

# COSI-136A Project Part 2: An ASR System For Vietnamese

Team members: Nhi Le and Zhihan Li

## I. Context, purpose of training & training procedure:

### Context: why we chose to train with Whisper:

Out of the two given architectures: Word2Vec + CTC and Whisper, our group decided to choose Whisper for several of its advantages:

Whisper	Word2Vec + CTC
<p><b>Multilingual Capabilities:</b> Whisper is designed as a multilingual ASR model, meaning it can recognize speech in various languages. This makes it versatile for applications that involve different languages without the need for language-specific models.</p> <p><i>Whisper was also trained on 691 hours of Vietnamese speech, which we think is a big advantage compared to Word2Vec.</i></p>	<p>Wav2Vec models are typically trained on specific languages, and extending them to support multiple languages might require training separate models for each language. Fine-tuning for multilingual support may not be as straightforward as using a dedicated multilingual model like Whisper.</p>
<p><b>Pre-trained Model:</b> Whisper is pre-trained on a large corpus of multilingual and multitask supervised data collected from the web. This pre-training allows it to capture general features of speech across different languages and tasks, potentially providing a good starting point for fine-tuning on specific tasks or languages.</p>	<p>While Wav2Vec models are pre-trained on large datasets, they may not inherently capture the same level of multilingual and multi-task features as Whisper. Fine-tuning Wav2Vec for specific tasks may require more effort and domain-specific data.</p>
<p><b>Fine-tuning Flexibility:</b> Users can fine-tune the Whisper model on their specific speech recognition tasks, adapting it to particular domains or languages of interest. Fine-tuning allows</p>	<p>Fine-tuning Wav2Vec models might require more effort and careful parameter tuning, especially for tasks involving languages or domains not well-represented in the original</p>

customization and enhancement of the model's performance for specific use cases.	pre-training data. The process may not be as straightforward as fine-tuning a dedicated multilingual model like Whisper.
<b>Ease of Use:</b> Hugging Face provides a user-friendly interface for accessing and using pre-trained Whisper models through the Transformers library. This simplifies the process of integrating ASR capabilities into applications without the need for extensive expertise in deep learning.	Wav2Vec models might have a steeper learning curve for users who are not familiar with the intricacies of training ASR models. Setting up and fine-tuning Wav2Vec models may require more expertise in deep learning and ASR compared to using pre-trained models like Whisper.
<b>Community Support:</b> Hugging Face has a large and active community, which means users can benefit from shared knowledge, resources, and pre-trained models. This community support can be valuable for troubleshooting, learning, and collaborating on ASR-related projects.	While Hugging Face provides resources for Wav2Vec, the community support for models trained in this architecture might not be as extensive or diverse as that for models like Whisper. Users might find fewer pre-trained models and community-contributed resources compared to more widely used architectures.

## Purpose of training:

1. Using Whisper out-of-the-box led to poor performance on Bible data. Since the Bible uses a special language that is not present in everyday Vietnamese, using Whisper without any fine-tuning resulted in high WER. We wanted to see if fine-tuning Whisper on a corpus of uncommon Vietnamese words will help improve the WER. If this is the case, we can potentially expand this to other areas where there is a need to accurately recognize words not commonly used in everyday life such as in medicine, scientific research, academic content such as lectures, etc.
2. We also want to see the advantage of using a high-quality corpus on improving the performance of an ASR system. Since the corpus created from part 1 of our project went through a very meticulous process of data ingestion, data quality evaluation and data issues amendment, we have high confidence that the alignment and match between the audio and the transcription are accurate for our corpus. The transcriptions were also sourced from written texts (the Bible and the book that the audiobook was read from), so there are no errors stemming from human errors as compared with other human-annotated corpus such as Common Voice.

## Training procedure:

### Data Partition

According to the template we used for the Whisper model, we did not manually divide the dataset into train and test sets. Instead, we used the Hugging Face library called 'datasets', loaded the dataset, shuffled it, then split it into the train set and test set. Details below. For these two purposes, we decided to follow the below steps:

1. For Experiment 1, we created the final corpus by combining the full audiobook data and half of the Bible data (chosen at random). This resulted in 3 hours of speech corpus to train on.

```
awk '{SUM += $1} END { printf "%d:%02d:%02d\n", SUM/3600,
SUM%3600/60, SUM%60 }'
```

3:08:19

For Experiment 2, we created the final corpus by using full audiobook data. This resulted in 1.5 hours of speech corpus to train on.

```
oxi -D "$file"; done | awk '{SUM
%60 }'
```

1:34:43

2. Shuffle the data using a random seed(seed=123) to ensure that both the train and test sets have an even distribution of bible and audiobook samples. **This step is crucial to ensure there is no bias between the train and test sets, and that they contain both the bible and audiobook data.**

For Experiment 1, we chose to reserve 80% of data for training (2364 examples) and 20% for testing (592 examples).

```
DatasetDict({
  train: Dataset({
    features: ['input_features', 'labels'],
    num_rows: 2364
  })
  test: Dataset({
    features: ['input_features', 'labels'],
    num_rows: 592
  })
})
```

