AALTO UNIVERSITY

SPEECH TRANSCRIPTION SERVICE FOR NORTHERN SAMI

ASR Group

Shen Jingwen (102787645)

School of Science
August 26, 2025

Contents

1 Project Overview

This project implements a Northern Sami speech transcription service. Frontend built with Gradio Blocks, provides audio upload, model selection, transcript display, editing, and saving. It supports offline transcription with word-level timestamp alignment and audio-text highlighting. Users can switch between different ASR models, and long audio is processed in chunks for segmented output.

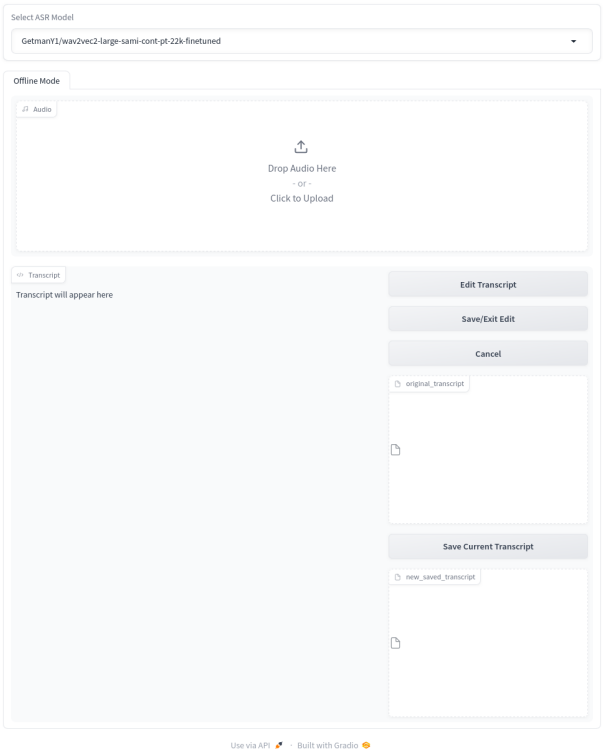

Figure 1: Transcription Service Interface

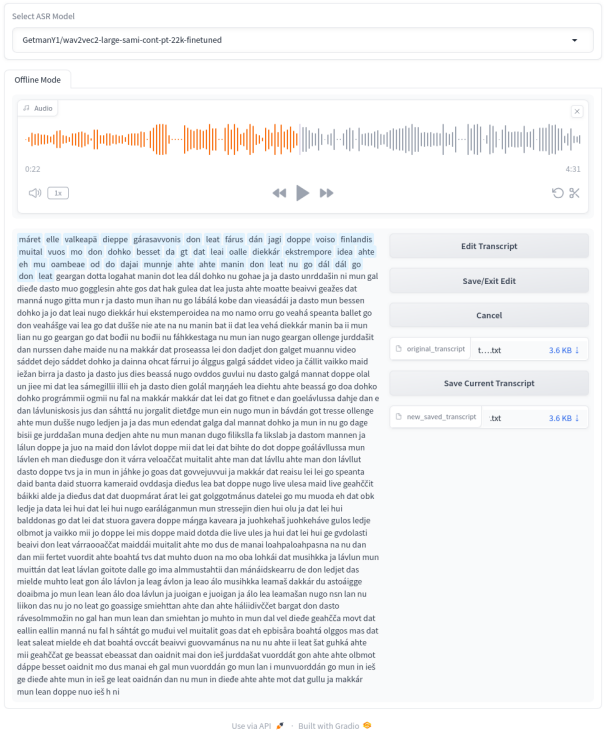

Figure 2: Output View

2 Key Features and Implementation

2.1 Speech Recognition and Model Switching

- Supports two fine-tuned Northern Sami ASR models [1] [2].
- Select models, and the system will automatically re-transcribe uploaded audio.

