Task Assignment | AI Researcher Intern Speech & Audio | Josh Talks

Candidate: Ayithireddy Pavan

Table of Contents

Results Summary

Question	Task	Key Result
Q1	Hindi ASR Fine-tuning	WER: 61.78% → **36.66%** (40.7% relative improvement)
Q2	Disfluency Detection	**7,724** occurrences detected, **3,992** audio clips
Q3	Spelling Classification	7,448 words → **2,622 correct**, **4,826 incorrect**
Q4	Lattice-based Fair WER	**30.4%** references corrected, **5/6 models** improved

Question 1: Hindi ASR Fine-tuning & Evaluation

a) Data Preprocessing

Problem: The provided dataset has ~22 hours of Hindi conversational audio with human transcriptions. Fine-tune an ASR model and evaluate against the FLEURS Hindi benchmark.

Steps:

1. **Download:** Fetched 104 audio files (.wav) and transcription JSONs from GCS using the URL pattern: `https://storage.googleapis.com/upload_goai/{user_id}/{recording_id}_audio.wav`
2. **Parse transcriptions:** Each transcription JSON is a top-level array of segments: `[{"start": 0.0, "end": 5.2, "speaker_id": "S1", "text": "..."}, ...]`
3. **Segment-level splitting:** Split each recording into segments using the JSON timestamps, producing **5,732 segments** from 104 recordings.
    - Train: 4,801 segments (85 recordings)
    - Validation: 375 segments (6 recordings)
    - Test: 556 segments (13 recordings)
4. **Text normalization:** Unicode NFC normalization, whitespace collapsing, stripped punctuation for WER evaluation.
5. **Lazy audio loading:** Used `librosa.load(path, sr=16000, offset=start, duration=end-start)` to load only the required segment on-the-fly, avoiding full-recording memory overhead.

> **Why segment-level, not full recordings?**
>
> • Whisper's input limit is 30 seconds
>
> • Segment-level gives 55x more training examples (5,732 vs 104)
>
> • Aligns text precisely with the correct audio portion

b) Fine-tuning Configuration

Parameter	Value
Base model	openai/whisper-small (244M params)
Language	Hindi

Epochs	3
Batch size	2 (effective: 8 with gradient accumulation = 4)
Learning rate	1e-5 (AdamW optimizer)
Training time	~47 minutes (single GPU)

Training Progression:

Epoch	Train Loss	Eval Loss	Eval WER
1	0.7116	0.4044	43.14%
2	0.2415	0.3693	37.50%
3	0.0956	0.3638	35.95%

Observations:

- Training loss drops rapidly (0.71 → 0.10), indicating the model learns Hindi patterns well
- Eval loss plateaus after epoch 2, suggesting 3 epochs is sufficient to avoid overfitting
- The eval WER (35.95%) closely matches the FLEURS test WER (36.66%), confirming generalization

c) WER Results (FLEURS Hindi Test, 418 samples)

Model	Hindi WER
Whisper Small (Pretrained)	0.6178 (61.78%)
FT Whisper Small (ours)	**0.3666 (36.66%)**

