based sequence-to-sequence models. After watching the 2020 Interspeech presentation, I was inspired by the paper "Efficient Minimum Word Error Rate Training of RNN-Transducer for end-to-end-speech recognition"[3]. This paper accomplished a similar task, using the RNNT-base model and loss. This paper also achieved competitive gains. As part of my internship project, I did a paper review of this, familiarizing myself with other researchers work. The transducer loss measures the discrepancy between the model's predictions and the ground truth labels, considering both the insertion and deletion errors. The transducer loss is often computed using the forward-backward algorithm. $P(Y|X)$ is the conditional probability of the target sequence given the input sequence.

The algorithm has three parts:

1. On-the-fly-decoding

2. RNNT Logits Generation

3. Computation of Loss

This report describes each aspect of the algorithm in the below sections.

6.1.1 On-the-fly-decoding

On the fly decoding generates alignment scores for the n-best predictions. These are text-based predictions the foundation model gives us. We can call these alignment scores the edit-distance. In other words, the edit distance is finding the minimum number of operations required to transform one tensor into another.In this implementation, I used the Levenshtein distance algorithm.

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j)=0 \\ \min\begin{cases} \operatorname{lev}_{a,b}(i-1,j)+1 \\ \operatorname{lev}_{a,b}(i,j-1)+1 \\ \operatorname{lev}_{a,b}(i-1,j-1)+\mathbf{1}_{(a_i \neq b_j)} \end{cases} & \text{otherwise} \end{cases}$$

Where: $\operatorname{lev}_{a,b}(i,j)$ represents the Levenshtein distance between prefixes of lengths i and j of strings a and b, respectively. a_i and b_j represent the ith and jth characters of strings a and b. $\mathbf{1}_{(a_i \neq b_j)}$ is an indicator function that equals 1 if $a_i \neq b_j$ and 0 otherwise.

I represented the edit distance as a float, with a lower value 0.0 indicating no errors. A higher value denotes a considerable amount of difference between the ground-truth sequence and the predicted sequence.

6.1.2 Use the RNNT Network to generate the logits

Logits are the output of the network for the RNNT given audio and labels. The logits have the shape [B, T, U, V]. B represents the batch size. It signifies the number of sequences or utterances processed simultaneously during training or inference. Each sequence in the batch can correspond to different audio segments or utterances.T denotes the maximum time steps or sequence length in the audio features. It indicates the temporal dimension of the input audio data. U represents the maximum label sequence length. This dimension is associated with the length of the label sequences or transcripts corresponding to the audio. It defines the maximum number of labels the model predicts for a given audio input. V signifies the number of units or classes in the output vocabulary. It represents the number of distinct symbols or categories that the model can predict.

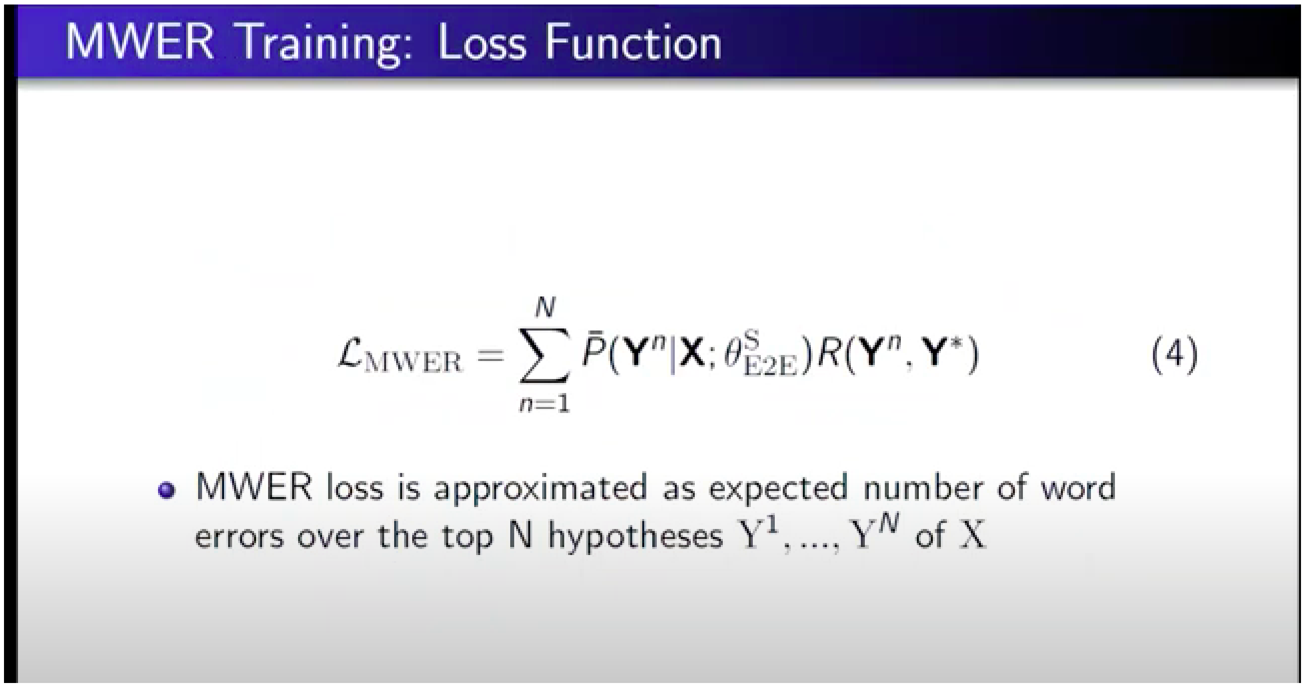
We need to pass the shape of the n-best logits to compute it for each of the n-best list. Given the n-best as inputs to the prediction network, we can generate the output of the RNNT model where x is the audio sequence and y_i is the decoding hypothesis. Traditionally, we'd pass this to $\log P(y_i \mid x)$. In our algorithm, we feed the hypothesis y_i back into the prediction network. This allows us to generate all the possible alignments of y_i given x. [3]

6.1.3 Compute the MWER Error Rate

After getting the original logits and logits given the n-best (B*N,T, U, vocab), we can compute the both normal RNNT loss and MWER loss. The MWER loss has two parts.

One of the risk function, that which we calculated in step 1. We also refer to this as the edit-distance.