For Experiment 2, we chose to reserve 80% of data for training (1275 examples) and 20% for testing (319 examples).

```
DatasetDict({
  train: Dataset({
    features: ['input_features', 'labels'],
    num_rows: 1275
  })
  test: Dataset({
    features: ['input_features', 'labels'],
    num_rows: 319
  })
})
```

## Experiment 1:

3. **Baseline 1:** using the Whisper Small model from OpenAI out-of-the-box, we did not perform any training but instead went straight ahead with model evaluation on the test set. Specifically, we use Whisper Small to perform speech transcription on each of the 592 test cases and compare the output transcription (hypothesis) with the true text (reference). We calculated the WER for each of these 592 test cases and the final WER on the entire test set.
4. **Model 1 - fine-tuned model with Bible + audiobook data:** we performed fine-tuning on the Whisper Small model from OpenAI using our corpus specified in steps 1-2 above. Details can be found in the next sections. We also performed a similar model performance evaluation to the baseline model in step 3 where we reported the WER for each of the 592 test cases and the overall test set WER. During this step, we experimented with a few models of different parameters, model choice and corpus split until we arrived at the best configuration. For brevity purposes, we will only report our final model as the other models were purely for experimentation purposes while we figured out how best to design the final experiment.

## Experiment 2:

After evaluating our model 2 and witnessing significant improvement compared to the baseline, upon further analysis, we noticed that most of the improvement comes from model 2's ability to more accurately transcribe Bible text compared to the baseline (i.e., it might provide marginal improvement on audiobook data but massive improvements on Bible data). We suspect that since the Bible is considered a special and uncommon text (i.e., common Vietnamese does not have words that are commonly found in the Bible), fine-tuning model 2 on a corpus of both Bible and audiobook data might have allowed it to transcribe Bible text better than the baseline Whisper model, which was trained on everyday language data. With this view in mind, we decided to carry out an additional experiment to test our hypothesis:

5. **Baseline 2:** exactly the same procedure as baseline 1, except for the fact that the test set now only contains audiobook examples.
6. **Model 2 - fine-tuned model with only audiobook data:** similar to model 2, we fine-tuned a Whisper Small model on our own corpus, this time with only audiobook data, without any Bible data. We want to see if our hypothesis above (i.e., fine-tuning a pre-trained Whisper model on a special corpus for uncommon text will result in better transcription for that uncommon text) is correct. Specifically, we suspect that, because the audiobook is a popular, recently published non-fiction on various societal issues in Vietnam, the language used in the book is considered common and everyday Vietnamese language. Therefore, we believe that using Whisper out-of-the-box will provide just as good a result as fine-tuning it on the audiobook data. If the WER and the CER values of the baseline model and model 3 are somewhat similar, this will confirm our belief that fine-tuning Whisper is only beneficial if we want to create an ASR system for a special text that is under-represented in daily language.

We also trained our models on our local Jupyter Lab notebooks, despite it not allowing us to utilize Google Colab's GPUs, because we could not figure out a method to upload and read in our corpus through Google Colab. Since we started the project early enough, we were still able to test out our hypothesis and ran our experiments with ample time.

## II. The corpus:

From part 1 of our project, we were able to create the following corpus for Vietnamese:

Number of Speakers: 2

Speaker Gender: Male

Noise: sourced from audiobook and the Bible, almost noise-free.

Data: Vietnamese Bible and an Audiobook

Final form: SampleRate = 16000 Hz, Channels = 1, wav

Length of wav files: more than 2 seconds and less than 10 seconds each

Time Duration:

- Total: 7:28:54
  - Audiobook: 1:34:43
  - Bible: 5:54:11

Given the time and compute power intensities of evaluating and fine-tuning Whisper models, we decided to only use half of the Bible data and the entire audiobook data, giving us a total of 3 hours of speech. Moreover, we also decided that since Whisper is already performing really well, with 691 hours of Vietnamese speech in its training corpus, we suspected that focusing on the training data quality and the time spent training is more important than trying to maximize the size of our train set.

### III. Baseline model: pretrained Whisper Small out-of-the-box:

We decided to use Whisper Small for both our baseline and fine-tuned models because it is middle-of-the-line in terms of number of parameters, which will ensure we will not run into the problem of lacking GPUs for training while also making sure we have good enough performance. We also tried Whisper Large models (V1-3) but quickly ran into memory issues.

Size	Parameters	English-only	Multilingual
tiny	39 M	✓	✓
base	74 M	✓	✓
small	244 M	✓	✓
medium	769 M	✓	✓
large	1550 M	x	✓
large-v2	1550 M	x	✓

#### Step 1: Create metadata file:

We followed the guidelines here to load our own corpus: [https://huggingface.co/docs/datasets/v2.8.0/en/audio\\_load#local-files](https://huggingface.co/docs/datasets/v2.8.0/en/audio_load#local-files)

This required creating a metadata.csv file which looks like this:

file_name	transcription
B0403John_segment10.wav	nhưng chúa thánh linh mới sinh ra tâm linh
B0403John_segment11.wav	vậy đừng ngạc nhiên khi nghe ta nói ông phải tái sinh
B0403John_segment12.wav	nghe tiếng gió nhưng ông không biết gió đến từ đâu hay sẽ đi đâu
B0403John_segment13.wav	người được chúa thánh linh sinh thành cũng thế

