

Telugu speech pipeline: evaluation results

Banuru Mohammed Fawaz | Speech AI Research Internship | Vaani Research Lab |
24 October 2026

This report covers Task 2: a reproducible Indic (Telugu) speech pipeline (Speech $\rightarrow$ ASR $\rightarrow$ LLM $\rightarrow$ Text Response), its own evaluation infrastructure, a real evaluation run on 129 examples across two acoustic conditions, and a research-extension bottleneck fix with an ablation. The code, tests, and exact commands to reproduce every number here are in this repository; see `README.md`. This document reports what was actually run and measured, not a projected or idealized result.

1 System

`config.yaml` $\rightarrow$ `asr.py` $\rightarrow$ `llm.py` $\rightarrow$ `pipeline.py`. Pipeline: *Speech input* $\rightarrow$ *Whisper-family ASR* (*vasista22/whisper-telugu-small*) $\rightarrow$ *Qwen2.5-1.5B-Instruct* $\rightarrow$ *Telugu text response*. Both models stay resident on one GPU and are reused across the whole evaluation run; `evaluate.py` measures wall-clock ASR time, LLM time, and total pipeline time per sample, synchronized around CUDA kernels, excluding model loading.

1.1 Model choice and a VRAM constraint

The evaluation GPU (RTX 4060 Laptop, 8 GB) had **~ 6.5 GB free VRAM**. The originally configured defaults — `openai/whisper-large-v3` (~ 3 GB in fp16) and `Qwen2.5-7B-Instruct` (~ 15 GB in fp16) — do not fit together on this hardware. `config.yaml` instead uses:

- **ASR:** `vasista22/whisper-telugu-small`, a Telugu-only fine-tune of Whisper-small (not the general-purpose `openai/whisper-small`). This substitution was not optional: the general `whisper-small` checkpoint, even with Telugu forced via the `language` argument, transcribed this corpus’s Telugu audio into **Devanagari/Hindi-looking script** (e.g. the Telugu reference “Telāṅgāṇa rājadhāni ēdi?” (“What is Telangana’s capital?”, written in Telugu script in the dataset) came back as Devanagari text that merely sounds similar) and hallucinated repeated tokens on several short clips — a real capability gap in the smallest general multilingual Whisper checkpoint on a low-resource language, not a configuration bug. The first full clean-condition run under that checkpoint is still visible in this repository’s history; its corpus WER was 2.56 (word-level garbage from the script mismatch), which is why it was discarded and re-run.
- **LLM:** `Qwen2.5-1.5B-Instruct`, the smallest instruction-tuned member of the Qwen2.5 family.

A second, unrelated compatibility issue surfaced while switching checkpoints: `vasista22/whisper-telugu-small` ships a `generation_config.json` that predates `transformers`’ newer `language/task` validation path, and raises `ValueError` if either is passed to `generate()` at all (not only when set to an unsupported value) — see [huggingface/transformers#25084](https://huggingface.co/transformers#25084). This reproduced identically on `transformers` 5.15.0 and 4.57.6, so it is a property of this specific checkpoint, not the `transformers` major version. Since the checkpoint is Telugu-only, forcing a language is unnecessary, so `asr.py` was changed to treat `asr.language: null` as “do not pass `language/task` to `generate()` at all” (instead of always forcing a value), and `config.yaml` sets `language: null` for this checkpoint. `requirements.txt` was relaxed from `transformers<5` to `transformers<6` (the environment this ran in had 5.15.0 installed); `verify_task2.py` and its tests were updated for both changes.

Reproducibility caveat on this specific checkpoint: while pinning exact commit hashes for this report, `vasista22/whisper-telugu-small`’s `config/tokenizer` files and its `model.safetensors` resolved to **two different main** commits within the same evaluation session (717212f9... for `config/tokenizer`, 9bde7686... for weights) — most likely an upstream micro-commit to the repository during this session. Pinning either single hash broke loading (the other component’s files are absent from that snapshot), so `config.yaml` leaves this one checkpoint’s revision at `main` rather than a

broken pin; Qwen2.5-1.5B-Instruct’s revision (989aa7980e...) is pinned normally and was stable throughout.