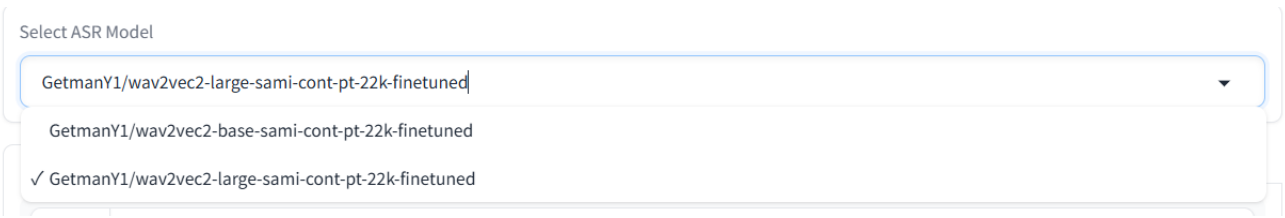

Figure 3: Select ASR Models (user interface)

2.1.1 Backend Code Implementation

- Implemented in custom_asr.py & asr_instance.py. Called in app.py.

- Uses HuggingFace's pipeline("automatic-speech-recognition") for loading models in custom_asr.py.
- retranscribe_on_model_change() in app.py called function change_model() in custom_asr.py. Triggered by model_selector.change in app.py.

2.2 Offline Audio Transcription (chunk-based streaming)

- Upload audio files to be processed by the ASR model into text.
- The transcript is generated in segments and updated every second, while streaming pipeline processes each 10-second chunk (about 1s processing per chunk).

2.2.1 Backend Code Implementation

- Chunk pipeline processed in custom_asr.py. Called in app.py.
- Implements overlap detection and removal to avoid duplicate words caused by chunking. (CustomAutomaticSpeechRecognizer.transcribe(self, stream, result_queue))

2.3 Transcript Editing and Saving

- Press "Edit" button to enter edit mode, modify transcript text.
- Press "Save/Exit Edit" button to exit edit mode and save.
- Press "Cancel" button to cancel and exit edit mode.
- Press "Save Current Transcript" button to save edited transcript, each press will create a .txt file in new_saved_transcript box, click the blue arrow on the right to download file.
- The "original_ranscript" box automatically saves original transcript when switching models or uploading new audio. Click the blue arrow to download.

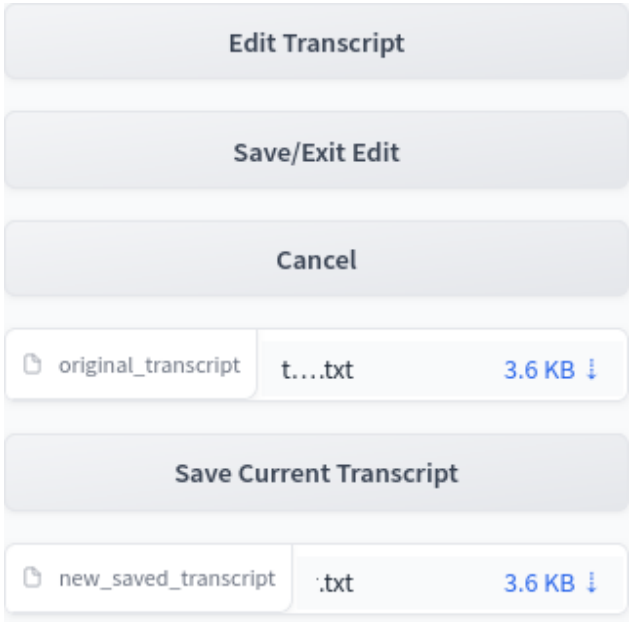

Figure 4: Buttons of Edit/Cancel/Save (user interface)

2.3.1 Backend Code Implementation

- Implemented in app.py based on Gradio, called tool functions in util/audio_transcript_utils.py.
- create_transcript_ui() for UI display, bind_edit_buttons() for click event, save_edited_transcript() and save_original_transcript() for saving (both called function save_transcript in util/audio_transcript_utils.py)

2.4 Alignment and Highlighting

During audio playback, the corresponding words in the transcript are automatically highlighted.