Step 2: Loading data, splitting into train and test sets (in this baseline model, only test set was used for evaluation purposes)

```
from transformers import WhisperProcessor, WhisperForConditionalGeneration
from datasets import Dataset, load_dataset
import torchaudio
import torch
from jiwer import wer # Install jiwer using: pip install jiwer

# Load the Whisper processor and model
processor = WhisperProcessor.from_pretrained("openai/whisper-small")
model = WhisperForConditionalGeneration.from_pretrained("openai/whisper-small")
model.config.forced_decoder_ids = None

# Replace "/path/to/folder" with the actual path to your data folder
data_folder_path = "./data/all/"

# Load the audiofolder dataset
dataset = load_dataset("audiofolder", data_dir=data_folder_path)

# Shuffle the dataset
shuffled_dataset = dataset.shuffle(seed=123)

# Manually split the dataset into train and test
split_percentage = 0.8
split_index = int(len(shuffled_dataset["train"]) * split_percentage)

train_dataset = Dataset.from_dict(shuffled_dataset["train"][:split_index])
test_dataset = Dataset.from_dict(shuffled_dataset["train"][split_index:])

# Display available splits in the custom DatasetDict
print(train_dataset, test_dataset)
```

Resolving data files: 100%  5952/5952 [00:00<00:00, 22776.75it/s]

```
Dataset({
  features: ['audio', 'transcription'],
  num_rows: 2364
}) Dataset({
  features: ['audio', 'transcription'],
  num_rows: 592
})
```

Loading OpenAI's Whisper Small processor and model

Setting random seed for shuffling data (so both train and test sets contain both bible and audiobook data)

80-20 train-test split

Since data is already shuffled, train and test sets are partitioned at the split index

Step 3: Calculate WER for each example in the test set and report the overall test set WER:

Reference: <https://huggingface.co/spaces/evaluate-metric/wer>

Word error rate (WER) is a common metric of the performance of an automatic speech recognition system. The general difficulty of measuring performance lies in the fact that the recognized word sequence can have a different length from the reference word sequence (supposedly the correct one). Word error rate can then be computed as:

$$\text{WER} = (S + D + I) / N = (S + D + I) / (S + D + C)$$

where

- S is the number of substitutions,
- D is the number of deletions,
- I is the number of insertions,
- C is the number of correct words,
- N is the number of words in the reference ( $N=S+D+C$ ).

This value indicates the average number of errors per reference word. **The lower the value, the better the performance of the ASR system with a WER of 0 being a perfect score.**

A snapshot of what this step produces: the highlighted words are discrepancies between the hypothesis (Whisper's transcription) vs. the reference text (actual transcription)

```
test case: 1
hypothesis: rằng đã trung tín tường thuật mọi điều ông đã chứng kiến
reference: giăng đã trung tín tường thuật mọi điều ông đã chứng kiến
wer: 0.08333333333333333
test case: 2
hypothesis: như một số báo chí đã bình luận trong thời gian qua
reference: như một số báo chí đã bình luận trong thời gian qua
wer: 0.0
test case: 3
hypothesis: làm long tu chọn bị xây xước nghiêm trọng
reference: làm lòng tự trọng bị xây xước nghiêm trọng
wer: 0.4444444444444444
test case: 4
hypothesis: do đó niềm hy vọng của chúng ta đây sức sống vì chú cứu thế đã từ có chút
reference: do đó niềm hy vọng của chúng ta đây sức sống vì chúa cứu thế đã từ cõi chết
wer: 0.15789473684210525
test case: 5
hypothesis: chúng ta càng chịu khổ đau vì chú có thể bao nhiêu ngay càng an ủi giúp chúng ta
reference: chúng ta càng chịu khổ đau vì chúa cứu thế bao nhiêu ngài càng an ủi giúp chúng ta
wer: 0.21052631578947367
```

.....



Step 3: calculate CER for each example in the test set and report and overall test set CER:

Reference: <https://huggingface.co/spaces/evaluate-metric/cer>

Character error rate (CER) is a common metric of the performance of an automatic speech recognition system. CER is similar to Word Error Rate (WER), but operates on character instead of word. Character error rate can be computed as:

$$\text{CER} = (S + D + I) / N = (S + D + I) / (S + D + C)$$

where

- S is the number of substitutions,
- D is the number of deletions,
- I is the number of insertions,
- C is the number of correct characters,
- N is the number of characters in the reference ( $N=S+D+C$ ).

CER's output is not always a number between 0 and 1, in particular when there is a high number of insertions. This value is often associated with the percentage of characters that were incorrectly predicted. **The lower the value, the better the performance of the ASR system with a CER of 0 being a perfect score.**

## IV. Fine-tuned models: pretrained Whisper Small fine-tuned on our corpus:

Experiment 1: Bible + audiobook data:

Step 1: Loading data, splitting into train and test sets (in this baseline model, only test set was used for evaluation purposes):

Similar to step 1 in section III. This is to ensure the train and test sets are exactly the same across all our models and the baseline so we can make fair comparisons and evaluations.