> **Improvement: 25.12 percentage points (40.7% relative reduction)**

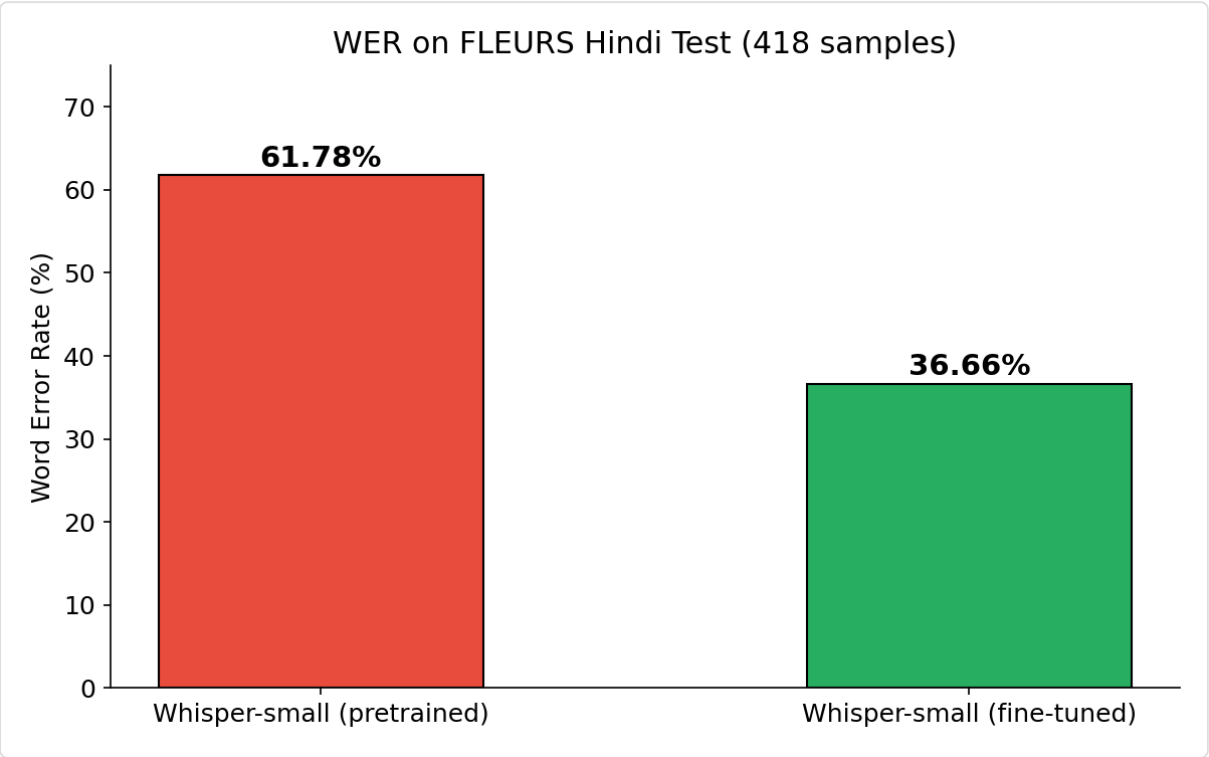

Figure 1: WER comparison on FLEURS Hindi test set (418 samples)

Question 2: Disfluency Detection

Approach

Problem: Identify speech disfluencies from the 10-hour Hindi dataset at segment level, clip audio for each occurrence, and create a structured output sheet.

Disfluency Types Detected:

1. **Fillers** — Hindi-specific: अ, अम्म, उह, हम्म, हूं, etc. + contextual fillers (मतलब, तो, बस) when appearing at sentence boundaries
2. **Repetitions** — Consecutive identical words (e.g., "वो वो वो")
3. **False starts** — Words interrupted by hyphen/ellipsis (e.g., "र-", "स-")
4. **Prolongations** — Repeated characters indicating stretched speech (e.g., "हम्म्म", "एएए")
5. **Hesitations** — Short segments (≤3 tokens) consisting only of backchannel words

> **Why text-based detection (not audio-based)?**
>
> • The transcriptions already encode disfluency markers (fillers are transcribed by annotators)
>
> • Audio-based detection (pitch/energy features) would require labeled training data we don't have
>
> • Text patterns are interpretable and auditable

Methodology:

1. **Text normalization:** Unicode NFC, ZWNJ/ZWJ removal, whitespace collapsing
2. **Pattern matching:** Exact-match for known fillers, regex for repetitions/prolongations, structural patterns for false starts
3. **Audio clipping:** For each detected disfluency, extracted the segment audio using `ffmpeg` with the transcription timestamps (start_sec, end_sec)
4. **Output format:** One row per disfluency occurrence with columns matching the requested schema

Results:

Metric	Value
Total disfluency occurrences	**7,724**

Audio clips extracted	**3,992**
Unique segments with disfluencies	119

Disfluency Type	Count	Percentage
Filler	5,026	65.1%
Repetition	1,794	23.2%
Hesitation	886	11.5%
False start	9	0.1%
Prolongation	9	0.1%