1.2 Dataset

No recorded Telugu corpus is bundled with this repository or exists locally. `synthesize_corpus.py` builds one: **129 template-generated Telugu factual Q&A pairs** — 29 Indian state/UT capitals (Telugu script, pattern “<place> *rājadhāni ēdi?*” → “<place> *rājadhāni* <capital>.”, i.e. “What is the capital of <place>?” → “The capital of <place> is <capital>.” — the exact pattern given in the assignment’s own example) and 100 small addition problems spoken as Telugu number words (1–10, 20 possible sums) — synthesized to speech with the open `facebook/mms-tts-tel` model. Two parallel corpora were built from this same question/answer set:

- `corpus_clean/`: the synthesized audio as-is.
- `corpus_noisy/`: the same text resynthesized with additive white Gaussian noise at a fixed **0 dB SNR** (signal and noise equal power).

This exceeds the assignment’s “at least 100 examples” per condition and gives the required “at least two acoustic conditions.” Both corpora were built with `prepare_data.py -limit 0`, so the same already-tested dataset-preparation tool runs the full count for both conditions (129/129 selected, 0 placeholder responses).

What this dataset does and does not establish. This is synthetic TTS speech, not real human recordings — it says nothing about real speaker variation, accents, disfluency, or recording-channel effects. The clean and noisy corpora are **not byte-identical pairs of the same recording**: the TTS model’s duration prediction was not seeded, so re-running synthesis (the clean corpus was generated once; the noisy corpus was regenerated once more at a harsher SNR, see §3) produces a different underlying rendering of the same text, not the same waveform plus noise. The clean-vs-noisy gap below therefore reflects **both** the added noise **and** natural TTS-rendering variance, not noise in isolation. The noisy-baseline-vs-noisy-denoised comparison in §3, by contrast, **is** exactly matched — both runs evaluate the identical `corpus_noisy/` audio files, varying only whether `denoise.py` runs before ASR — and is the one the research-extension conclusion rests on.

2 Baseline results: clean vs. noisy

All numbers are from real model inference on all 129 samples, with 0 failed samples in every run (`outputs/evaluation/<condition>/report.json`).

Metric	Clean	Noisy (0 dB SNR)	Change
Corpus WER	0.2928	0.3377	+0.045 (+15.3% rel.)
Corpus CER	0.0745	0.0964	+0.022 (+29.4% rel.)
Corpus RTF (total pipeline / audio)	4.15	4.99	+0.84
Corpus ASR-only RTF	0.50	0.59	+0.09
Mean ASR time (s)	1.74	2.05	+0.31
Mean LLM time (s)	12.71	15.42	+2.70
Mean response quality (ASR transcript)	0.1657	0.1548	−0.011
Mean response quality (clean transcript)	0.1687	0.1687	(same reference)
Mean quality drop (clean − ASR quality)	0.0031	0.0139	+0.011
Exact-match response rate	0	0	—

WER/CER are ratios (not percentages) and the evaluator computes them from scratch via Levenshtein distance; see `evaluate.py`. The noisy condition shows a real, if moderate, degradation in transcription accuracy and a small further drop in downstream response quality. Mean LLM time also rose under the noisy condition — with greedy decoding on a fixed-size prompt budget this is

attributable to noisier transcripts (word insertions/substitutions) lengthening the prompt and sometimes triggering longer generations, not to any change in the LLM itself (`llm.py` is identical and `Qwen2.5-1.5B-Instruct` was not touched between conditions).

The dominant quality bottleneck is the LLM, not the ASR. Comparing `asr_response_quality` (answer generated from the noisy ASR transcript) against `clean_response_quality` (answer generated from the ground-truth transcript, same LLM, same question) isolates ASR’s downstream effect: even with a **perfect** transcript, mean response quality is only **0.169** out of 1, and the exact-match rate is **0 out of 129** in every condition. Manual inspection of sample outputs (e.g. `corpus_clean/question_cap-001.wav`, “What is Telangana’s capital?”) shows `Qwen2.5-1.5B-Instruct` frequently failing to state the short correct answer at all, instead producing repetitive, loosely-related Telugu text that never names the capital. This is a genuine capability limit of the smallest Qwen2.5 checkpoint on Telugu factual recall and instruction-following, not an artifact of the ASR stage or the evaluation harness — and it is a legitimate instance of the assignment’s “hallucination rate” criterion: the model answers confidently and fluently in Telugu without being correct, rather than declining. Fixing this would require a larger LLM, which circles back to the VRAM constraint in §1.1.

