Performance from Base Model (Task 2a)

Min WER: 1.0

Max WER: 2.25

Average WER: 1.01

Performance from Trained Model (Task 3)

Apologies, did not have time to finish training it

Min WER:

Max WER:

Average WER:

1. Preprocessing Improvements

Better preprocessing can reduce noise and improve model generalization:

Suggested Enhancements:

- **Silence trimming** using `torchaudio.functional.vad()` or `librosa.effects.trim()` to remove leading/trailing silence.
- **Loudness normalization** so that audio has consistent volume levels across samples.
- **Filter out short or blank transcripts** (e.g., less than 3 characters).
- **Resample audio offline** instead of at runtime to reduce noise introduced during dynamic resampling.
- **Apply audio augmentations** (only during training):
    - Random time-stretch, pitch shift

- Add environmental background noise
- Use tools like `audiomentations`, `sox`, or `torchaudio`

---

2. Tokenization and Label Cleaning

Remove punctuation and convert text to lowercase to reduce vocabulary size:

python
CopyEdit

```
import re
df["text"] = df["text"].str.lower().apply(lambda x: re.sub(r"[^a-z'
]", "", x))
```

-
- Strip whitespace and normalize contractions.

This helps reduce the edit distance between predicted and target text, lowering WER.

---

3. Training Hyperparameter Tuning

You're currently using:

python
CopyEdit

```
num_train_epochs=3,
learning_rate=2e-5,
per_device_train_batch_size=8,
```

✅ **Recommended Tuning:**

Parameter	Recommendation
`num_train_epochs`	Increase to 10–15 with early stopping
`learning_rate`	Try 1e-5 or use a learning rate scheduler
`lr_scheduler_type`	`"linear"` or `"cosine"` with `warmup_steps=500`

`fp16`	Enable for faster training and better memory efficiency
`gradient_accumulation_steps`	Use if increasing batch size is not possible

Example:

python
CopyEdit

```
training_args = TrainingArguments(
    output_dir="./results",
    per_device_train_batch_size=8,
    gradient_accumulation_steps=2,
    num_train_epochs=10,
    learning_rate=1e-5,
    warmup_steps=500,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    fp16=True,
    logging_dir="./logs",
    logging_steps=10,
    report_to="none"
)
```

---

4. Model Checkpoints and Early Stopping

Use `EarlyStoppingCallback`:

python
CopyEdit

```
from transformers import EarlyStoppingCallback

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    tokenizer=processor,
```

```
    data_collator=data_collator,
    callbacks=[EarlyStoppingCallback(early_stopping_patience=2)]
)
```

---

5. Data Volume and Quality

If you're only using `cv-valid-train.csv`, you can:

- Add more data from `cv-other-train.csv` or `cv-valid-dev.csv`
- Filter these for `locale == 'en'` and merge with your training set

More high-quality data always helps fine-tuning.

---

6. Postprocessing / Decoding Improvements

You can improve transcription accuracy by using a **language model during decoding** (KenLM or similar) using `CTCDecoder` or `flashlight`.

Libraries to consider:

- `pyctcdecode`
- `ctcdecode`
- `kenlm`

Summary of Key Improvements

Area	Strategy
Preprocessing	Silence trimming, normalization, augmentation
Text Cleaning	Lowercasing, punctuation removal, contraction normalization

Hyperparameters	More epochs, scheduler, warmup, gradient accumulation, fp16
Evaluation	Add early stopping and WER evaluation
Data	Add more Common Voice splits, filter for English
Decoding	Use beam search + language model for better final transcription outputs