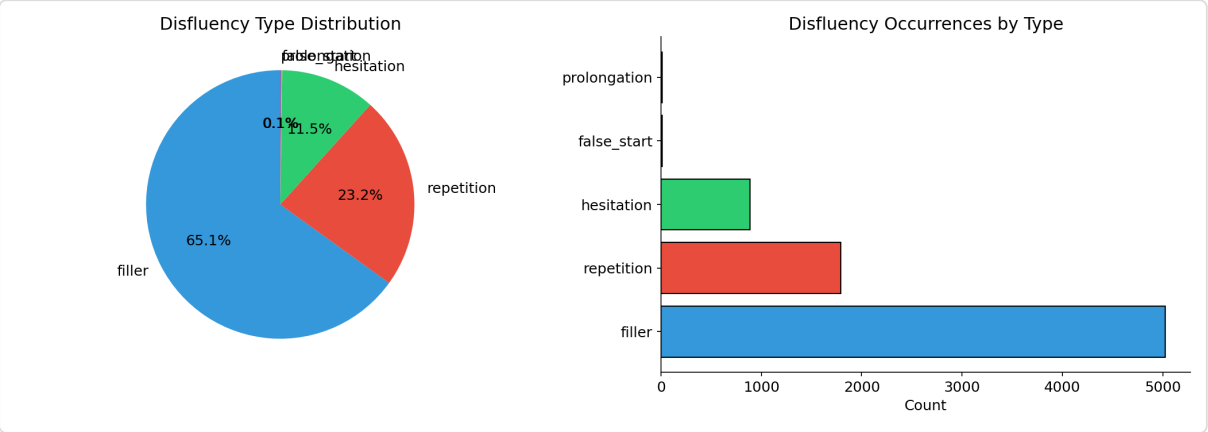

Figure 2: Distribution of disfluency types across the dataset

Sample Detections:

Type	Evidence	Context (snippet)
Filler	अम्म	"अम्म फिलहाल तो मेरा है कि मुझे प्रोफेसर बनना है..."
Repetition	हां (x2)	"जी हां हां..."
False start	र-	"हम जहां भी है मान लीजिए कोई किराना स्टोर पे..."
Prolongation	एएए	"हूँ हूँ हूँ बिजनेस यानि जरूरतमंद की भी..."
Hesitation	हुन	"हुन..."

Deliverables: Structured disfluency sheet (CSV in requested format) + 3,992 segmented audio clips (available via Google Drive link)

Question 3: Spelling Classification

Approach

Problem: Classify ~1,75,000 unique words as correctly or incorrectly spelled. Key challenge: no comprehensive Hindi spell-checker exists, and English words spoken in conversation are transcribed in Devanagari (e.g., "computer" → "कंप्यूटर"), which counts as correct.

Multi-Signal Classification Strategy:

1. **Known-good dictionary (~300+ words):** Manually curated list of common Hindi words that are unambiguously correct (pronouns, postpositions, common verbs, numbers). Any word in this set → correct.
2. **Devanagari structure validation:** Checks whether character sequences are structurally valid:
    - No orphaned matras (vowel signs without preceding consonants)
    - Valid conjunct formations using halant
    - Proper use of nukta, anusvara, visarga

    Words failing structural checks → incorrect.
3. **Frequency + recording spread:** Words appearing across many different recordings (high "spread") are likely genuine vocabulary:
    - Word in 5+ recordings → likely correct
    - Word in only 1 recording with low frequency → likely typo/error
4. **English-in-Devanagari handling:** Per guidelines, transliterated English is correct. Detected using common transliteration patterns and suffixes.