The second part is the normalized probabilities per hypothesis. On Figure 3, we can see this on the right hand portion. This is "Gamma", γ. To extract gamma, you have to normalize logprob, or get the probabilities of the rnn-T loss, and make them sum up to one for a given sample. The Interspeech 2020 presentation gave excellent detail. [3]

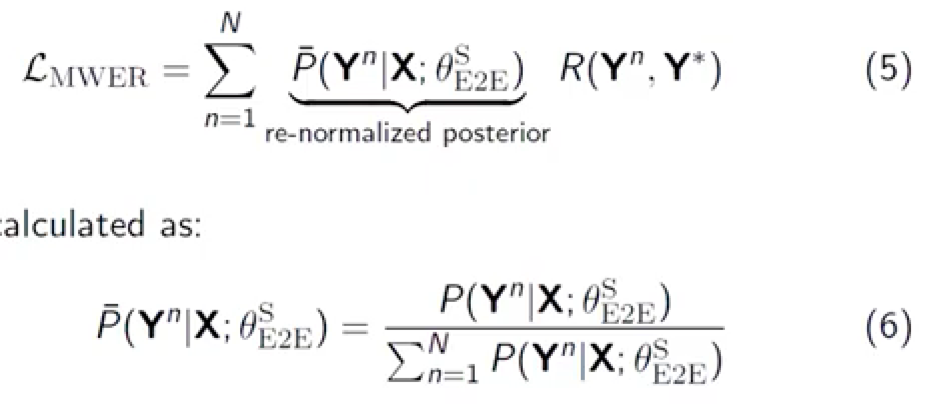


MWER Training: Loss Function

$$\mathcal{L}_{\mathrm{MWER}} = \sum_{n=1}^{N} \bar{P}(\mathbf{Y}^n|\mathbf{X};\theta^{\mathrm{S}}_{\mathrm{E2E}})R(\mathbf{Y}^n,\mathbf{Y}^*) \quad (4)$$

- MWER loss is approximated as expected number of word errors over the top N hypotheses $\mathrm{Y}^1, ..., \mathrm{Y}^N$ of X

Figure 3: Interspeech 2020: Step 0

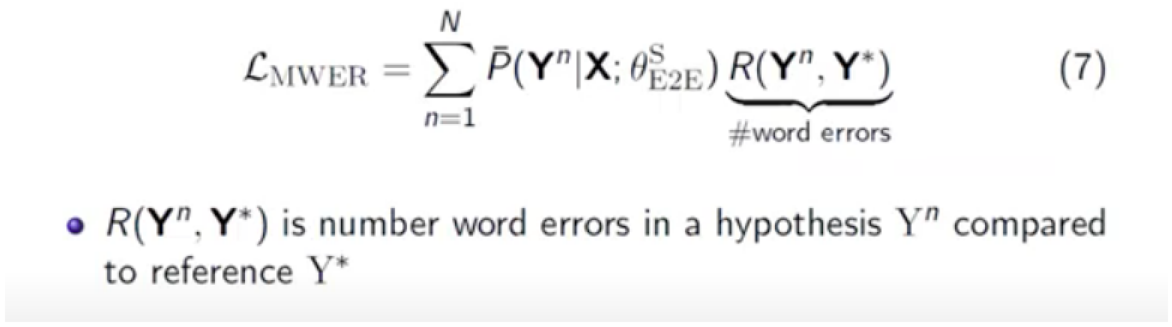


$$\mathcal{L}_{\text{MWER}} = \sum_{n=1}^{N} \underbrace{\bar{P}(\mathbf{Y}^n|\mathbf{X};\theta_{\text{E2E}}^{\text{S}})}_{\text{re-normalized posterior}} \; R(\mathbf{Y}^n,\mathbf{Y}^*) \qquad (5)$$

- $\bar{P}$ is calculated as:

$$\bar{P}(\mathbf{Y}^n|\mathbf{X};\theta_{\text{E2E}}^{\text{S}}) = \frac{P(\mathbf{Y}^n|\mathbf{X};\theta_{\text{E2E}}^{\text{S}})}{\sum_{n=1}^{N} P(\mathbf{Y}^n|\mathbf{X};\theta_{\text{E2E}}^{\text{S}})} \qquad (6)$$

Figure 4: Step 1: Renormalized Posterior



$$\mathcal{L}_{\text{MWER}} = \sum_{n=1}^{N} \bar{P}(\mathbf{Y}^n|\mathbf{X};\theta_{\text{E2E}}^{\text{S}}) \underbrace{R(\mathbf{Y}^n,\mathbf{Y}^*)}_{\#\text{word errors}} \qquad (7)$$

- $R(\mathbf{Y}^n,\mathbf{Y}^*)$ is number word errors in a hypothesis Y^n compared to reference Y^*

Figure 5: Step 3: Combined Loss

Finally, we come to our output which is the expected number of word errors within the n-best list. [3]

$$\hat{R} = \sum_{y_j \in \text{n-best}(x)} \hat{P}(y_j|x) R(y_j, y_r) \tag{2}$$

To help balance the variance, we also subtracted $\hat{R}$ from R also helps to reduce the variance of gradients. Following the advice of the paper, we reason:

1. Considering that $\sum_i \hat{P}(y_i|x) = 1$, when examining hypotheses having fewer errors than $\hat{R}$, results in $R(y_i, y_r) - \hat{R} < 0$, thus leading to an enhancement in $\log P(y_i|x)$.

2. For hypotheses surpassing $\hat{R}$ in errors, their probability diminishes. In contrast to the RNN-T loss, which solely focuses on elevating the likelihood of the ground-truth transcription, the Minimum WER loss engages in discriminative training among hypotheses, aiming for a more refined differentiation between them.

7 Data Representation

7.1 Background

For the minWER task, we need to prepare the data for training. The raw data is audio (mp3) with a text annotation. Often, the annotations are provided by transcribers and are representative of the ground truth. The label is in word pieces. For example, the word ”unhappiness” might be divided into subword tokens like ”un,” ”happ,” and ”iness.” When we tokenize this text, we can represent this as an array [34, 102, 135].

The data itself was represented as a a feature, a tensor. Additionally, it has information about it’s label. The audio itself is represented as tensor of mel filter banks. We represent audio this way to reduce dimensionality. Mel filter banks are typically much smaller in dimension compared to the original spectral data (e.g., Fourier transform magnitude spectrum or spectrogram). This reduction in dimensionality simplifies the data, making it more computationally efficient to process and reducing the risk of overfitting in machine learning models.

These are also machine-friendly and compatible with our training setup. By converting audio into a tensor, it becomes compatible with these models, enabling them to learn complex patterns and features from the data. As discussed previously, our model consists of layers like convolutional layers, recurrent layers, and fully connected layers, which expect input data in tensor format. These layers perform operations such as convolution, pooling, and recurrent processing, which require multi-dimensional data. Figure 6 depicts the machine representation of our data with information about the total frames, total frames

```
{'utt_ids': ['c-38a2d578c8eb_0.wav'],  'sweep': 1.0,
'total_frames': 342, 'total_frames_with_padding': 342,

'feature': tensor([[[1.7389, 4.3355, 5.5984,  ..., 0.0000, 0.0000,
   0.0000],
'feature_lens': tensor([342]), 'label': tensor([[1607,    0, 1483, 1376,
   1606]]), 'label_lens': tensor([5]),

'dl_block_metrics': OrderedDict([('IIDListSampler0__total_time',
   0.0025589466094970703), ('IIDListSampler0__total_calls', 42), ('
   BinaryChunkDeserializer0__total_time', 1.1449501514434814),...}
```

Figure 6: Python representation of our data

with padding and the feature-lens.

8 Minimum Wer Training Procedure

8.1 High Level Implementation

For this internship project, I developed and added the Minimum Word Error Rate (MWER) training module to two internal code bases. Both of these code bases are python based, Automatic Speech Training toolkits. One is written with the pytorch module and that will be the focus of this report. This internship project involved code, system design and Machine Learning pipelines.

8.2 Software Design

As the MWER training module uses the traditional RNNT loss, I wrote a new class TransducerModelMinWER which inherited the necessary properties from the Transducer-Model. We uses a super overriding function to take the properties of the original loss.