- Alignment implemented in util/forced_alignment.py using CTC segmentation algorithm [3], called by app.py. Audio processing use Wav2Vec2 model with Northern Sami language support. Produces word-level timestamps with confidence scores.
- Highlighting implemented in util/audio_transcript_utils.py, called by app.py.
- JavaScript triggers Python backend to update highlights based on current audio playback position (hidden_trigger.click, audio_input.play, played_duration.change for triggers in app.py) and transcript alignment timestamps (process_audio_with_timestamps() in util/audio_transcript_utils.py). Updates every second after playback starts (js in audio_input.play in app.py), with a simple adjustment to fix 2s processing delay (in function update_highlight() of audio_transcript_utils.py, called by update_text_highlight in app.py).

2.5 Punctuation and Capitalization (Commented Out)

Adds punctuation and capitalization to transcript text to improve readability.

- Current implementation (in utils/punctuation_restorer.py, called in custom_asr.py).
- Tried two different approaches: a multilingual punctuation model (oliverguhr/fullstop-punctuation-multilang-large) and NeMo's punctuation-capitalization models (punctuation_en_bert and punctuation_en_distilbert). However, both models demonstrated limited effectiveness when processing Northern Sami transcripts.

2.6 Play-from-Text Function (Partially Implemented)

Double-click any word in the transcript to jump to the corresponding position in the audio.

- Implemented in app.py, called tool functions in util/audio_transcript_utils.py.
- **Frontend Event Handling to Capture Double Clicked Word (Done):** JavaScript event listeners are attached to both the HTML transcript display and textbox edit mode. (js in audio_input.change and word_click_trigger in app.py to capture target word (double clicked word) and one previous and one next word for precise position, use t_hidden_btn to store word infomations for str_time.change to update target word timestamps).
- **Got Timestamp/Position of Double Clicked Word (Done):** The get_word_timestamp() function in audio_transcript_utils.py resolves word to timestamp using contextual matching. As shown in Figure 5, target word "geargan" got two in transcript and use one previous and one next word for precise position to second one. (get_word_timestamp() return word_timestamp and str_time, one is float time position in audio, one is string time format corresponding to UI display of played time shows in Figure 5 left under audio progress bar, functions seconds_to_time_str() and time_str_to_seconds() convert time format).

- **Audio Progress Bar Control (Unfinished):** The current challenge involves programmatically triggering the Gradio audio component's progress bar click event to achieve precise jump-to-position functionality. **While the frontend click event code has been identified (as shown in the blue box in Figure 5), how to call this click event remains under investigation.** Once achieved, the implementation will trigger the play button click event (already done) to initiate playback from the target word's timestamp. **(implemented in app.py, str time.change, JS code under testing)**.