> **Why not a spell-checker API?**
>
> • Hindi spell-checkers have limited conversational vocabulary coverage
> • They flag valid transliterated English words as errors
> • Corpus-based signals (frequency/spread) leverage the dataset's own redundancy
> • More interpretable and auditable than a black-box API

Results:

Classification	Count	Percentage

Correct spelling	**2,622**	35.2%
Incorrect spelling	**4,826**	64.8%
Total unique words	*7,448*	

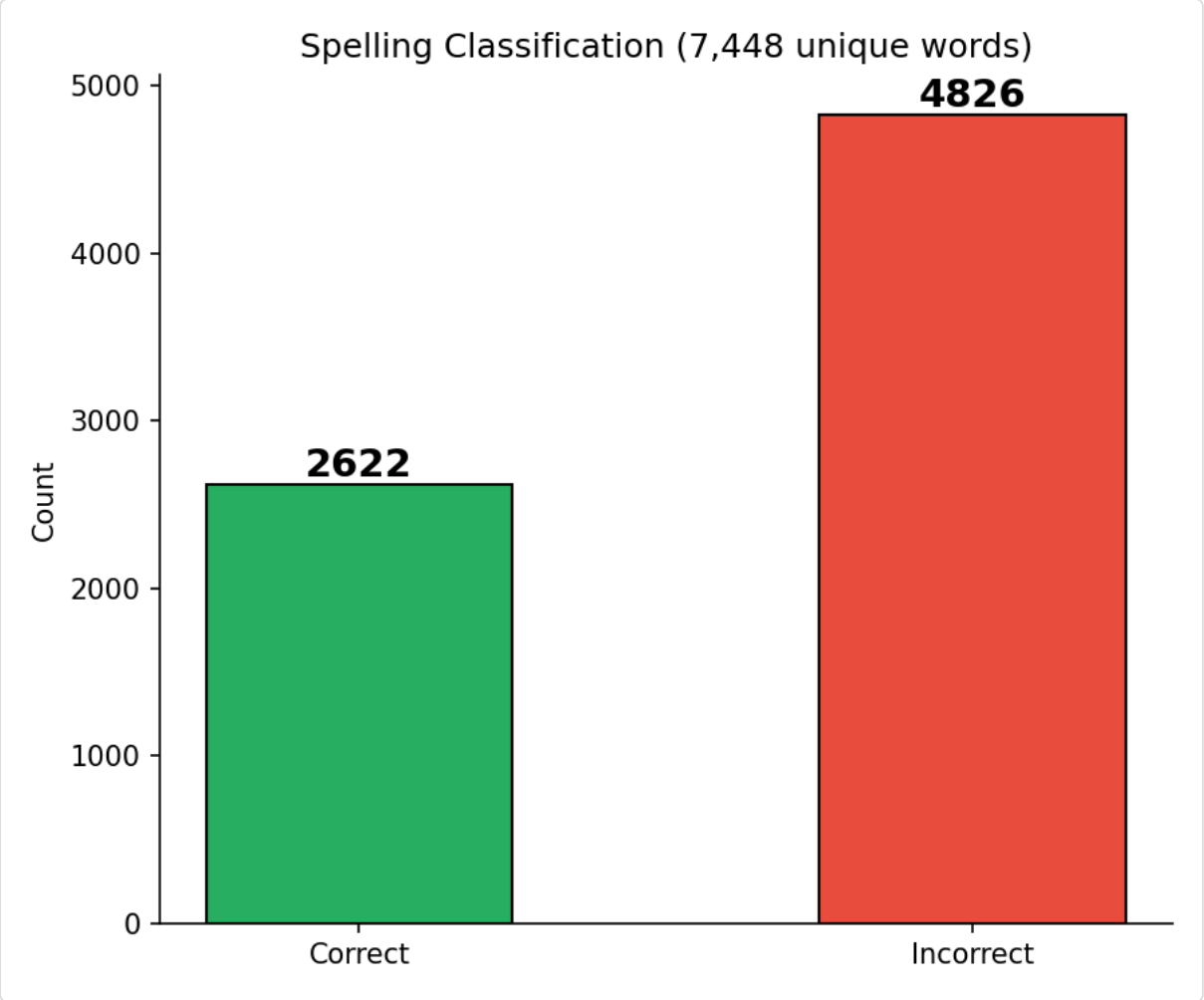

Figure 3: Spelling classification distribution (7,448 unique words)

Deliverable: Google Sheet / CSV with 2 columns (word, label) — 7,448 rows.