I added a new functionality to the training forward look. As the graphic below shows, the code

1. Loads the decoder: The decoding step is a critical part of the ASR pipeline because it bridges the gap between the ASR model's output, which is a sequence of tokens, and the intended human-readable text. By incorporating language models and search algorithms, it helps improve the overall quality and fluency of the transcriptions. Decoding is also important for handling challenges like out-of-vocabulary words, disfluencies in speech, and language-specific variations. By using in decoding in training,

we are effectively boosting predictions.

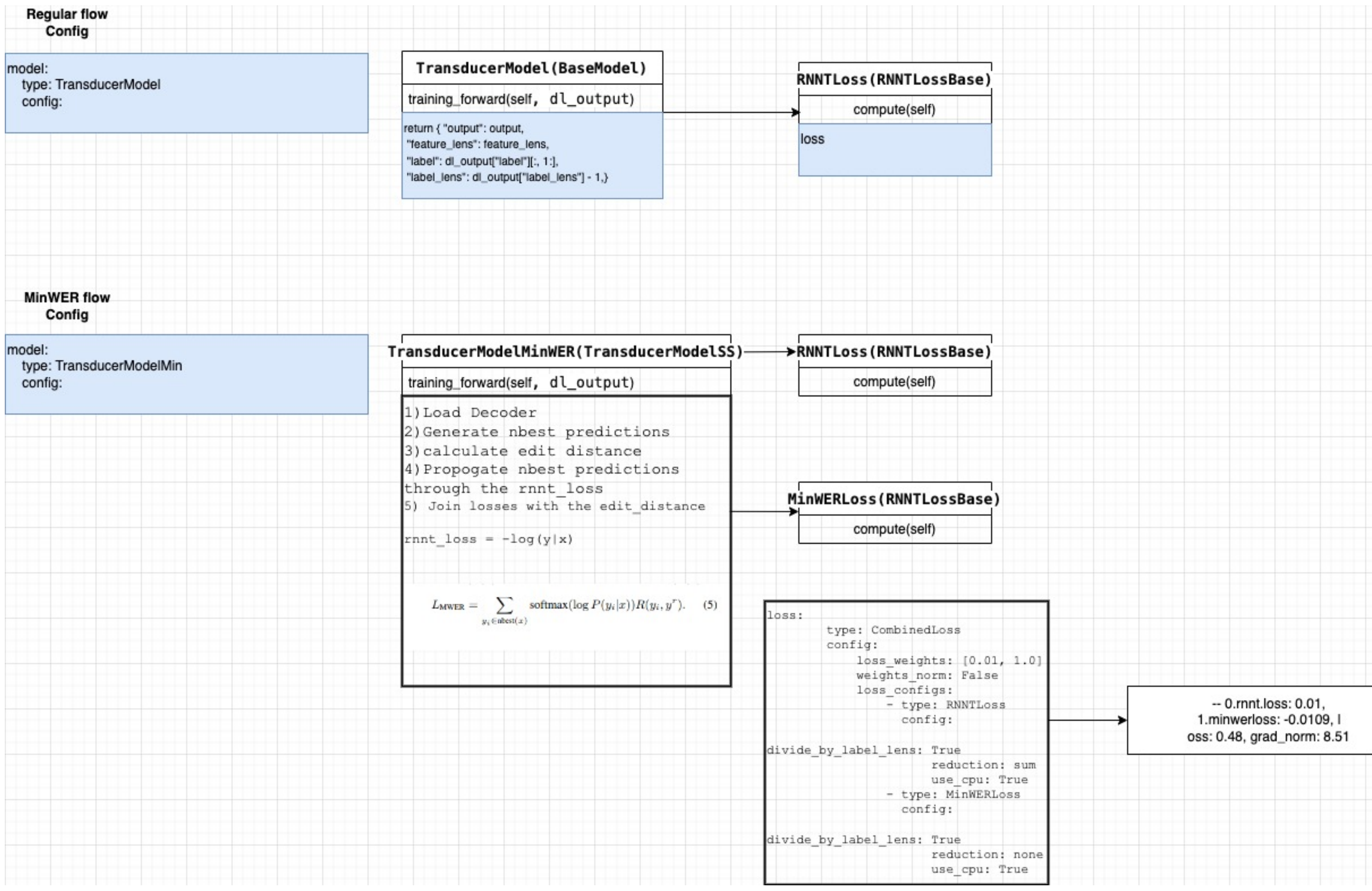
2. Generate the n-best predictions: The number of n best is a hyperparameter. An n-best list refers to a list of the top n hypotheses generated by the ASR system for a particular speech input. The highest-ranked hypothesis in the n-best list is considered the most probable transcription of the spoken input. However, exploring the n-best list can be valuable for various purposes to improve transcription accuracy by considering alternative possibilities.

3. Calculates the edit distance: the edit distance is a measure of similarity between the sequences, representing the minimum number of operations required to transform one sequence into another. I experimented with several different algorithms in calculating the edit distance.

4. Propagation the n-best predictions through the rnnt loss and joins the losses with the edit distances: the process of such is described in the algorithm section of this report.

8.3 Implementation in Cascades (Internal Codebase)

As I was working with a new codebase, I need to first understand the software flows. I analyzed a traditional training recipe. I ran training and then I ran inference. This also gave me a baseline for what the training trajectory looked like.



```
model:
   type: TransducerModel
   config:
```

```
return { "output": output,
 "feature_lens": feature_lens,
 "label": dl_output["label"][:, 1:],
 "label_lens": dl_output["label_lens"] - 1,}
```

```
model:
   type: TransducerModelMin
   config:
```

```
1)Load Decoder
2)Generate nbest predictions
3)calculate edit distance
4)Propogate nbest predictions
through the rnnt_loss
5) Join losses with the edit_distance

rnnt_loss = -log(y|x)
```

$$L_{\text{MWER}} = \sum_{y_i \in \text{nbest}(x)} \text{softmax}(\log P(y_i|x))R(y_i, y^r). \quad (5)$$

```
loss:
        type: CombinedLoss
        config:
            loss_weights: [0.01, 1.0]
            weights_norm: False
            loss_configs:
                - type: RNNTLoss
                  config:

divide_by_label_lens: True
                      reduction: sum
                      use_cpu: True
                - type: MinWERLoss
                  config:

divide_by_label_lens: True
                      reduction: none
                      use_cpu: True
```

```
-- 0.rnnt.loss: 0.01,
1.minwerloss: -0.0109, l
oss: 0.48, grad_norm: 8.51
```

Figure 7: Software Diagram of MinWER

8.4 Combing the Losses

The MWER training is an interpolation of losses. This means it combines the losses with their respective weights. We use a class to manage this for us. Interpolating losses allows us to create custom loss functions by combining multiple components to better reflect the underlying problem's complexity.

8.5 Challenges faced in the code implementation

I faced three large challenges in this implementation. These were the choice of decoder, speed and memory. I also worked on maintaining a CPU and GPU implementation of this code.