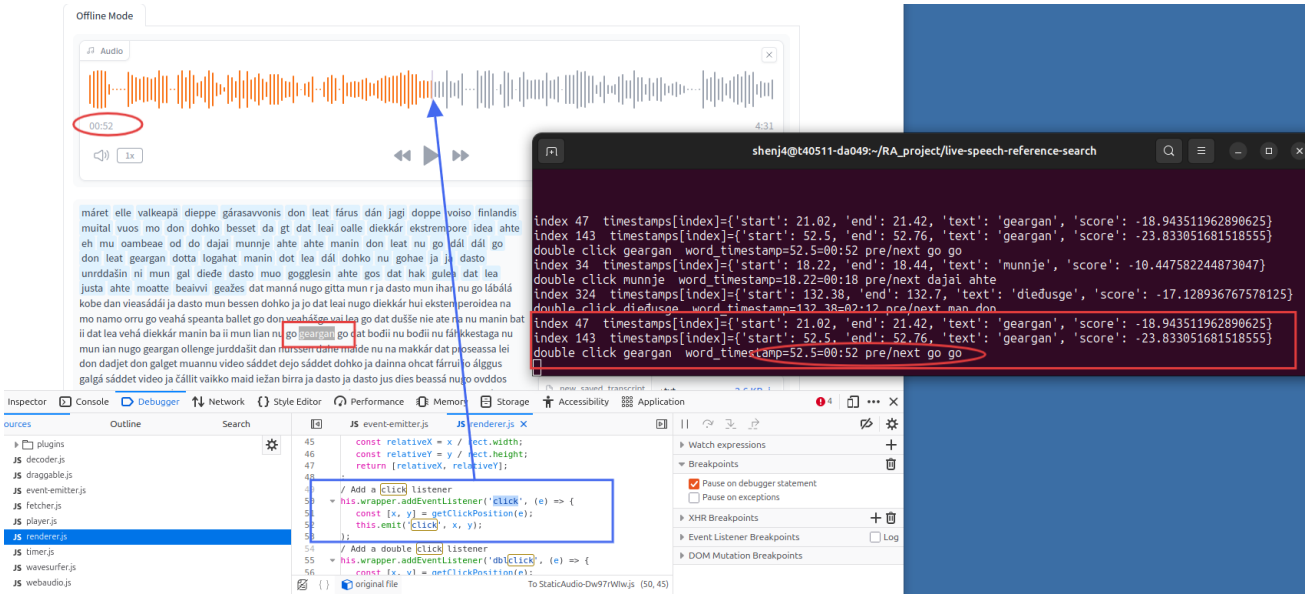

Figure 5: Buttons of Edit/Cancel/Save (user interface)

3 Conclusion

This project implements a basic speech-to-text transcription system that covers audio input, model management, transcript generation, word-level alignment, interactive visualization, and saving results. With chunk-based streaming transcription and synchronized highlightinbg, it provides an intuitive way for users to review and refine recognition results, making it useful for speech analysis on Northern Sami language.

4 Next Steps

- **Punctuation and Capitalization:** Current implementation (in utils/punctuation_restorer.py, called in custom_asr.py) is commented out due to poor generalization. **Future plan** could be to train a custom model on Northern Sami text data to improve performance.
- **Audio-Text Synchronization (highlighting):** Highlighting implemented in utils/audio_transcript_utils.py and updates every second after playback starts (with a simple adjustment to fix 2s delay - works without visible lag but still can be improved if we have better solution).
- **Audio-Text Synchronization (alignment):** Current alignment implemented in utils/forced_alignment.py [3] using forward pass. **Future plan** could be to integrate NVIDIA NeMo alignment [4] for better precision.

- **Play-from-Text Function:** Double-clicking a word of transcript already retrieves its timestamp and logs it (combined with one previous and one next word to position). **Future plan** is to use this timestamp to trigger the Gradio audio component's progress bar (implemented in app.py, str_time.change, JS code under testing).
- **Editing Mode:** Currently shows plain text without highlighting. Cancel/Save to exit editing can return correct highlighting html transcript. **Future plan** is to preserve edit mode with highlight effects if possible.
- **Service Stability:** Occasionally throws errors — workaround is to close the popup or refresh. Need to fix errors and improve stability if time available.

5 System Deployment and Operation

The service currently on cPuota and configured a floating IP for SSH remote debugging (ubuntu@195.148.31.79)

- **Code Location:** /home/ubuntu/live-speech-reference-search
- **Working Directory:** live-speech-reference-search
- **Activate the Conda environment [5]:** conda activate live_speech_0
- **Start/Restart the service in daemon mode:** ./start.sh –daemon
- **Logging and Debugging:** tail -f log.txt

After starting the service, use commend of Logging and Debugging to check log.txt:

- Obtain the public access link for the service
- Monitor system status and operation logs
- Debug any runtime issues

The log file provides real-time operational feedback and contains the publicly accessible URL once the service is successfully deployed. (72h each time, need Restart the service after 72h to get a new available one)

References

[1] Yaroslav Getman. *Sámi Wav2vec2-Base ASR*. URL: https://huggingface.co/GetmanY1/wav2vec2-base-sami-cont-pt-22k-finetuned.

[2] Yaroslav Getman. *Sámi Wav2vec2-Large ASR*. URL: https://huggingface.co/GetmanY1/wav2vec2-large-sami-cont-pt-22k-finetuned.

[3] MahmoudAshraf97. *Forced Alignment with Hugging Face CTC Models*. URL: https://github.com/MahmoudAshraf97/ctc-forced-aligner/tree/main.

[4] NVIDIA. *NeMo Forced Aligner (NFA)*. URL: https://github.com/NVIDIA/NeMo/tree/main/tools/nemo_forced_aligner.

[5] SubtleParesh. *Live Speech Reference Search*. URL: https://github.com/SubtleParesh/live-speech-reference-search.