## Step 2: Model training:

The full code for this step can be found in our project submission Box repository. Here, we will only report the parameters of our final and the reasoning behind our parameter choices:

```
from transformers import Seq2SeqTrainingArguments

training_args = Seq2SeqTrainingArguments(
    output_dir="ZHPProject23/whisper-small-vn",
    per_device_train_batch_size=16,
    gradient_accumulation_steps=1, # increase by 2x for every 2x decrease in batch size
    learning_rate=1e-5, # 1e-5
    warmup_steps=8,
    max_steps=148,
    gradient_checkpointing=True,
    # fp16=True,
    # fp16_full_eval=False,
    evaluation_strategy="steps",
    per_device_eval_batch_size=8,
    predict_with_generate=True,
    generation_max_length=225,
    save_steps=2,
    eval_steps=2, # number of steps to log evaluation metrics
    logging_steps=1, # 25
    report_to=["tensorboard"],
    load_best_model_at_end=True,
    metric_for_best_model="wer",
    greater_is_better=False,
    push_to_hub=True, # set to True if want to upload mdl checkpoints to Hugging Face hub
)
```

Below is the explanation of the training arguments we used in Experiment 1. For most parameters, we use the default values because it is the common setting for our project.

- `output_dir`: This specifies the directory where the output files will be saved.
- `per_device_train_batch_size`: The batch size per device during training. Here, it's set to 16. This means each training step will have 16 samples to be trained.
- `gradient_accumulation_steps`: This is used to accumulate gradients for multiple steps. We set it to 1, meaning the gradients are not accumulated and it will be updated after each step.

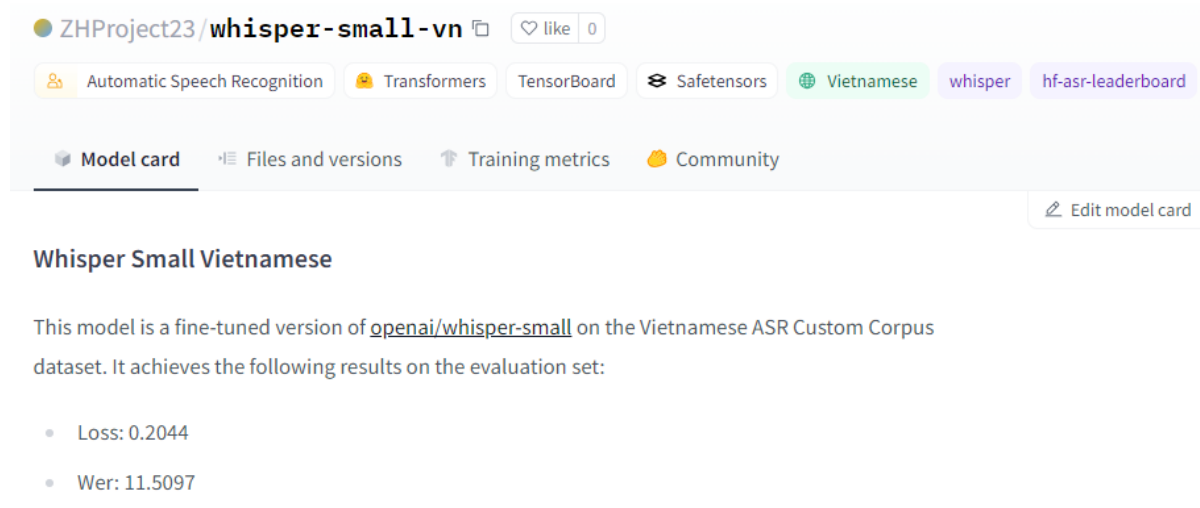
- **learning\_rate:** The learning rate is a parameter used for the machine learning training process. It determines the step size at each iteration while moving toward a minimum of the loss function. We set it to 0.00001 which is a typical value for fine-tune models in deep learning.
- **warmup\_steps:** warmup step is used at the beginning of the training, it will limit learning rate at the beginning of the training. We set it to 8 considering we only have 148 steps in the total training process.
- **max\_steps:** We set max\_steps to 148 for the first fine-tune model. This value is calculated based on the # of training data we have and the batch size we chose.

$$\text{Max Steps Per Epoch} = \frac{\text{Dataset Size}}{\text{Batch Size}} = \frac{2364}{16} = 148$$


- **gradient\_checkpointing:** When set to True, it enables gradient checkpointing for memory efficiency, which is helpful for training large models.
- **evaluation\_strategy:** This parameter defines when to evaluate the model. Setting it to "steps" means the model will be evaluated at regular intervals of training steps.
- **per\_device\_eval\_batch\_size:** This is the batch size for evaluation. Setting it to '8' means 8 samples will be evaluated each time.
- **predict\_with\_generate:** This set to True means it will enable the use of the model's generate method for prediction, which is often used in sequence-to-sequence tasks.
- **generation\_max\_length:** Maximum length of the sequence to be generated. We use the default value.
- **save\_steps:** How often to save a model checkpoint. We set it to 2 means it will save with every 2 steps.
- **eval\_steps:** The number of steps after which to perform an evaluation. We also set it to 2.
- **logging\_steps:** Frequency of logging training information. We set it to 1 to indicate that every step will be logged.
- **report\_to:** Determines where to send logs. It's set to report to "tensorboard". This is the default setting.
- **load\_best\_model\_at\_end:** If True, the model checkpoint with the best evaluation score is loaded at the end of training. This is the default setting. We could use the best model to make predictions
- **metric\_for\_best\_model:** The metric to use for determining the best model. It's set to "wer" (word error rate) since we use wer as the evaluation metric for our project. This is the default setting.
- **greater\_is\_better:** A flag to indicate if a higher value of the metric specified in metric\_for\_best\_model is better. Set to False, as lower WER indicates better performance. This is the default setting.
- **push\_to\_hub:** Set it to True means it will push the model checkpoints to the Hugging Face Hub. This is the default setting.