8.5.1 Choice of decoder

As we only had used decoding in inference, we need to decide which decoder to use in this training module. The decoders out of box

- TransducerBeamSearchGenerator
- TransducerBeamSearchSimpleGenerator
- Online C++ BeamSearchGenerator

8.5.2 Speed

As I had a relatively large dataset of 14k hours of speech data, I monitored the speed of each decoder. I wrote code to profile. I used the azure machine learning toolkit to measure

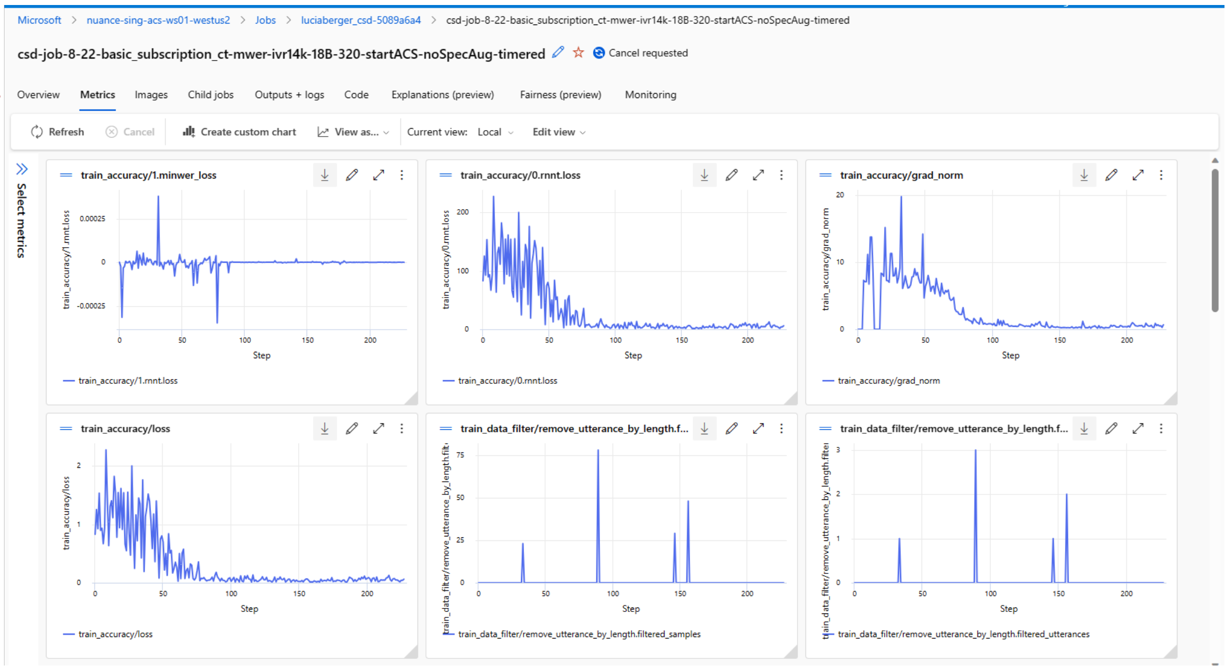

Microsoft > nuance-sing-acs-ws01-westus2 > Jobs > luciaberger_csd-5089a6a4 > csd-job-8-22-basic_subscription_ct-mwer-ivr14k-18B-320-startACS-noSpecAug-timered
csd-job-8-22-basic_subscription_ct-mwer-ivr14k-18B-320-startACS-noSpecAug-timered
Cancel requested
Overview
Metrics
Images
Child jobs
Outputs + logs
Code
Explanations (preview)
Fairness (preview)
Monitoring
Refresh
Cancel
Create custom chart
View as...
Current view: Local
Edit view
Select metrics
train_accuracy/1.minwer_loss
train_accuracy/1.rnnt.loss
train_accuracy/0.rnnt.loss
train_accuracy/grad_norm
train_accuracy/loss
train_data_filter/remove_utterance_by_length.f...
train_data_filter/remove_utterance_by_length.filtered_samples
train_data_filter/remove_utterance_by_length.filtered_utterances
Step

Figure 8: Azure Machine Learning Dashboard

these parameters. Figure 8 shows how I conducted the profiling.

As the speed profiling table shows above, the out of box decoder was far too slow to use in training. The original non-MWER training ran at 30,000 samples/second. Using the inference based decoding, this dropped to 500 samples/second. This would be impossible to conduct iterative experimentation.

Respectively, I modified the approach and used an online decoder. An online decoder refers to the real-time processing component responsible for converting spoken language into text as the speech is being received. It operates by continuously processing incoming audio segments and making predictions about the words or phrases being spoken. All of these provide accurate decoding results, but run 30x slower than the regular training. The python modules are not optimized for in-training decoding. It should be noted there is

Table 1: Speed Profiling

	Training timing/samples/second (Units)
Baseline	30,000 samples/second
MinWER + decoderOffline Y	500 samples/second
MinWER + decoderOffline Z	800 samples/second
MinWER no decoding	25,000 samples/second
MinWER + Optimised decoding	12,000 samples/second

work ongoing to improve their performance. We found using a batched streaming decoding strategy (c++ module) gave us appropriate speed. I needed build the module in the cascade project and debug it.

Finally, we reached an acceptable setup for training six epochs. We maintained the same 800k sample batch. This experiment takes 38 hours to run. For us, this was an acceptable time for running experiments and iterating through different setups.

8.5.3 Memory

With four n-bests predictions at each utterance, we are propagating a larger sample of predictions through the network. If we try to define the joint network, we will run out of memory.

We use a subsetting strategy in which we iterate through the hypothesis, in this way, we can use the same batch size, just processes the large batches in ”mini-batches”. We can maintain the same batch size.

```
hypothesis_lengths_mod = hypothesis_lengths.view(encoder_output.shape[0],
                                                                n-best)
hypothesis_= hypothesis.view(encoder_output.shape[0], n-best, hypothesis.
    shape[1]
loss_out = []
for i in range(n-best):
     first_rows = hypothesis_[:, i, :]
     first_column = hypothesis_lengths_mod[:, i]
     decoder_embedding_output = self.decoder_embedding(first_rows)
     decoder_output, *_ = self.decoder(
                         decoder_embedding_output, **self.decoder_args_fn(
    dl_output)
                    )
     labels = first_rows[:, 1:]
     label_lens = first_column - 1
     total_batches = encoder_output.shape[0]
     ss_size = total_batches // (self.subset_size)
     remains = total_batches % (self.subset_size) if ss_size > 0 else
    total_batches
     start = 0
     for i in range(self.subset_size):
         // propogate our losses
```

The code above shows how we can split the memory up and use a divide-and-conquer approach. We will want to maintain our batch size to ensure diversity in our samples passed in training.

8.6 Training and Evaluation

As Minimum WER Training is a tuning technique, a foundation model needs to be defined and trained before applying it. This internship project used two derivations of the foundation model.

- Model A) Simplified Model trained only with IVR Data.
- Model B) State-of-the-Art trained first with all Human-to-Human data and tuned with 14k IVR data.

The focus of this report will be the Model A case. I trained a base model with 14,000 (14k) hours of IVR, human to machine data. Both of the foundation model were trained using an inhouse dataset, a large corpus of many different domains. In Model B, the model was fine-tuned with a smaller IVR specific dataset to enhance capacity. Two foundation models were employed to demonstrate the capabilities of the tuning and then to use it in a real application. It was decided to test on these two model configurations as Model A demonstrated the feature worked.

The base model is trained using the RNNT loss. The RNNT loss is the Recurrent Neural Network Transducer Loss. RNNT loss handles tasks where the alignment between the input and output sequences is not one-to-one, meaning that the lengths of the input and output sequences may differ, and the model needs to learn to align and transcribe the input sequence into the output sequence.