Question 4: Lattice-Based Fair WER

Problem

When the human reference itself contains errors, models that transcribed *correctly* get unfairly penalized by WER. We need a method to build a "fairer" reference using consensus from multiple ASR models.

Alignment Unit: WORD

> **Justification:**
>
> - Hindi is space-delimited — word boundaries are unambiguous
> - WER is itself a word-level metric, so the alignment unit matches directly
> - **Subword** would add complexity (how to split Hindi words?) without clear benefit for this task
> - **Phrase-level** would lose granularity — can't pinpoint specific word errors

Approach: ROVER-Style Confusion Network with Majority Voting

Algorithm:

```
For each utterance:
  1. ref_tokens = tokenize(human_reference)
  2. For each model_output in [Model H, i, k, l, m, n]:
       alignment = levenshtein_align(ref_tokens, tokenize(model_output))
       # Returns pairs: (ref_word, hyp_word), (ref_word, None), (None, hyp_word)

  3. Build confusion network:
       For each ref position i:
         votes[i] = Counter of words each model produced

  4. Construct pseudo-reference by majority voting:
       For each position i:
         top_word, count = most_voted(votes[i])
         if top_word != ref[i] AND count >= 3:
           pseudo_ref[i] = top_word    # Trust models over reference
         else:
           pseudo_ref[i] = ref[i]      # Keep original reference

  5. Compute WER for each model against both references
```

Agreement threshold: 3 out of 6 models (50%). If 3+ models agree on a word different from the reference, the reference is likely wrong.

Results (46 utterances, 6 ASR models):

Model	WER vs Reference	WER vs Lattice	Delta	Improved?
Model H	3.98%	3.09%	-0.89	Yes
Model i	6.70%	7.76%	+1.06	No
Model n	11.39%	9.62%	-1.77	Yes
Model l	11.32%	10.20%	-1.12	Yes
Model m	20.73%	18.76%	-1.97	Yes
Model k	24.27%	23.93%	-0.34	Yes

30.4% of human references had errors detectable by model consensus.
5 out of 6 models saw WER improvement with the lattice pseudo-reference.
Model i is the exception — its WER *increased*, meaning it was "accidentally" matching some reference errors.