### Step 3: Saving model and uploading to Hugging Face hub:






This is the model card for our final model on the Hugging Face hub: <https://huggingface.co/ZHProject23/whisper-small-vn>. This is a great functionality offered by Hugging Face, which not only allows us to save our model for easy access and detailed documentation, but also helps us share our model to other people for reproducibility and evaluation purposes.



The screenshot shows the Hugging Face model card for 'ZHProject23/whisper-small-vn'. At the top, there's a header with the model name, a 'like' button showing 0 likes, and a row of tags: 'Automatic Speech Recognition', 'Transformers', 'TensorBoard', 'Safetensors', 'Vietnamese', 'whisper', and 'hf-asr-leaderboard'. Below this is a navigation bar with 'Model card' (selected), 'Files and versions', 'Training metrics', and 'Community'. An 'Edit model card' button is in the top right. The main content area has the title 'Whisper Small Vietnamese' and a description: 'This model is a fine-tuned version of [openai/whisper-small](#) on the Vietnamese ASR Custom Corpus dataset. It achieves the following results on the evaluation set:'. Below the description is a bulleted list of evaluation results: 'Loss: 0.2044' and 'Wer: 11.5097'.

ZHProject23/**whisper-small-vn**  like 0

 Automatic Speech Recognition  Transformers  TensorBoard  Safetensors  Vietnamese  whisper  hf-asr-leaderboard

 **Model card**  Files and versions  Training metrics  Community  Edit model card

### Whisper Small Vietnamese

This model is a fine-tuned version of [openai/whisper-small](#) on the Vietnamese ASR Custom Corpus dataset. It achieves the following results on the evaluation set:

- Loss: 0.2044
- Wer: 11.5097

The following hyperparameters were used during training:

- learning\_rate: 1e-05
- train\_batch\_size: 16
- eval\_batch\_size: 8
- seed: 42
- optimizer: Adam with betas=(0.9,0.999) and epsilon=1e-08
- lr\_scheduler\_type: linear
- lr\_scheduler\_warmup\_steps: 8
- training\_steps: 148

## Experiment 2: audiobook data only:

Our Hugging Face model card for experiment 2: <https://huggingface.co/ZHProject23/whisper-small-audiobook>

Step 1: Loading data, splitting into train and test sets (in this baseline model, only test set was used for evaluation purposes):

Similar to step 1 in section III. This is to ensure the train and test sets are exactly the same across all our models and the baseline so we can make fair comparisons and evaluations.

### Step 2: Model training:

The full code for this step can be found in our project submission Box repository. Here, we will only report the parameters of our final and the reasoning behind our parameter choices.

```
from transformers import Seq2SeqTrainingArguments

training_args = Seq2SeqTrainingArguments(
    output_dir="ZHProject23/whisper-small-audiobook", # change to a repo name of your choice
    per_device_train_batch_size=16,
    gradient_accumulation_steps=1, # increase by 2x for every 2x decrease in batch size
    learning_rate=1e-5, # 1e-5
    warmup_steps=8, # 500
    max_steps=80, # 4000
    gradient_checkpointing=True,
    # fp16=True,
    # fp16_full_eval=False,
    evaluation_strategy="steps",
    per_device_eval_batch_size=8,
    predict_with_generate=True,
    generation_max_length=225,
    save_steps=2, # 1000
    eval_steps=2, # 1000 # number of steps to log evaluation metrics
    logging_steps=1, # 25
    report_to=["tensorboard"],
    load_best_model_at_end=True,
    metric_for_best_model="wer",
    greater_is_better=False,
    push_to_hub=True, # set to True if want to upload mdl checkpoints to Hugging Face hub
)
```

Experiment 2 utilizes the same arguments as Experiment 1, with the only difference being the value of **max\_steps**. In Experiment 2, only audiobook data is used, resulting in a change in data size. Therefore, the max\_steps is adjusted to 80 to allow the model to train for 1 epoch. Please refer to Step 2: Model training for the detailed explanation of each argument.