In the RNNT loss, the model produces two sequences: a target sequence (the ground truth) and a prediction sequence. The loss function measures the dissimilarity between these two sequences, taking into account variable alignments. It encourages the model to align the input sequence correctly and generate the correct output sequence.

After solving the challenges discussed above, I submitted my jobs to a 8-GPU cluster. At times, my jobs needed to queue. I often submitted for processing over the weekend. I monitored the training using a devset. A dev sev, short for development set, refers to a portion of a dataset that is used during the development and fine-tuning of machine learning models, algorithms, or systems. It is also known as a validation set or validation data.

Using the ModelA configuration, I observed immediate gains from the baseline.

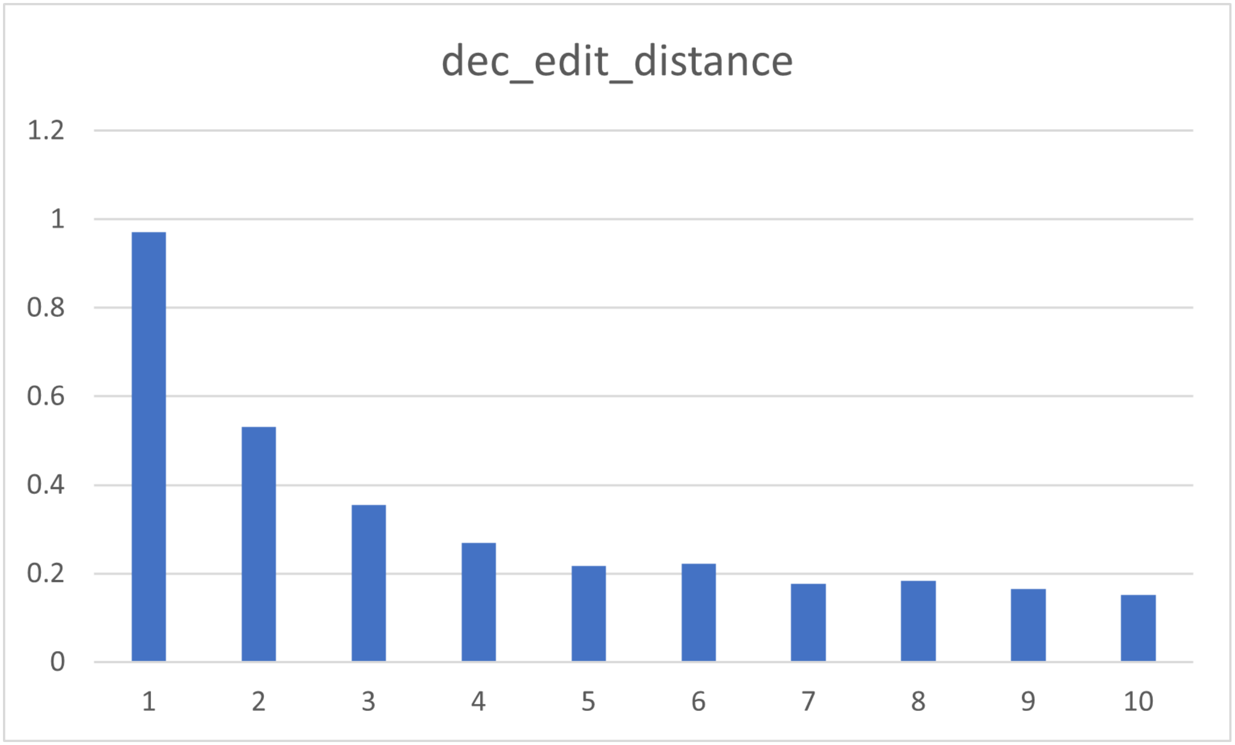


Figure 9: Model A: Dev Edit Distance

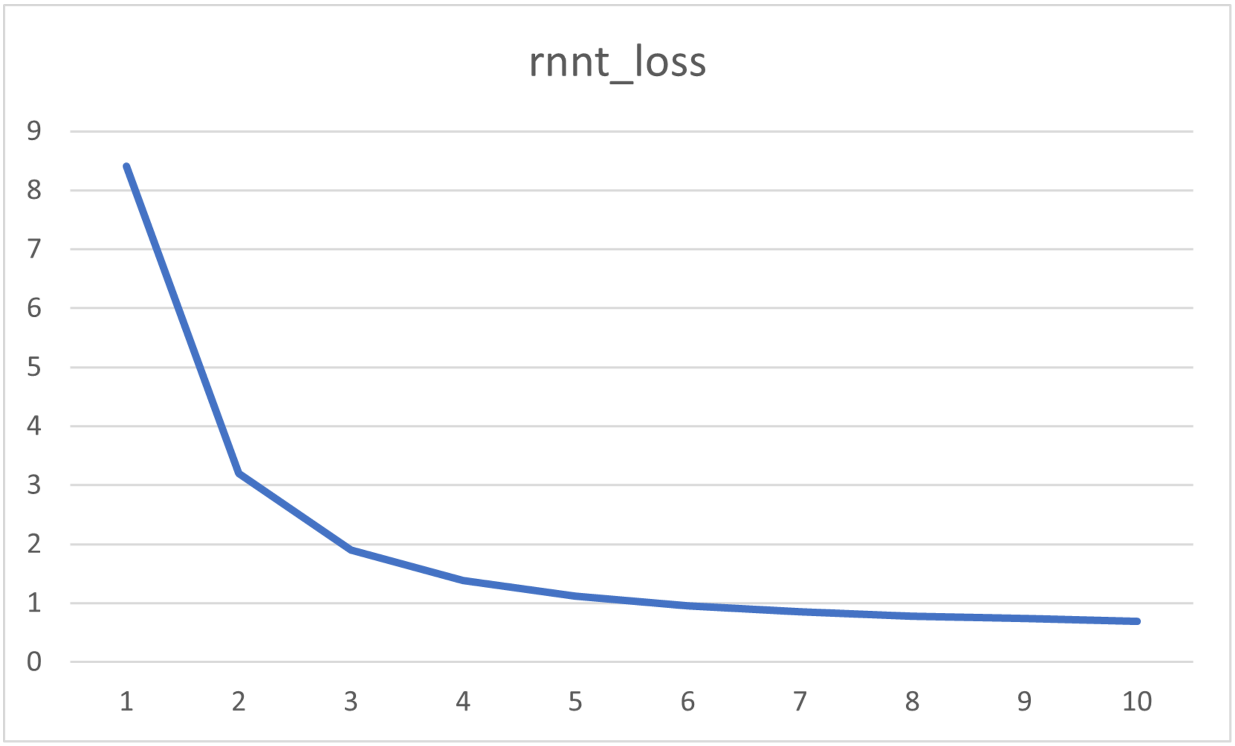


Figure 10: Model A: Dev Edit Distance

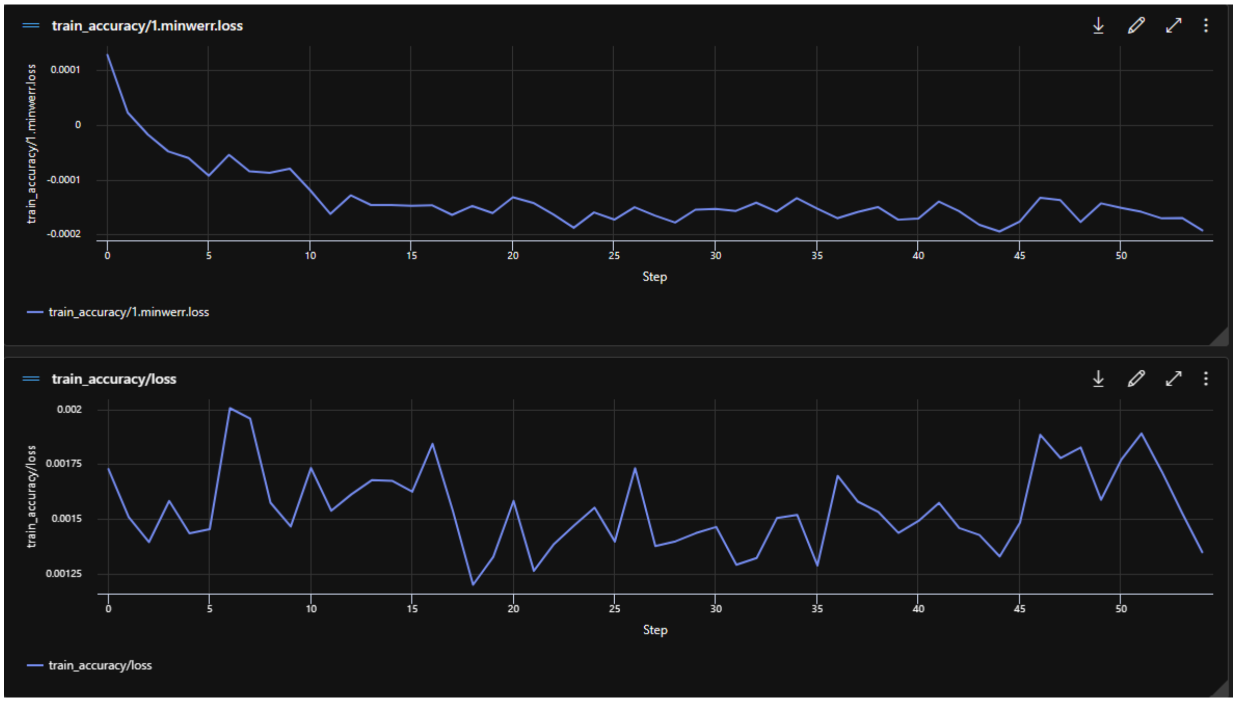
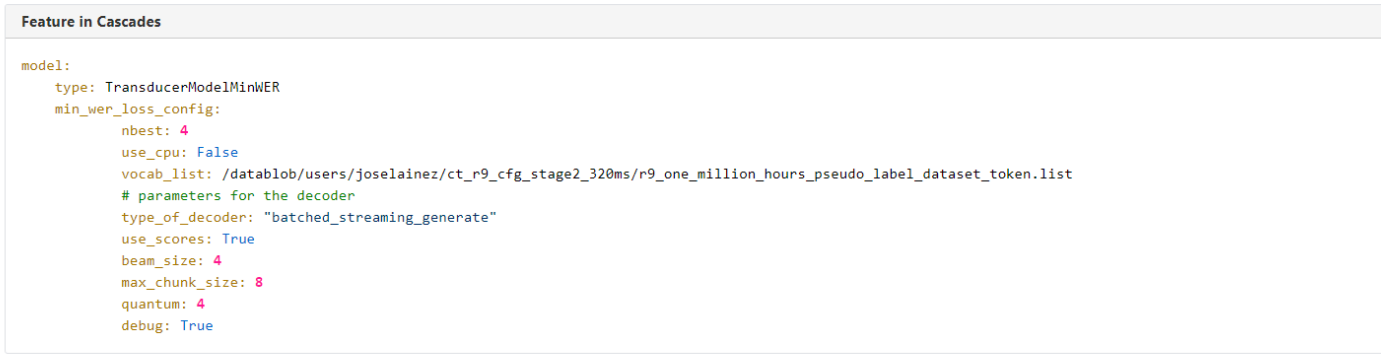


Figure 11: Model A: Training Loss Formalized in Azure Training

8.7 How other users can use this feature



Feature in Cascades

```
model:
    type: TransducerModelMinWER
    min_wer_loss_config:
            nbest: 4
            use_cpu: False
            vocab_list: /datablob/users/joselainez/ct_r9_cfg_stage2_320ms/r9_one_million_hours_pseudo_label_dataset_token.list
            # parameters for the decoder