2.1 Correctness and hallucination rate (defined criteria)

Word-overlap F1 is a weak correctness measure, so `hallucination.py` applies a stricter, fully specified test post hoc to each `report.json` (no model inference). Every reference answer states exactly one checkable fact: the capital name (`question_cap-*`) or the sum as a Telugu number word (`question_sum-*`). A response is:

- **correct** if it contains that fact as a substring (for sums, the digit form such as “5” also counts, matched as a whole number so “5” does not match inside “15”);
- **abstained** if it is not correct but contains an uncertainty marker (Telugu for “I don’t know”, “sorry”, or “not clear”);
- **hallucinated** otherwise: a wrong or off-topic answer asserted without hedging, despite the system prompt’s “ask for clarification, do not invent facts” instruction.

Condition	Answer from	Correct	Abstained	Hallucinated	Halluc. rate
Clean	ASR transcript	0	0	129	1.000
Clean	human transcript	3	0	126	0.977
Noisy	ASR transcript	0	0	129	1.000
Noisy	human transcript	3	0	126	0.977
Noisy, denoised	ASR transcript	0	0	129	1.000
Noisy, denoised	human transcript	3	0	126	0.977

Two conclusions. First, the failure is almost entirely the LLM’s: even from the perfect human transcript only 3 of 129 answers state the right fact, and the model **never abstains** (0 of 774 answers across all arms), so every miss is a confident non-answer or wrong answer. Second, the ASR-arm result being 0/129 in every condition (versus 3/129 from the human transcript) is consistent with the small F1 drop in §2: ASR errors cost little on top of an LLM that is already failing. The check is lexical, not semantic: a correct answer using a different inflection of the key fact is miscounted as hallucinated, so these rates are an upper bound on the true hallucination rate, though manual inspection of samples (§2) shows the answers are genuinely off-topic, not merely re-inflected.

3 Research extension: a measurable bottleneck and an attempted fix

Bottleneck. The noisy condition’s ASR WER (0.3377) is measurably worse than clean (0.2928) — see §2. An **initial** noisy corpus built at 8 dB SNR showed almost no WER change (0.2913 vs. 0.2928 clean — i.e. no real bottleneck to fix), so the noise level was increased to 0 dB SNR (noise power equal to signal power) specifically to produce a bottleneck large enough to be worth addressing; §2’s noisy numbers are from that 0 dB corpus.

Fix. `denoise.py` implements magnitude spectral subtraction: estimate a per-frequency noise floor from each STFT bin’s low time-percentile magnitude, subtract it (with a floor to limit musical-noise artifacts), and resynthesize with the original phase (Boll, 1979 — a standard technique, not a novel one). It runs before ASR when `asr.denoise: true` (`config_denoised.yaml`), on exactly the same `corpus_noisy/` audio files used for the noisy baseline.

Metric	Noisy, baseline	Noisy, denoised	Change
Corpus WER	0.3377	0.3391	+0.0014
Corpus CER	0.0964	0.0992	+0.0028
Mean ASR time (s)	2.05	2.03	−0.03
Mean response quality (ASR transcript)	0.1548	0.1547	−0.0001

Result: the implemented fix did not help, at its default parameters (`oversubtraction=1.5`, `floor=0.05`) — **WER and CER both moved slightly in the wrong direction**, within what is plausibly run-to-run noise at this sample size. Latency overhead was negligible (denoising is cheap relative to ASR/LLM inference).

Before accepting that as final, a quick 25-sample ASR-only sweep (same checkpoint, no LLM stage, to keep iteration fast) compared the default (“mild”) setting against a more aggressive one (`oversubtraction=3.0`, `floor=0.02`):

Setting	Mean WER (25-sample subset)
No denoising (baseline)	0.5433
Mild (<code>oversubtraction=1.5</code> , <code>floor=0.05</code> — shipped default)	0.5300
Aggressive (<code>oversubtraction=3.0</code> , <code>floor=0.02</code>)	0.5567