## V. Results & quantitative analysis: our model vs. baseline:

Experiment 1: baseline 1 vs. model 1 (corpus: Bible + audiobook):

	Baseline 1	Fine-tuned model 1	Fine-tuned WER - Baseline WER
WER on test set	21.56	11.46	10.1 WER improvement
CER on test set	10.94	5.27	5.67 CER improvement

Our model reported better WER and CER metrics than the baseline even with just 3 hours of training corpus. From looking at the individual examples in the test set and seeing how the baseline model's predictions differ from those of our fine-tuned model, we can understand that the big improvement in performance mainly comes from our model's ability to better transcribe the Bible texts (both baseline and fine-tuned models performed relatively well for audiobook texts). Below is one example from many that we saw:

True transcription	Baseline model's transcription	Fine-tuned model's transcription
reference: <b>chúa cứu thế</b> có trước vạn vật	hypothesis: <b>chú cưu thế</b> có trước vàng vật wer: 0.42857142857142855	reference: <b>chúa cứu thế</b> có trước vạn vật wer: 0.0

In this case, “chúa cứu thế” in the true transcription translates to “Jesus” in English. This is apparently not a word that normally appears in everyday language (since Christianity is not a main religion in Vietnam, you would not expect to hear Vietnamese say “Jesus” a lot in their daily lives). The baseline model failed to transcribe this word correctly, while the fine-tuned model was able to.

## Experiment 2: baseline 2 vs. model 2 (corpus: audiobook only):

	Baseline 2	Fine-tuned model 2	Fine-tuned WER - Baseline WER
WER on test set	21.24	17.33	3.91 WER improvement
CER on test set	12.61	8.7	3.91 CER improvement

**In experiment 1, we saw that the fine-tuned model provided a 10.1 WER improvement over the baseline model, whereas this improvement is only 3.91 for experiment 2. Similarly, for CER, experiment 1 saw a 5.67 improvement over the baseline model, where as it is only 3.91 for experiment 2.**

*To reiterate our hypothesis: We suspect that since the Bible is considered a special and uncommon text (i.e., common Vietnamese does not have words that are commonly found in the Bible), fine-tuning model 1 on a corpus of both Bible and audiobook data might have allowed it to transcribe Bible text better than the baseline Whisper model, which was trained on everyday language data. In other words, we hypothesize that the significant improvement in WER seen in experiment 1 is mainly due to model 1's ability to better transcribe Bible text thanks to our corpus. Conversely, because the audiobook is a popular, recently published non-fiction on various societal issues in Vietnam, the language used in the book is considered common and everyday Vietnamese language. Therefore, we believe that using Whisper out-of-the-box will provide just as good a result as fine-tuning it on the audiobook data*

Hence, the results seen in experiment 1 and 2 have confirmed our hypothesis that **the true value of fine-tuning a Whisper model (which is already very powerful) lies in the context of circumstances where the vocabulary spoken is not commonly used in everyday conversations, such as religion in this case.**

## VI. Limitations & conclusion:

### Limitations:

We acknowledge that our project still has room for improvements and further experimentation. Specifically, if given more compute power, we would like to train on a larger corpus with more diversity and variations (speaker gender, speaker dialect, topic of speech, etc.) to make the fine-tuned Whisper model even more accurate to everyday Vietnamese language. Currently, it only performs well for speech that is spoken clearly, using standard Vietnamese dialect and with minimal background noise. We also would like to experiment with running the Whisper Large V3 model, which is the best version of the Whisper model to date.

### Conclusions:

Through this project, we could fully appreciate the power of Whisper on ASR tasks for a lower-resourced language like Vietnamese. Through our 2 experiments, we found out that Whisper can be improved even more for more special and uncommon texts by fine-tuning with our own corpus. This makes sense as even though Whisper was trained on 691 hours of Vietnamese speech, most of this is everyday language, which might not contain words found in other more niche topics such as religion (as we saw with the Bible).

We believe that Whisper can be easily adapted to various use cases in different areas such as medicine, scientific content or research, to name just a few. Moreover, we think that significant improvements can be achieved with just a few hours of speech as long as the corpus is accurate and high-quality enough.

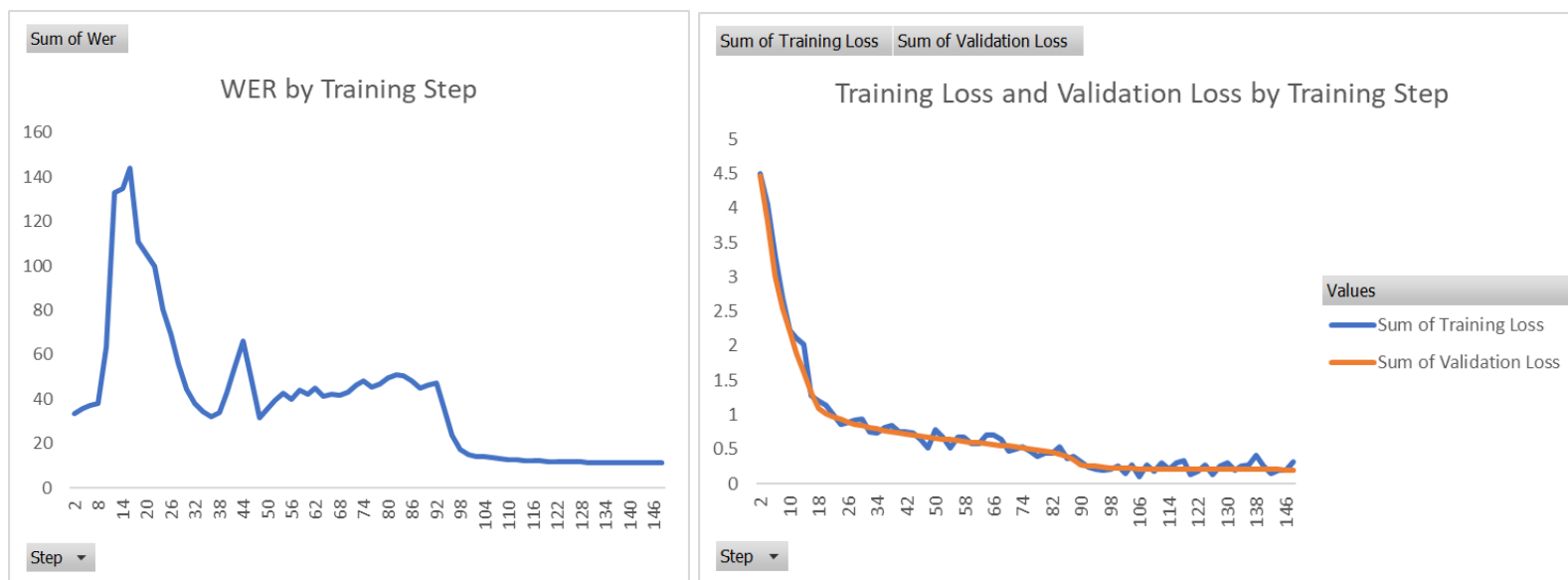
## VII. Reference:

1. <https://huggingface.co/blog/fine-tune-whisper>
2. <https://medium.com/@bofenghuang7/what-i-learned-from-whisper-fine-tuning-event-2a68dab1862>
3. [https://huggingface.co/docs/datasets/v2.8.0/en/audio\\_load#local-files](https://huggingface.co/docs/datasets/v2.8.0/en/audio_load#local-files)
4. <https://github.com/huggingface/community-events/tree/main/whisper-fine-tuning-event>
5. <https://huggingface.co/spaces/openai/whisper/discussions/6#63c142a294b28327f0e6bebd>
6. [https://colab.research.google.com/github/sanchit-gandhi/notebooks/blob/main/fine\\_tune\\_whisper.ipynb](https://colab.research.google.com/github/sanchit-gandhi/notebooks/blob/main/fine_tune_whisper.ipynb)



## VIII. Appendix:

Training logs of fine-tuned model in experiment 1:



Training Loss	Epoch	Step	Validation Loss	Wer
4.5043	0.01	2	4.4639	33.6957
4.0539	0.03	4	3.7975	35.9783
3.3205	0.04	6	3.0084	37.2101
2.7077	0.05	8	2.5555	37.971
2.2203	0.07	10	2.2051	63.587
2.1151	0.08	12	1.9006	132.9227

2.0148	0.09	14	1.6122	134.7222
1.2862	0.11	16	1.3244	144.0821
1.207	0.12	18	1.0984	110.8575
1.146	0.14	20	1.0196	105.2053
1.0035	0.15	22	0.9752	99.8792
0.8611	0.16	24	0.9322	80.3986
0.8995	0.18	26	0.8938	69.07
0.9165	0.19	28	0.8634	55.9058
0.9399	0.2	30	0.8389	44.5894
0.7586	0.22	32	0.8171	38.1884
0.7416	0.23	34	0.7963	34.4444
0.8141	0.24	36	0.7765	31.9082
0.8389	0.26	38	0.7578	34.1546
0.753	0.27	40	0.7393	42.6449
0.756	0.28	42	0.7228	54.4203
0.7347	0.3	44	0.7077	66.2681
0.6526	0.31	46	0.6939	49.2874
0.5208	0.32	48	0.6803	31.5821
0.7804	0.34	50	0.6657	35.3261

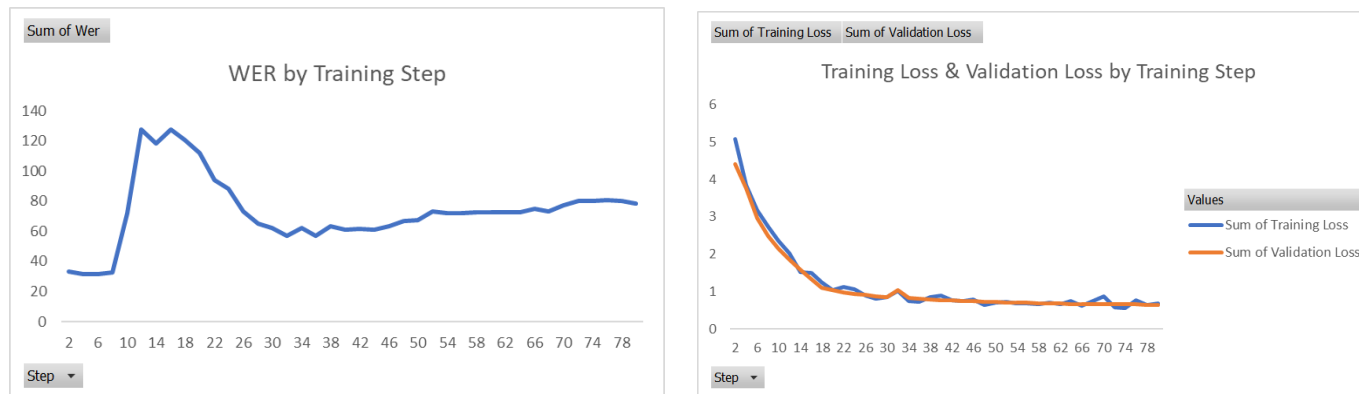
0.6831	0.35	52	0.6524	39.2995
0.5237	0.36	54	0.6409	42.6087
0.6799	0.38	56	0.6305	40.0966
0.681	0.39	58	0.6195	44.2754
0.5804	0.41	60	0.6062	42.2101
0.5869	0.42	62	0.593	44.9517
0.7131	0.43	64	0.5805	41.4734
0.7012	0.45	66	0.5689	42.1498
0.6513	0.46	68	0.5576	41.9203
0.4762	0.47	70	0.546	43.0193
0.503	0.49	72	0.5333	46.2198
0.5342	0.5	74	0.52	48.1522
0.4769	0.51	76	0.5055	45.3382
0.3922	0.53	78	0.491	46.6667
0.4501	0.54	80	0.4752	49.5169
0.4469	0.55	82	0.4574	50.8092
0.5398	0.57	84	0.4356	50.5797
0.3611	0.58	86	0.4053	48.2246
0.4015	0.59	88	0.3565	44.9275