            type_of_decoder: "batched_streaming_generate"
            use_scores: True
            beam_size: 4
            max_chunk_size: 8
            quantum: 4
            debug: True
```

We controlled our model setup with a YAML file. The researcher could vary the foundation model, data and hyperparameter tunings. YAML files are versatile and allow for flexibility in specifying various types of data. YAML configuration files can be version-controlled using tools like Git. This means that changes to the configuration, hyperparameters, or experiment settings can be tracked over time, enhancing reproducibility and collaboration in machine learning projects. We published our best model YAML as a starting point for others.

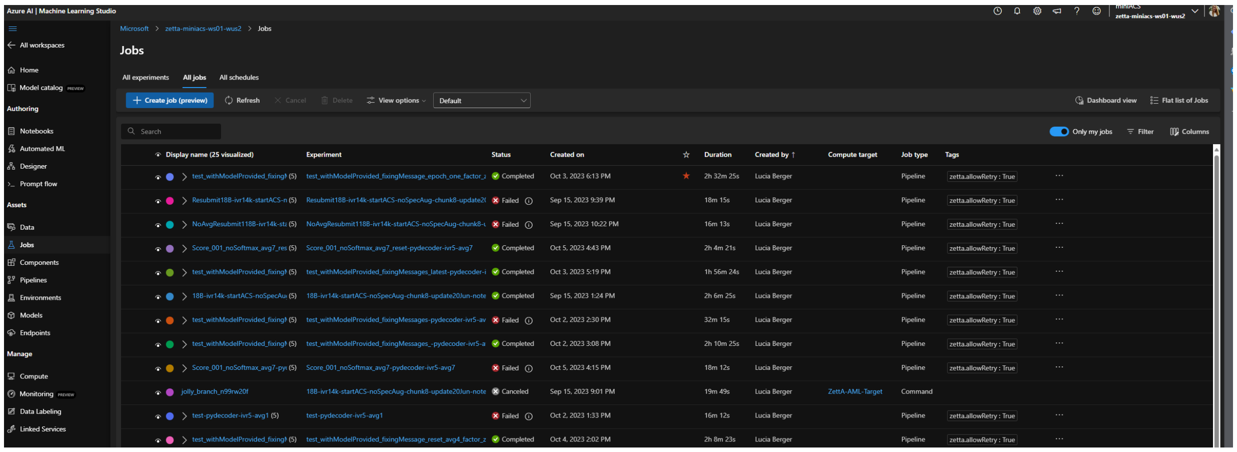
8.8 Testing

Evaluating the model on testdata was an essential part of the procedure. I used the Azure ML datakit to do this. I used the below procedure:

1. Download the logs
2. Grep the logs to find model location
3. If model averaging is required, download the models. Conduct the averaging on the downloaded model.
4. Copy the averaged model back to the datastore.

As everything was done on a cloud service, this had a number of benefits for us. The clear benefit was keeping dedicated storage of all of our testing experiments. Cloud servers typically offer robust backup and disaster recovery solutions. They automatically backup data and provide options for creating redundant copies in different geographical locations, ensuring data resilience in case of unexpected events.