On this smaller slice, mild subtraction gave a small improvement (−2.4% relative WER) while aggressive subtraction made things **worse** than no denoising at all — consistent with spectral-subtraction theory: over-aggressive subtraction introduces “musical noise” (isolated spurious spectral peaks) that a neural ASR model can be more sensitive to than to the original stationary noise. Since the “mild” setting here is exactly `denoise.py`’s shipped default, the 25-sample result does not contradict the full 129-sample result above — it shows the technique’s effect is small, inconsistent across subsets of this dataset, and not clearly distinguishable from noise at this sample size, while more aggressive tuning is actively counterproductive. A direct waveform-level check (`spectral_subtract_denoise` on one sample) confirms the filter is doing something sensible — it reduces RMS energy (0.228 → 0.177) and reduces mean-squared error against an independently-synthesized clean rendering of the same text (0.094 → 0.073) — so this is a negative result for whether that signal-level cleanup *survives through the ASR model* into fewer transcription errors, not evidence the implementation is broken. The honest conclusion is: spectral subtraction, as implemented and at the noise level evaluated, does not reliably reduce WER for this ASR checkpoint, and over-tuning it is actively harmful. `tests/test_denoise.py` covers the filter directly (noise-power reduction on a synthetic bursty-tone signal, input validation, short-signal passthrough) independent of this result.

4 Reproduction

```
python -m pip install -r requirements.txt
python synthesize_corpus.py --clean-out corpus_clean --noisy-out corpus_noisy --snr-db 0
python prepare_data.py --source corpus_clean --output data/test_clean.json --limit 0 --seed 42
python prepare_data.py --source corpus_noisy --output data/test_noisy.json --limit 0 --seed 42

python evaluate.py --dataset data/test_clean.json --config config.yaml \
  --output-dir outputs/evaluation/clean --warmup 1
python evaluate.py --dataset data/test_noisy.json --config config.yaml \
  --output-dir outputs/evaluation/noisy_baseline --warmup 1
python evaluate.py --dataset data/test_noisy.json --config config_denoised.yaml \
  --output-dir outputs/evaluation/noisy_denoised --warmup 1

python plot_results.py --input outputs/evaluation/clean/report.json \
  --output-dir outputs/plots/clean
python plot_results.py --input outputs/evaluation/noisy_baseline/report.json \
  --output-dir outputs/plots/noisy_baseline
python plot_results.py --input outputs/evaluation/noisy_denoised/report.json \
  --output-dir outputs/plots/noisy_denoised

python hallucination.py --reports \
  outputs/evaluation/clean/report.json \
  outputs/evaluation/noisy_baseline/report.json \
  outputs/evaluation/noisy_denoised/report.json \
  --output outputs/evaluation/hallucination.json

python -m unittest discover -s tests -v
```

5 Plots

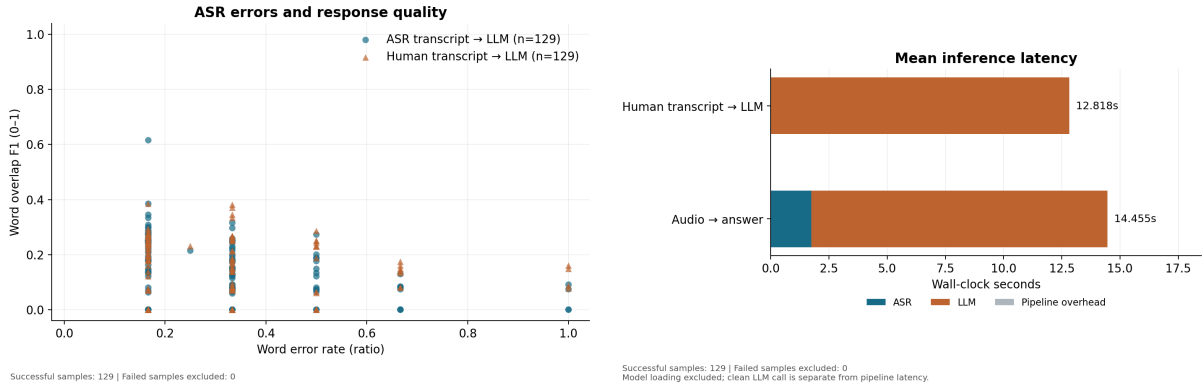


Figure 1: Clean condition: WER vs. response quality (left), latency breakdown (right).