0.3192	0.61	90	0.2724	46.4493
0.2466	0.62	92	0.2516	47.1256
0.2147	0.64	94	0.2574	35.0483
0.1898	0.65	96	0.2431	23.8527
0.2059	0.66	98	0.2314	17.1981
0.2634	0.68	100	0.2258	15.1208
0.1498	0.69	102	0.2223	14.3599
0.2672	0.7	104	0.2202	13.913
0.0989	0.72	106	0.2192	13.5024
0.2685	0.73	108	0.2181	13.128
0.1886	0.74	110	0.2168	12.8744
0.3012	0.76	112	0.2155	12.7174
0.2134	0.77	114	0.2143	12.3792
0.3099	0.78	116	0.2133	12.3188
0.336	0.8	118	0.2124	12.2826
0.1386	0.81	120	0.2112	12.0652
0.1756	0.82	122	0.21	11.9807
0.2789	0.84	124	0.2092	11.8237
0.1284	0.85	126	0.2085	11.7512

0.2586	0.86	128	0.2078	11.6304
0.31	0.88	130	0.2072	11.5942
0.1971	0.89	132	0.2067	11.57
0.2664	0.91	134	0.2062	11.5459
0.2684	0.92	136	0.2058	11.5459
0.4082	0.93	138	0.2053	11.4976
0.2593	0.95	140	0.205	11.4855
0.143	0.96	142	0.2048	11.4976
0.2015	0.97	144	0.2046	11.5097
0.1958	0.99	146	0.2045	11.5217
0.3197	1	148	0.2044	11.5097

## Training logs for fine-tuned model in experiment 2:

<https://huggingface.co/ZHProject23/whisper-small-audiobook>



Training Loss	Epoch	Step	Validation Loss	Wer
5.0759	0.03	2	4.3984	33.2959
3.8512	0.05	4	3.7516	31.7467
3.1744	0.07	6	2.9537	31.2528
2.7374	0.1	8	2.4814	32.4427
2.3471	0.12	10	2.1517	72.1374
2.018	0.15	12	1.8614	127.7279
1.5275	0.17	14	1.5862	118.3431
1.4899	0.2	16	1.3252	127.5932

1.2433	0.23	18	1.1108	120.5433
1.0353	0.25	20	1.0291	112.0790
1.1132	0.28	22	0.9825	93.9156
1.0524	0.3	24	0.9412	87.9883
0.8907	0.33	26	0.9065	72.9232
0.8172	0.35	28	0.8786	64.9753
0.8563	0.38	30	0.8584	62.3934
1.0131	0.4	32	1.0352	56.7131
0.752	0.42	34	0.8189	62.1015
0.7312	0.45	36	0.8031	57.0723
0.8391	0.47	38	0.7888	63.4710
0.8875	0.5	40	0.7756	60.8217
0.7641	0.53	42	0.7633	61.7647
0.737	0.55	44	0.7519	61.2259
0.7782	0.57	46	0.7417	63.5384
0.6495	0.6	48	0.7318	67.0409

0.7102	0.62	50	0.7219	67.2205
0.7225	0.65	52	0.7132	72.9232
0.6752	0.68	54	0.7053	71.7782
0.679	0.7	56	0.6978	72.0476
0.6642	0.72	58	0.6906	72.8334
0.7048	0.75	60	0.6836	72.5191
0.6554	0.78	62	0.6773	72.4742
0.7454	0.8	64	0.6714	72.5415
0.6286	0.82	66	0.6663	74.7643
0.7423	0.85	68	0.6620	72.9681
0.8805	0.88	70	0.6584	77.0094
0.5701	0.9	72	0.6554	80.0404
0.5509	0.93	74	0.6531	80.0180
0.7618	0.95	76	0.6514	80.3996
0.6455	0.97	78	0.6503	79.8383
0.6748	1.0	80	0.6499	78.4463