In addition, I could easily share the findings of my experiments with colleagues. We all shared the same test dataset, use a published baseline and we able to easily identify if the method in question was helpful. This organized structure allowed me to conduct efficient hyperparameter tuning. I used the Azure ML to store and run my jobs, shown in the Figure below.



Jobs

Display name (25 visualized)	Experiment	Status	Created on	☆	Duration	Created by ↑	Compute target	Job type	Tags
test_withModelProvided_fixingM (5)	test_withModelProvided_fixingMessage_epoch_one_factor_z	Completed	Oct 3, 2023 6:13 PM	★	2h 32m 25s	Lucia Berger		Pipeline	zetta.allowRetry : True
Resubmit18B-ivr14k-startACS-n (5)	Resubmit18B-ivr14k-startACS-noSpecAug-chunk8-update2(	Failed	Sep 15, 2023 9:39 PM		18m 15s	Lucia Berger		Pipeline	zetta.allowRetry : True
NoAvgResubmit118B-ivr14k-sta (5)	NoAvgResubmit118B-ivr14k-startACS-noSpecAug-chunk8-u	Failed	Sep 15, 2023 10:22 PM		16m 13s	Lucia Berger		Pipeline	zetta.allowRetry : True
Score_001_noSoftmax_avg7_res (5)	Score_001_noSoftmax_avg7_reset-pydecoder-ivr5-avg7	Completed	Oct 5, 2023 4:43 PM		2h 4m 21s	Lucia Berger		Pipeline	zetta.allowRetry : True
test_withModelProvided_fixingM (5)	test_withModelProvided_fixingMessages_latest-pydecoder-i	Completed	Oct 3, 2023 5:19 PM		1h 56m 24s	Lucia Berger		Pipeline	zetta.allowRetry : True
18B-ivr14k-startACS-noSpecAu (5)	18B-ivr14k-startACS-noSpecAug-chunk8-update20Jun-note	Completed	Sep 15, 2023 1:24 PM		2h 6m 25s	Lucia Berger		Pipeline	zetta.allowRetry : True
test_withModelProvided_fixingM (5)	test_withModelProvided_fixingMessages-pydecoder-ivr5-av	Failed	Oct 2, 2023 2:30 PM		32m 15s	Lucia Berger		Pipeline	zetta.allowRetry : True
test_withModelProvided_fixingM (5)	test_withModelProvided_fixingMessages_-pydecoder-ivr5-a	Completed	Oct 2, 2023 3:08 PM		2h 10m 25s	Lucia Berger		Pipeline	zetta.allowRetry : True
Score_001_noSoftmax_avg7-pyc (5)	Score_001_noSoftmax_avg7-pydecoder-ivr5-avg7	Failed	Oct 5, 2023 4:15 PM		18m 12s	Lucia Berger		Pipeline	zetta.allowRetry : True
jolly_branch_n99rw20f	18B-ivr14k-startACS-noSpecAug-chunk8-update20Jun-note	Canceled	Sep 15, 2023 9:01 PM		19m 49s	Lucia Berger	ZettA-AML-Target	Command	
test-pydecoder-ivr5-avg1 (5)	test-pydecoder-ivr5-avg1	Failed	Oct 2, 2023 1:33 PM		16m 12s	Lucia Berger		Pipeline	zetta.allowRetry : True
test_withModelProvided_fixingM (5)	test_withModelProvided_fixingMessage_reset_avg4_factor_z	Completed	Oct 4, 2023 2:02 PM		2h 8m 23s	Lucia Berger		Pipeline	zetta.allowRetry : True

Figure 12: Evaluation Log

9 Hyperparameter Tuning

9.1 HyperParameter Tuning

Hyperparameter tuning is a crucial step in machine learning model development. It involves finding the best set of hyperparameters for a given machine learning algorithm that optimizes the model's performance. Hyperparameters are configuration settings for an algorithm that are not learned during training but significantly impact the model's performance and behavior. In ASR, we have training parameters and inference parameters. The inference parameters are the decoding parameters, or how we process the decoding output. Initially, we saw modest gains on test and our dev set. However, I made effort to tune the interpolation factors.

$$\text{loss} = \alpha \cdot \text{rnnt\_loss}(\text{logits\_target}, \text{target}) + \beta \cdot \text{mwer\_loss}(\text{logits\_nbest}, \text{nbest}, \text{distance}(\text{nbest}, \text{target})) \tag{3}$$

As the above equation shows, the loss is a combination of the traditional rnnt and minimum wer loss.

I selected a representative sample (10% of the data) and performed experimentation to find the best hyperparameters. The three hyperparameters I focused on were:

1. Learning Rate

2. Alpha

3. Beta

4. Number of epochs/Early stopping

The learning rate tuning was important as a high learning rate can cause the model to overshoot the minimum and fail to converge. It may lead to instability in training, oscillations, or even divergence, preventing the model from learning effectively. A very low learning rate can slow down the training process significantly. It might take a long time for the model to converge, or it might get stuck in local minima, hindering the model's ability to find the global minimum of the loss function. The alpha and beta interpolation factors were important as they measured the weight given to each component.

I used the dev set edit distance to measure the difference in modifying these parameters. Along with the edit distance on the dev set, I monitored accuracy, loss, grad norm as well as metrics on maximum memory. I used logs such as the below and charts to monitor changes:

```
2023-08-12 15:31:55,118 cascades.metrics.metrics_logger INFO  P2913     [
    train][sweep 38][step 299880]
2023-08-12 15:31:55,119 cascades.metrics.metrics_logger INFO  P2913
        accuracy          -- loss: 0.152,
grad_norm: 0.362
2023-08-12 15:31:55,121 cascades.metrics.metrics_logger INFO  P2913
        info              -- avg_utt_length: 638, padding_rate: 0.032,
    utterances: 9044, samples: 2179808, samples_padded: 2251798,
    max_utt_length: 2729, max_label_length: 96,
    max_ft_time_max_label_length_time_max_utt_num: 999936, sweep: 38,
    min_utt_length: 30, min_label_length: 3
```

```
2023-08-12 15:31:55,122 cascades.metrics.metrics_logger INFO  P2913
     mem              -- max_model_output_size_gb: 2.93e-06,
   max_dl_output_size_gb: 0.0136, pt_peak_allocated_gb: 17.2,
   pt_peak_reserved_gb: 28.7, pt_oom_count: 0, nvml_used_gb: 31.4
2023-08-12 15:31:55,122 cascades.metrics.metrics_logger INFO  P2913
     optim            -- lr: 2.22e-07, lr_layer1: 1
2023-08-12 15:31:55,123 cascades.metrics.metrics_logger INFO  P2913
     timing           -- model_step_time: 6.51e-05, model_forward_time:
   6.29e-05, samples/second: 7.59e+04, data_time: 3.37e-06
```

alpha	beta	learning rate
0.001	1,0	0.0001
0.01	1.0	0.001
0.02	1,0	0.001

Table 2: Example Hyperparameter Tuning Conditions

Table 2 shows the variants I explored. I started from the hyperparameter suggested in the Paper [3]. We found the hyperparameters of alpha = 0.02, beta=1.0 and lr=0.00001 gave us the best accuracy. In our experiments, we used a fixed learning rate. We started learning rates include 0.1, 0.01, or 0.001. It is possible using a learning rate schedule would give increased performance. We found six epochs of training to be when the model converged. We averaged these epochs and found this to be the point where results were the best.