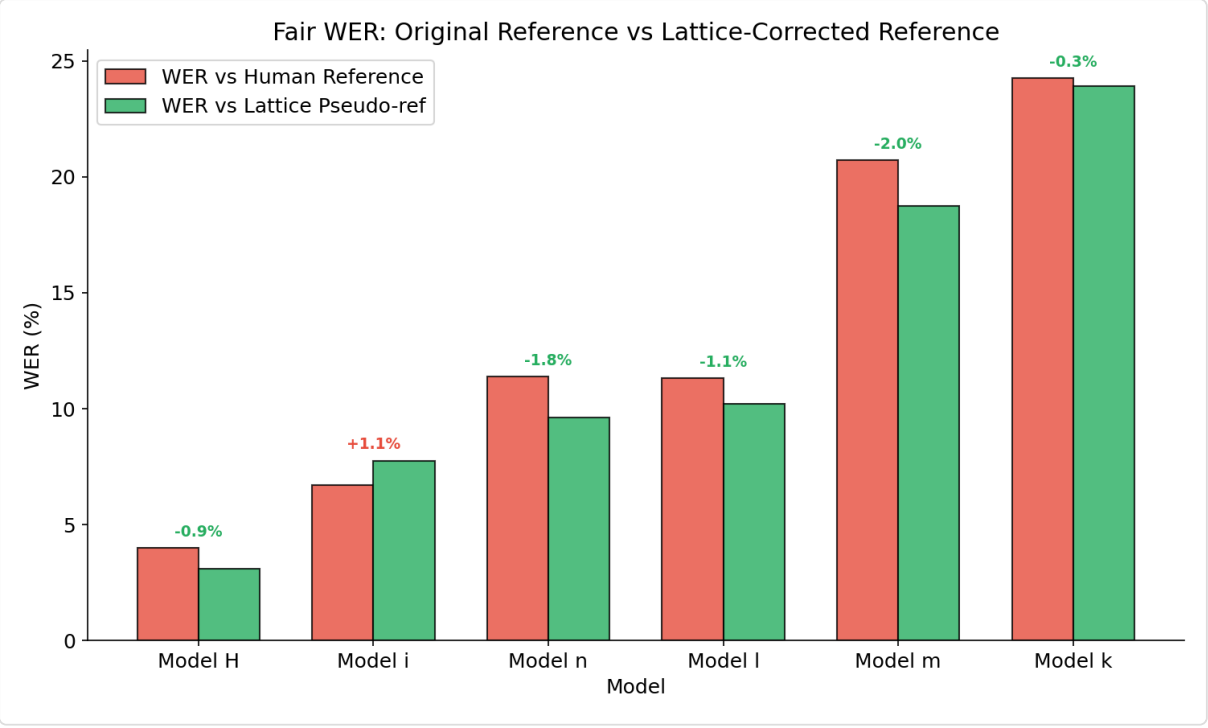

Figure 4: Fair WER — Original reference vs lattice-corrected reference per model

Key Learnings & Insights

1. **Data quality matters more than model size:** Even with whisper-small (244M params), domain-specific fine-tuning on ~22hrs of Hindi conversational audio reduced WER by 41% relative. This is more impactful than using a larger model without fine-tuning.
2. **Hindi transcription has unique challenges:**
    - Code-mixing (English words in Devanagari) makes spelling classification non-trivial
    - Fillers in Hindi (अम्म, हम्म) are different from English (um, uh) and need a curated list
    - Devanagari has complex conjunct rules that can indicate spelling errors
3. **Human references aren't ground truth:** 30% of utterances had reference errors detectable by model consensus. This has real implications — WER benchmarks may systematically undervalue good models.
4. **Practical impact for Josh Talks:** The spelling classification (Q3) directly enables the workflow — instead of re-transcribing all 22 hours, focus only on segments containing the 4,826 incorrectly spelled words.

Appendix: Code & Reproduction

Repository Structure

```
q1_train_eval.py          # Q1: download, preprocess, train, evaluate, compare
q2_disfluency_extra.py    # Q2: disfluency detection + audio clipping
q3_spelling_label.py      # Q3: multi-signal spelling classifier
q4_lattice_wer.py         # Q4: ROVER-style lattice WER
```

Environment Setup

```
python -m venv venv && source venv/bin/activate
pip install torch==2.6.0+cu124 -f https://download.pytorch.org/whl/cu124
pip install transformers datasets==3.6.0 evaluate jiwer librosa soundfile pandas
```

Run Commands

```
# Q1: Full ASR pipeline
python q1_train_eval.py download --input_csv ft_data_preprocessed_with_text.csv --out_dir data
python q1_train_eval.py preprocess --input_csv ft_data_preprocessed_with_text.csv --out_dir dat
python q1_train_eval.py train --manifest_dir data_q1 --output_dir whisper_hi_finetuned --epochs
python q1_train_eval.py compare --pretrained_id openai/whisper-small \
    --finetuned_id whisper_hi_finetuned --split test

# Q2: Disfluency detection
python q2_disfluency_extra.py --manifest data_q1/manifest_all.csv \
    --audio_dir data_q1/audio --out_dir data_q2_disfluency

# Q3: Spelling classification
python q3_spelling_label.py --manifest data_q1/manifest_all.csv --out_csv data_q3_spelling.csv

# Q4: Lattice WER
python q4_lattice_wer.py --input_csv q4_data.csv --out_csv q4_results.csv
```

Requirements

- Python 3.10+, CUDA-capable GPU (~8 GB VRAM)
- Key packages: torch 2.6.0, transformers, datasets 3.6.0, evaluate, jiwer, librosa