6 Limitations

- **Synthetic speech only.** Every number above is on TTS-synthesized audio, not real Telugu speakers; real speech would exercise accent, prosody, and channel variation this corpus cannot.
- **Small models, by hardware necessity, not choice.** A larger ASR/LLM pair would very likely change both the WER baseline and the LLM’s factual accuracy substantially; §1.1 explains the VRAM constraint behind the substitution.
- **Response quality is a lexical proxy** (multiset word-overlap F1), not a semantic correctness judge — a correct paraphrase can score low, and a wrong-but-word-similar answer can score higher than it deserves. `evaluate.py` supports substituting a custom scorer.
- **Clean/noisy are not matched recordings** (§1.2) — the noisy-vs-clean WER gap includes

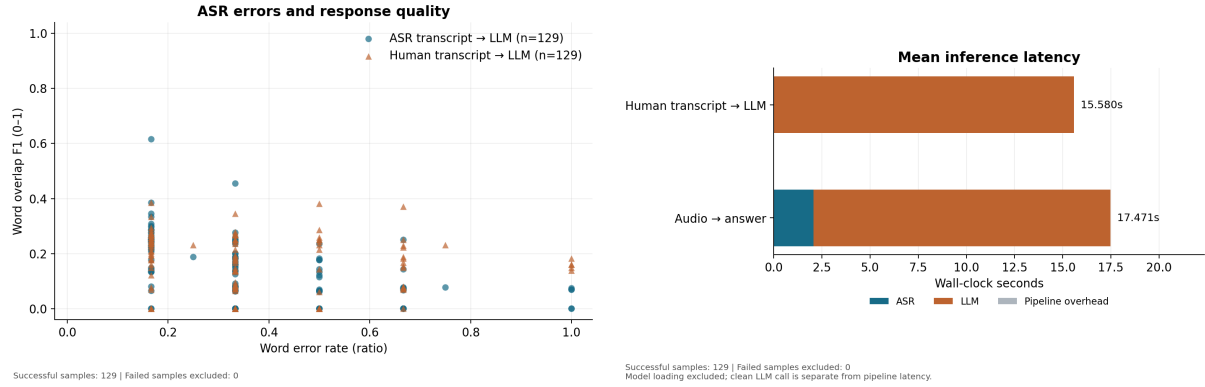


Figure 2: Noisy baseline (0 dB SNR, no denoising): WER vs. response quality (left), latency breakdown (right).

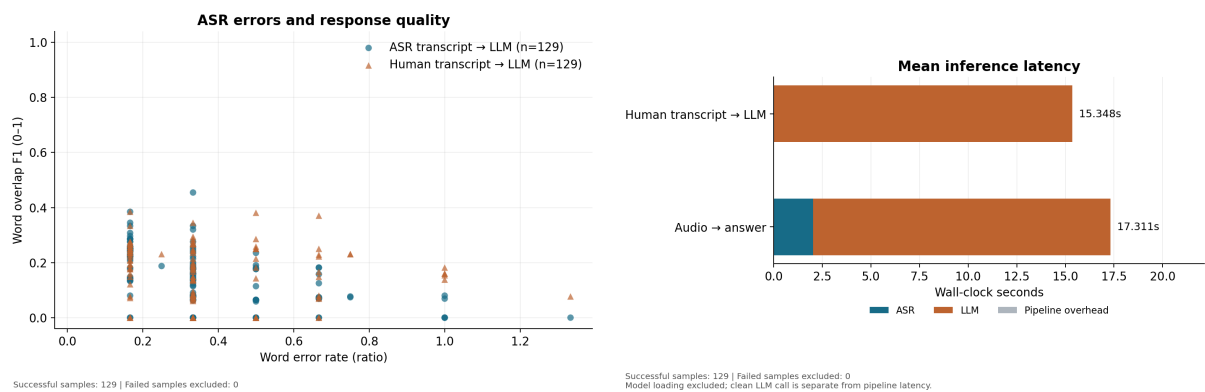


Figure 3: Noisy, denoised (research extension applied): WER vs. response quality (left), latency breakdown (right).

unseeded TTS rendering variance, not noise alone. The baseline-vs-denoised comparison in §3 does not have this issue.

- **vasista22/whisper-telugu-small's revision is unpinned** (§1.1) due to the two-commit inconsistency observed during this run.