10 Results and Discussion

10.1 High Level Results

Our implementation computes both the normal RNNT loss and the Minimum WER loss. The principal idea is that hypotheses with fewer errors will be boosted. With our IVR tests (customer datasets), we observed on average 4.1% Word Error Rate Reduction, with some datasets reaching 8% Word Error Rate Reduction (WERR) on production models.

The results found matched those cited in literature. The 9 dataset shown in Table 3 represent important domains for Microsoft. They are customer datasets which are used as a baseline. Once improvements are shown in the 9 datasets, I then evaluated the 77 datasets. The 77 datasets are diverse customer outputs in the Human-to-Machine Domain.

Table 3: High Level Result

	Baseline 1 (WER)	minWER 2 (WER)	Average (WERR)
9 datasets	12.39851582	11.931123	4.123
77 datasets	11.88535105	11.22656442	4.6976

On each of the result tables, we can observe the Word Error Rate Reduction. Word Error Rate (WER) reduction refers to the improvement in the accuracy of automatic speech recognition (ASR) or speech-to-text systems. WER is a metric used to evaluate the performance of such systems by measuring the percentage of errors made in transcribing speech.

Dataset	**Baseline**	**minWER**	**WERR**
results	12.39851582	11.931123	4.123
test1	11.38259331	10.4445755	8.24
test2	13.05361305	12.16006216	6.85
test3	13.42986939	12.734	5.18
test4	8.24871768	7.84	4.95
test5	8.95979021	8.555506993	4.51
test6	10.96074641	10.54	3.84
test7	8.627984285	8.431550317	2.28
test8	21.68130489	21.51	0.79
test9	15.2420231	15.16441606	0.51

Table 4: The 9 Fundamental Datasets

WER reduction, therefore, signifies the decrease in errors made by the system when transcribing spoken language into text. This reduction can be achieved through various mean. In our case, the results have improved due to improved algorithms and models. While our baseline model was strong, we can enhancing the underlying algorithms and machine learning model. Our changes have lead to better recognition accuracy. This improvement is due to our sophisticated training techniques. The following tables show these results are fairly consistent through the 77 datsets shown on the following tables.

In the datasets, we postulate that our training is most helpful for:

- Boolean Datasets: Boolean Datasets are those which contain many ”no/yes” operators. These may be deleted in other cases. By applying more attention to these deletions, the model can correct such errors.

- Adaptation to Specific Domains: we fine-tuned the ASR system to specific domains or industries (like healthcare, finance, etc.). This enhanced the accuracy as it becomes more attuned to the specialized vocabulary and language nuances of those domain.

The following tables show a sample of the 77 datasets. With this level of testing, we were satisfied the feature was helpful in refining results.

	baseline wer	**min-wer**	WERR
0	11.89	11.23	**4.70**
1	0.99	0.65	**34.55**
2	36.38	24.07	**33.84**
3	9.96	8.03	**19.35**
4	15.15	12.66	**16.41**
5	14.17	11.89	**16.08**
6	12.17	10.30	**15.42**
7	2.23	1.90	**14.93**
8	15.34	13.31	**13.27**
9	17.14	14.97	**12.67**
10	18.02	16.34	**9.34**
11	22.31	20.25	**9.26**
12	18.01	16.45	**8.63**
13	24.98	22.91	**8.31**
14	11.19	10.30	**7.94**
15	15.46	14.24	**7.86**
16	12.44	11.52	**7.40**
17	20.14	18.65	**7.39**
18	17.16	15.92	**7.23**
19	13.05	12.16	**6.85**
20	10.53	9.82	**6.73**
21	3.72	3.54	**5.02**
22	8.69	8.28	**4.73**

	baseline wer	**min-wer**	WERR
1	8.37	8.04	**3.99**
2	7.86	7.58	**3.59**
3	12.99	12.53	**3.59**
4	17.50	16.90	**3.43**
5	10.72	10.37	**3.22**
6	16.47	15.94	**3.20**
7	11.96	11.59	**3.10**
8	11.05	10.72	**2.96**
9	20.91	20.31	**2.87**
10	7.02	6.83	**2.70**
11	8.10	7.88	**2.69**
12	8.63	8.40	**2.62**
13	12.97	12.64	**2.53**
14	8.27	8.06	**2.50**
15	17.16	16.73	**2.49**
16	8.74	8.53	**2.30**
17	4.17	4.07	**2.25**
18	10.51	10.27	**2.24**
19	10.67	10.44	**2.19**
20	7.84	7.70	**1.85**

11 Critical evaluation of the contribution of the internship to the student's training:

The main outcomes of the internship were skill development and professional experience. This internship offered opportunities to develop and refine technical skills relevant to the field. I was also able to apply research directly to a customer problem.

For me, I learned how to use pytorch, deep learning and machine learning techniques. I gained practical experience in a professional setting. As this was a relatively short time period, I developed my ability for teamwork, problem-solving, and time management. My contribution was integrated into two codebases and can be used by other engineers. This technique is widespread and I am proud that we are able to use it.

12 Conclusion

Reflecting on the work completed in this internship, there are a number of things I would like to improve for my next research project. I expended energy in respecting the complexity and computational intensity of this method. With this deep learning method, decoding is an expensive action. Minimum Word Error Rate training involves complex optimization procedures, requiring significant computational resources and time. I would have like to dedicate more time to hyperparameter tuning. I observed early overfitting. This means the model might perform exceedingly well on the training set but may struggle with generalization when exposed to unseen or real-world data. While our datasets were all human-to-machine, it is unclear whether our model could be adjusted to an unknown domain.

While I used a predefined dataset, in the future, I will take steps to maintain data quality assurance. Minimum Word Error Training scores each word in the training corpus. In this way, it is very sensitive to data artefacts. I will ensure data collected is reliable, relevant, and sufficient for analysis.

I would have also liked to expand the analysis. In the future, I would like to study the contents of the training corpus by using appropriate statistical or analytical techniques and tools for accurate interpretation. The performance of Minimum Word Error trained models heavily relies on the quality, size, and diversity of the training corpus. In cases where the training data cannot adequately represent the variability encountered in real-world scenarios, the effectiveness of the trained model may be limited.

I also did not have enough time to observe other Microsoft Software users of this functionality. More specifically, I would also like to see how users use the Minimum Word Error Rate functionality. It is probable there are hyperparameters I did not tune that other researchers would exploit. I wrote the utility as software user but I would need to consider alternatives. It would be helpful to conduct a user study to explore and justify the chosen methodologies and be open to adapting if more suitable methods are found.

13 Literature Cited

References

[1] Google Research. Large-scale multilingual speech recognition with a streaming end-to-end model. https://blog.research.google/2019/09/large-scale-multilingual-speech.html, September 2019. [Online; accessed November 20, 2023].

[2] Harsh Shrivastava, Ankush Garg, Yuan Cao, Yu Zhang, and Tara Sainath. Echo state speech recognition. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5669–5673. IEEE, 2021.

[3] Jinxi Guo, Gautam Tiwari, Jasha Droppo, Maarten Van Segbroeck, Che-Wei Huang, Andreas Stolcke, and Roland Maas. Efficient minimum word error rate training of rnn-transducer for end-to-end speech recognition, 2020.